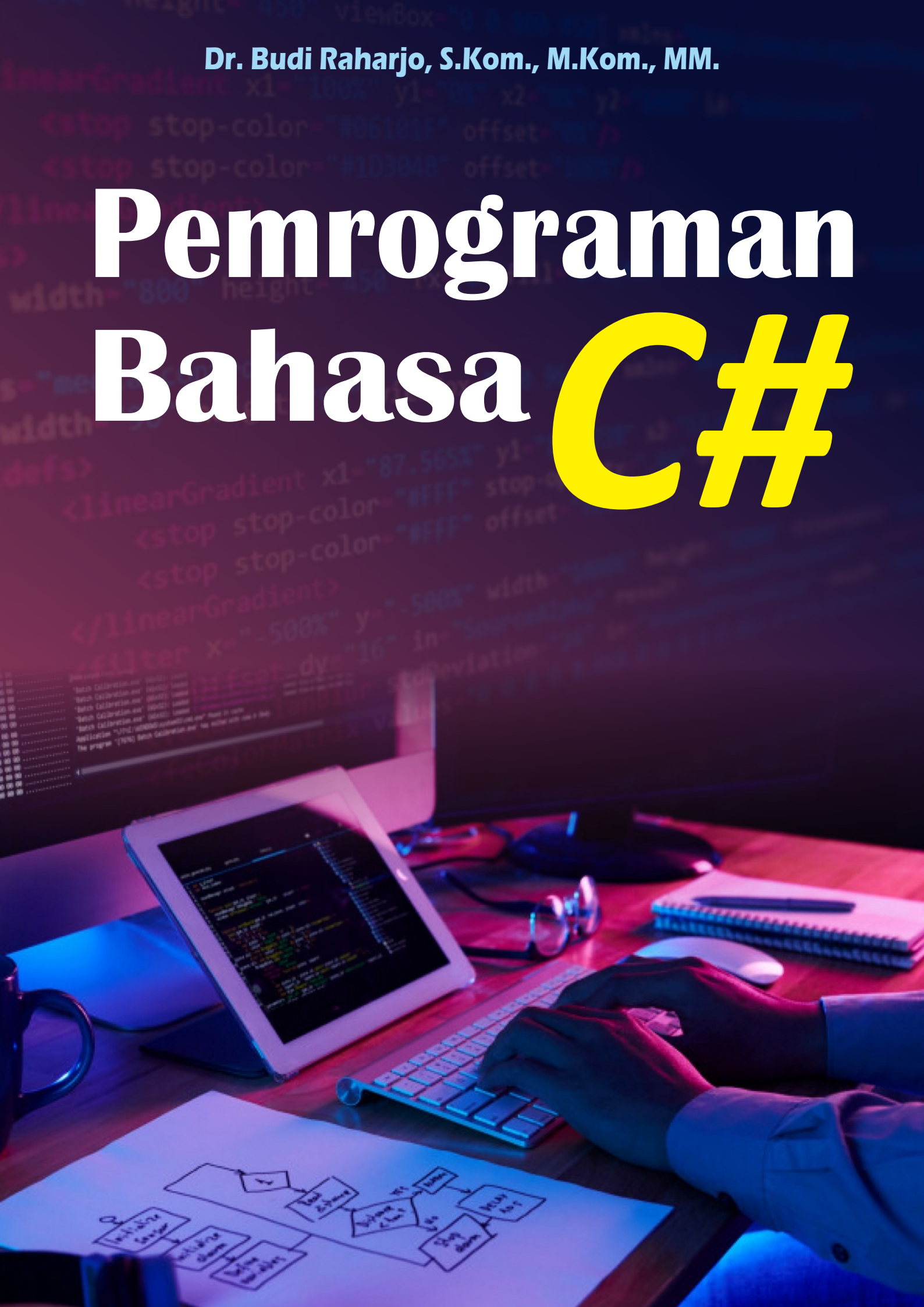
Dr. Budi Raharjo, S.Kom., M.Kom., MM.

Pemrograman Bahasa C#



Pemrograman Bahasa C#

Penulis :
Dr. Budi Raharjo, S.Kom., M.Kom., MM.

ISBN : 9 786235 734682

Editor :
Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

Penyunting :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom.

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Puji syukur kami panjatkan atas kehadirat Tuhan karena buku yang berjudul ***"Pemrograman Bahasa C#"*** dapat terselesaikan. C# adalah bahasa pemrograman sederhana yang digunakan secara umum, dalam artian bahasa pemrograman ini mampu dipergunakan untuk berbagai macam fungsi misalnya untuk pemrograman *server-side* pada website, aplikasi mobile ataupun desktop, dan sebagainya. Selain itu C# juga bahasa pemrograman yang berorientasi objek, maksudnya, Bahasa Pemrograman C# juga mengusung konsep objek seperti *inheritance, class, polymorphism* dan *encapsulation.*

Buku ini cocok digunakan bagi pemula yang ingin belajar maupun mengenal bahsa Pemrograman C#. Buku ini terbagi menjadi 11 Bab, Bab 1 berupa pengenalan atau pengantar Bahasa C#. sedangkan bab 2 akan membahas tentang instalasi software Visual Studio Community. Ini adalah software dimana pembaca akan membuat project menggunakan Bahasa C# dan mengenal struktur dasar pemrograman. Bab 3 akan menerangkan tentang Variabel-variabel yang digunakan di Bahasa pemrograman ini dan mengenal macam operator Bahasa C#. Pada bab selanjutnya, bab 4, akan menjelaskan tentang type data Array String dan Daftar (List). Bab selanjutnya yaitu bab 5 membahas tentang menerima input dari pengguna, bagaimana mengolahnya dan menyimpan data dari Variabel. Bab 6 Secara khusus akan belajar tentang pernyataan if, pernyataan if sebaris, pernyataan switch, perulangan for, perulangan foreach, perulangan while dan perulangan do while.

Pada 2 bab selanjutnya yaitu bab 7 dan 8 akan membahas tentang pemrograman berorientasi obyek dan menuliskan kelas kita sendiri dari obyek tersebut. Bab 9 membahas tentang dua tipe data yang ditentukan pengguna di C# yaitu enum dan struktur. Bab 10 akan membahas tentang pengenalan *Language-Integrated Query* atau LINQ. Bab 11 sekaligus menjadi bab terakhir dari buku ini akan mempelajari cara membaca dan menulis ke file eksternal. Akhir kata semoga buku ini bermanfaat bagi para pembaca.

Semarang, April 2022
Penulis

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

DAFTAR ISI

BAB 1
PENGANTAR C#

Selamat datang di pemrograman C# dan terima kasih banyak telah membaca buku ini! Apakah Anda seorang programmer berpengalaman atau pemula, buku ini ditulis untuk membantu Anda mempelajari pemrograman C# dengan cepat. Topik dipilih dengan cermat untuk memberi Anda paparan luas terhadap C# tanpa membebani Anda dengan informasi yang berlebihan. Pada akhir buku ini, Anda seharusnya tidak memiliki masalah dalam menulis program C# Anda sendiri. Sebenarnya, kita akan mengkodekan perangkat lunak penggajian sederhana bersama-sama sebagai bagian dari proyek di akhir buku ini. Siap untuk mulai? Pertama, mari kita jawab beberapa pertanyaan:

1.1 APA ITU C#?

C#, dikatakan sebagai C Sharp, merupakan bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft pada awal 2000-an, dipimpin oleh Anders Hejlsberg. Sebuah bagian dari kerangka .Net dan dimaksudkan untuk menjadi bahasa pemrograman tujuan umum sederhana yang dapat digunakan untuk mengembangkan berbagai jenis aplikasi, termasuk aplikasi konsol, windows, web, dan seluler. Seperti semua bahasa pemrograman modern, kode C# menyerupai bahasa Inggris yang tidak dapat dipahami oleh komputer. Oleh karena itu, kode C# harus diubah menjadi bahasa mesin menggunakan apa yang dikenal sebagai compiler (lihat catatan kaki). Kompiler yang akan kita gunakan dalam buku ini adalah Visual Studio Community 2015 gratis yang disediakan oleh Microsoft.

1.2 MENGAPA BELAJAR C#?

C# memiliki sintaks dan fitur yang menyerupai bahasa pemrograman lain seperti Java dan C++. Dengan demikian, jika Anda memiliki pengalaman pemrograman sebelumnya, Anda akan menemukan bahwa belajar C# sangat mudah. Bahkan jika Anda benar-benar baru dalam pemrograman, C# dirancang agar mudah dipelajari (tidak seperti C atau C++) dan merupakan bahasa pertama yang bagus untuk dipelajari. Selain itu, C# adalah bagian dari kerangka .Net. Kerangka kerja ini mencakup perpustakaan besar kode pra-tertulis yang dapat digunakan oleh programmer tanpa harus menulis semuanya dari awal. Hal ini memungkinkan pemrogram untuk dengan cepat mengembangkan aplikasi mereka dalam C#, menjadikan C# bahasa yang ideal untuk digunakan jika Anda memiliki jadwal yang ketat.

Terakhir, C# adalah bahasa pemrograman berorientasi objek (OOP). Pemrograman berorientasi objek adalah pendekatan pemrograman yang memecah masalah pemrograman menjadi objek yang berinteraksi satu sama lain. Kita akan melihat berbagai konsep pemrograman berorientasi objek dalam buku ini. Setelah Anda menguasai C#, Anda akan terbiasa dengan konsep-konsep ini. Ini akan memudahkan Anda untuk menguasai bahasa pemrograman berorientasi objek lainnya di masa mendatang.

Siap untuk terjun ke dunia pemrograman C#? Mari kita mulai.

Catatan kaki:

Konversi program C# ke dalam bahasa mesin sebenarnya sedikit lebih rumit dari ini. Komunitas Visual Studio hanya mengubah program Bahasa C menjadi MIL, yang merupakan singkatan dari Microsoft Intermediate Language. Kode MIL ini kemudian diubah menjadi bahasa mesin oleh sistem eksekusi virtual yang dikenal sebagai Common Language Runtime. Untuk informasi lebih lanjut, Anda dapat melihat https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx. Meskipun demikian, untuk tujuan kita, kita tidak perlu mengetahui detail rumit ini untuk mengembangkan program C# kita sendiri.

BAB 2
BERSIAP UNTUK BAHASA C MENGINSTAL *VISUAL STUDIO COMMUNITY*

2.1 MENGINSTAL *VISUAL STUDIO COMMUNITY*

Sebelum kita mulai mengembangkan aplikasi di C#, kita perlu mengunduh *Visual Studio Community.* Seperti disebutkan dalam Bab 1, *Visual Studio Community* (VSC) adalah kompiler gratis yang disediakan oleh Microsoft. Faktanya, VSC lebih dari sekedar compiler. Ini adalah Lingkungan Pengembangan Terpadu (IDE) yang menyertakan editor teks untuk kita menulis kode kita dan debugger untuk membantu kita mengidentifikasi kesalahan pemrograman.

Untuk mengunduh VSC, kunjungi https://www.visualstudio.com/en-us/products/visualstudio-community-vs.aspx.

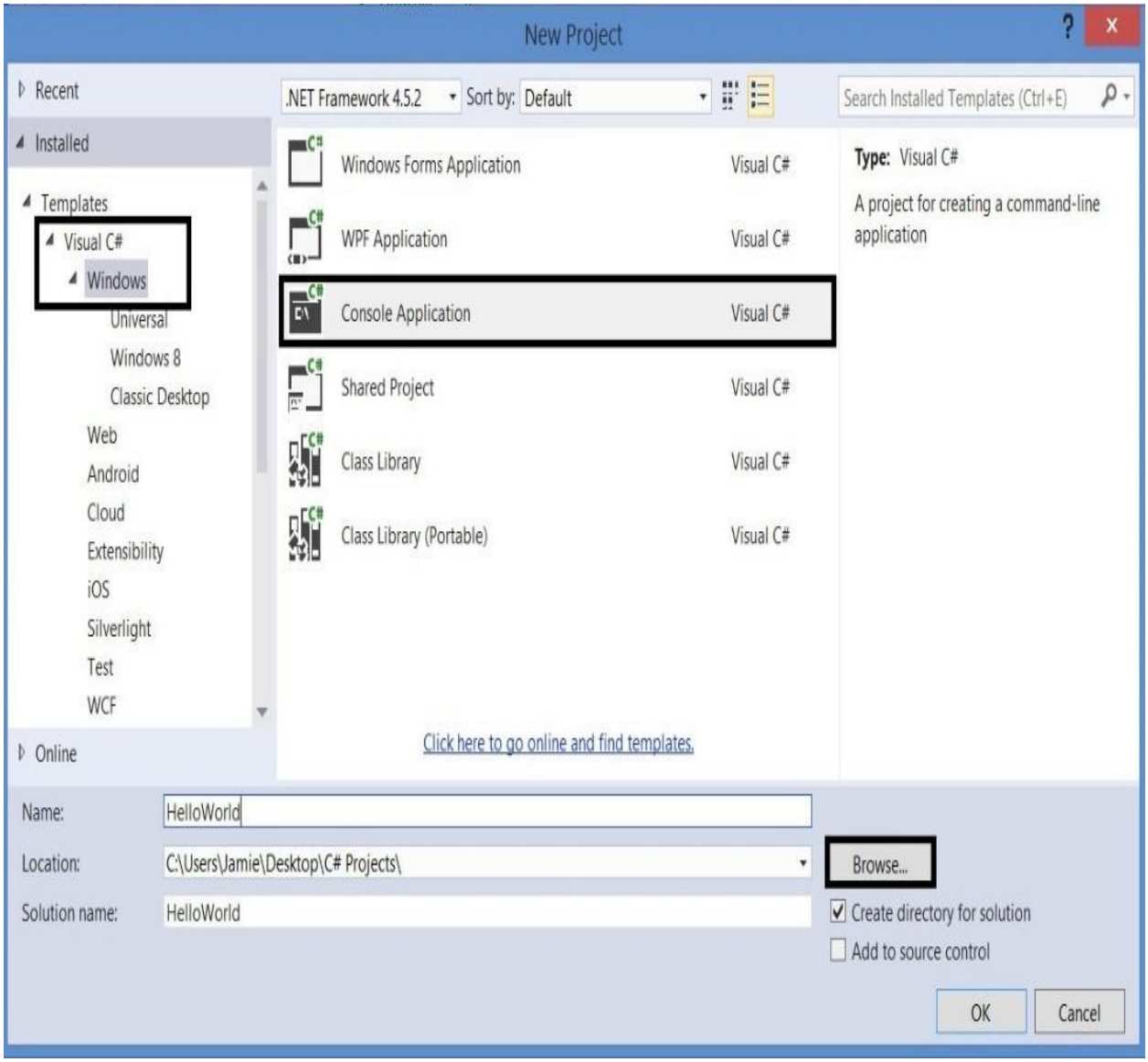
Gambar 2.1 Tombol untuk mengunduh Visual Studio Community

Klik tombol "Unduh Komunitas 2015" untuk mengunduh file. Setelah Anda mengunduh file, klik dua kali untuk menginstal VSC. Setelah Anda menginstal, Anda akan diminta untuk me-restart komputer Anda. Setelah Anda me-restart komputer Anda, Anda siap untuk mulai mengkode program pertama Anda.

2.2 PROGRAM C# PERTAMA ANDA

Untuk menulis program pertama kita, mari buat folder di desktop kita dan beri nama "C# Projects". Kita akan menyimpan semua proyek C# kita ke folder ini. Selanjutnya, luncurkan VSC dan pilih File > New > Project…. (Anda mungkin harus mencari "Visual Studio 2015" jika Anda tidak dapat menemukan VSC.) Program pertama yang akan kita tulis adalah aplikasi konsol. Aplikasi konsol mengacu pada program yang tidak memiliki antarmuka pengguna grafis.

Dari kotak dialog Proyek Baru, pilih "Visual C# > Windows" (di sebelah kiri) dan pilih "Aplikasi Konsol" di kotak utama. Beri nama program ini "HelloWorld" dan simpan di folder "C# Projects" yang dibuat sebelumnya. Anda dapat menggunakan tombol “Browse…” untuk menelusuri folder yang benar.
Terakhir, klik OK untuk membuat proyek.



Gambar 2.2 Jendela VSC untuk membuat projek baru

Anda akan disajikan dengan template default yang dibuat VSC untuk Anda secara otomatis.

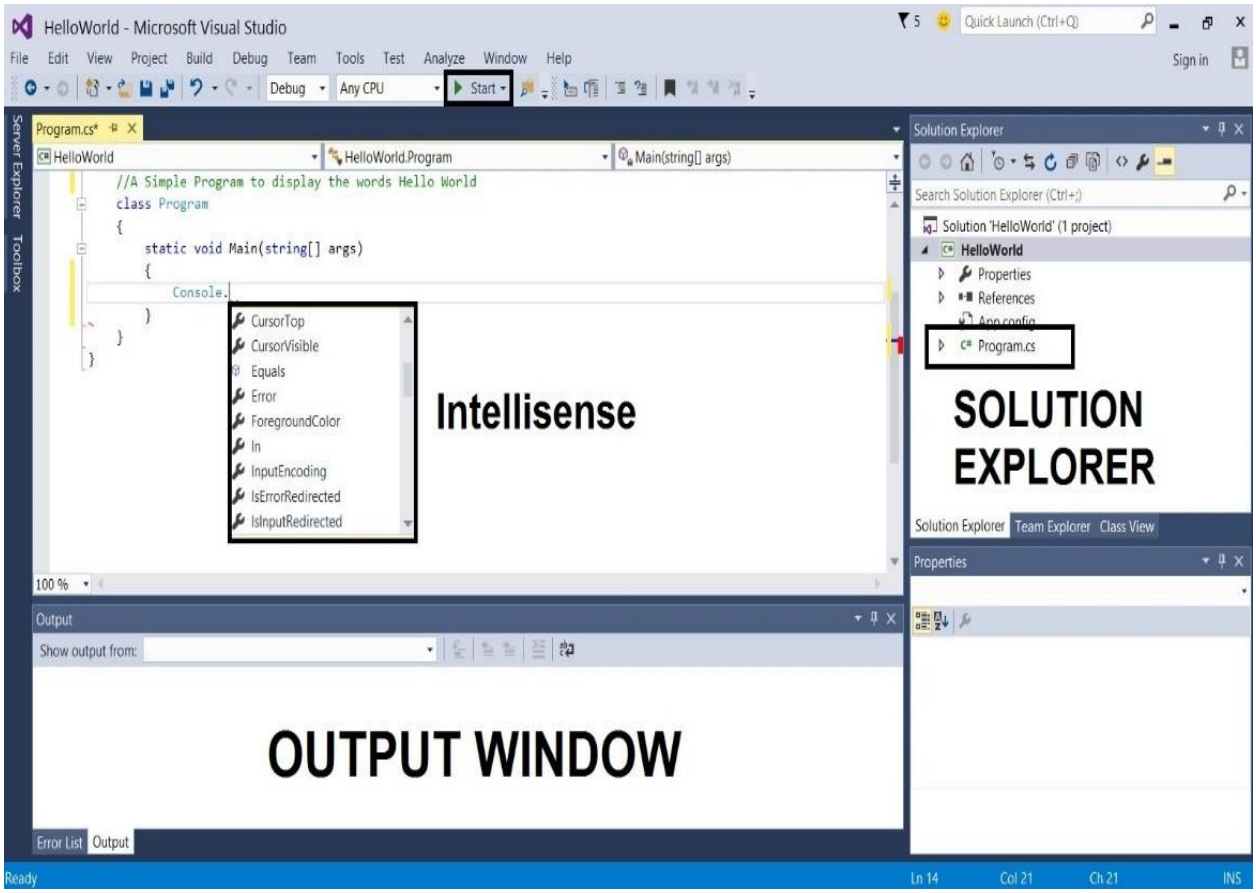
Ganti kode di template dengan kode di bawah ini. Perhatikan bahwa nomor baris ditambahkan untuk referensi dan bukan bagian dari kode yang sebenarnya. Anda mungkin ingin menandai halaman ini untuk referensi mudah nanti ketika kita membahas program ini. Karena ukuran layar yang kecil di sebagian besar perangkat seluler, kode mungkin terlihat campur aduk jika Anda melihatnya di Kindle, tablet, atau ponsel. Jika Anda mengalami masalah dalam membaca kode, Anda dapat mencoba mengubah layar ke mode lanskap. Atau, Anda

dapat mengunduh kode sumber untuk contoh program ini dan semua contoh program lain dalam buku ini di http://www.learncodingfast.com/csharp.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace HelloWorld
8 {
9
10 //A Simple Program to display the words Hello World
11
12 class Program
13 {
14 static void Main(string[] args)
15 {
16 Console.WriteLine("Hello World");
17 Console.Read();
18 }
19 }
20 }
```

Saya sangat menganjurkan agar Anda mengetik sendiri kodenya untuk lebih memahami cara kerja VSC. Saat Anda mengetik, Anda akan melihat bahwa sebuah kotak muncul di dekat kursor dengan beberapa pesan bantuan sesekali. Itu dikenal sebagai Intellisense. Misalnya, saat Anda mengetik titik (.) setelah kata "Konsol", daftar drop-drop akan muncul untuk memberi tahu Anda apa yang bisa Anda ketik setelah titik. Ini adalah salah satu fitur dari VSC untuk membantu membuat coding lebih mudah bagi programmer. Setelah Anda selesai mengetik, Anda dapat menjalankan program ini dengan mengklik tombol "Start" di menu atas (lihat gambar di bawah).

Jika program Anda gagal dijalankan, VSC akan memberi tahu Anda tentang kesalahan di "Jendela Output". Mengklik dua kali pada kesalahan memindahkan kursor ke tempat kesalahan itu. Periksa kembali kode Anda terhadap kode di atas untuk memperbaiki kesalahan dan jalankan program lagi.



Gambar 2.3 Jendela output untuk mengetahui letak kesalahan program

Jika semuanya berjalan dengan baik dan program Anda berjalan dengan sukses, sebuah jendela hitam akan muncul dengan tulisan “Hello World” berwarna putih. Jendela hitam ini dikenal sebagai konsol. Tekan Enter untuk menutup jendela. Itu dia! Anda telah berhasil mengkodekan program pertama Anda. Beri diri Anda tepukan di bahu. Jika Anda menavigasi ke folder "C# Projects" Anda sekarang, Anda akan menemukan folder bernama "Halo Dunia". Di dalam folder, Anda akan menemukan folder "HelloWorld" lain dan file "HelloWorld.sln". File .sln ini adalah file solusi. Kapan pun Anda perlu membuka kembali proyek, ini adalah file yang harus dibuka. Jika editor teks tidak menampilkan kode Anda saat Anda membuka file solusi, cukup klik dua kali pada file “Program.cs” di “Solution Explorer” di sebelah kanan (lihat gambar sebelumnya) untuk membukanya. File yang dapat dieksekusi (.exe) dari kode Anda dapat ditemukan di folder HelloWorld > HelloWorld > bin > Debug.

2.3 STRUKTUR DASAR PROGRAM C#

Sekarang, mari kita lakukan run-through singkat dari program dasar yang baru saja Anda kodekan.

Pengarahan

Dari baris 1 sampai 5, kita memiliki beberapa pernyataan yang dimulai dengan kata using. Pernyataan-pernyataan ini dikenal sebagai direktif. Mereka memberi tahu kompiler bahwa program kita menggunakan namespace tertentu.

Misalnya, baris pertama

using System;

memberitahu kompiler bahwa program kita menggunakan ruang nama System.

Namespace

Namespace hanyalah pengelompokan elemen kode terkait. Elemen-elemen ini termasuk kelas, antarmuka, enum dan struct dll (kita akan membahas masing-masing elemen ini di bab berikutnya).

C# hadir dengan sejumlah besar kode pra-tertulis yang diatur ke dalam ruang nama yang berbeda. Namespace System berisi kode untuk metode yang memungkinkan kita berinteraksi dengan pengguna kita. Kita menggunakan dua metode ini dalam program kita - metode WriteLine()dan Read(). Ruang nama lain tidak diperlukan dalam program kita. Namun, karena ruang nama ini disertakan dalam template default, kita akan membiarkannya dalam kode kita.

Selain ruang nama pra-tertulis yang disediakan oleh Microsoft, kita juga dapat mendeklarasikan ruang nama kita sendiri. Satu keuntungan mendeklarasikan namespace adalah mencegah konflik penamaan. Dua atau lebih elemen kode dapat memiliki nama yang sama selama mereka termasuk dalam ruang nama yang berbeda. Misalnya, kode di bawah ini mendefinisikan dua ruang nama, keduanya berisi kelas bernama MyClass. Ini diperbolehkan di C# karena kedua kelas memiliki ruang nama yang berbeda (First dan Second).

```
namespace First
{
    class MyClass
    {
    }
}

namespace Second
{
    class MyClass
    {
    }
}
```

Dalam contoh kita, kita mendeklarasikan satu namespace - HelloWorld.

Namespace HelloWorld dimulai pada baris 7, dengan kurung kurawal pembuka pada baris 8. Ini berakhir pada baris 20 dengan kurung kurawal penutup. Kurung kurawal digunakan secara ekstensif dalam C# untuk menunjukkan awal dan akhir elemen kode. Semua kurung kurawal di C# harus ditutup dengan kurung kurawal yang sesuai. Dalam namespace HelloWorld, kita memiliki kelas Program yang dimulai pada baris 12 dan berakhir pada baris 19. Di dalam kelas Program, kita memiliki metode Main() yang dimulai pada baris 14 dan berakhir pada baris 18.

Metode Main()

Metode Main() adalah titik masuk dari semua aplikasi konsol C#. Setiap kali aplikasi konsol dijalankan, metode Main() adalah metode pertama yang dipanggil.

Dalam buku ini, setiap kali Anda diminta untuk mencoba segmen kode tertentu, Anda harus membuat "Aplikasi Konsol" baru dan mengetikkan segmen kode yang diberikan ke dalam metode Main() (di antara kurung kurawal). Anda kemudian dapat menjalankan

program untuk menguji kode. Perhatikan kata-kata “string[] args” di dalam kurung metode Main()kita? Ini berarti metode Main() dapat mengambil array string sebagai input. Jangan khawatir tentang ini untuk saat ini. Kita akan membahas topik-topik ini di bab-bab berikutnya. Dalam contoh kita, metode Main() berisi dua baris kode. Baris pertama Console.WriteLine(“Hello World”);

menampilkan baris "Hello World" (tanpa tanda kutip) di layar.

Baris kedua
Console.Read();

menunggu penekanan tombol dari pengguna sebelum menutup jendela.

Kedua pernyataan di atas diakhiri dengan titik koma. Ini umum untuk sebagian besar pernyataan dalam C#. Setelah pernyataan Console.Read(), kita mengakhiri kode kita dengan tiga kurung kurawal untuk menutup kurung kurawal sebelumnya.
Itu dia! Ada semua yang ada untuk program sederhana ini.

Komentar

Kita telah membahas cukup banyak dalam bab ini. Anda sekarang harus memiliki pemahaman dasar tentang pemrograman C# dan cukup nyaman dengan VSC. Sebelum kita mengakhiri bab ini, ada satu hal lagi yang perlu dipelajari - komentar. Jika Anda merujuk kembali ke contoh "HelloWorld" kita dan melihat baris 10, Anda akan melihat bahwa baris ini dimulai dengan dua garis miring (//).

```
//A Simple Program to display the words Hello World
```

Baris ini sebenarnya bukan bagian dari program. Ini adalah komentar yang kita tulis untuk membuat kode kita lebih mudah dibaca oleh programmer lain. Komentar diabaikan oleh kompiler. Untuk menambahkan komentar ke program kita, kita mengetik dua garis miring (//) di depan setiap baris komentar seperti ini

```
  // This is a comment
  // This is another comment
  // This is yet another comment
```

Atau, kita juga dapat menggunakan /* … */ untuk komentar multiline seperti ini

```
  /* This is a comment This
  is also a comment
  This is yet another comment
*/
```

Komentar juga dapat ditempatkan setelah pernyataan, seperti ini:

```
  Console.Read(); //reads the next character
```

BAB 3
DUNIA VARIABEL DAN OPERATOR

Sekarang Anda sudah familiar dengan VSC dan telah menulis program pertama Anda, mari kita langsung ke hal-hal yang sebenarnya. Dalam bab ini, Anda akan mempelajari semua tentang variabel dan operator. Secara khusus, Anda akan mempelajari apa itu variabel dan bagaimana memberi nama, mendeklarasikan, dan menginisialisasinya. Anda juga akan belajar tentang operasi umum yang dapat kita lakukan pada mereka.

3.1 APA ITU VARIABEL?

Variabel adalah nama yang diberikan pada data yang perlu kita simpan dan manipulasi dalam program kita. Misalnya, program Anda perlu menyimpan usia pengguna. Untuk melakukannya, kita dapat menamai data ini userAge dan mendeklarasikan variabel userAge menggunakan pernyataan berikut:

```
int userAge;
```

Pernyataan deklarasi pertama menyatakan tipe data variabel, diikuti dengan namanya. Tipe data variabel mengacu pada tipe data yang akan disimpan variabel (seperti apakah itu angka atau teks). Dalam contoh kita, tipe datanya adalah int, yang mengacu pada bilangan bulat. Nama variabel kita adalah userAge. Setelah Anda mendeklarasikan variabel userAge, program Anda akan mengalokasikan area tertentu dari ruang penyimpanan komputer Anda untuk menyimpan data ini. Anda kemudian dapat mengakses dan memodifikasi data ini dengan merujuknya dengan namanya, userAge.

3.2 TIPE DATA DALAM C#

Ada beberapa tipe data yang umum digunakan dalam C#.

Int

Int singkatan dari integer (yaitu angka tanpa bagian desimal atau pecahan) dan memegang angka dari -2.147.483.648 hingga 2.147.483.647. Contohnya termasuk 15, 407, -908, 6150 dll.

Byte

Byte juga mengacu pada bilangan integral, tetapi memiliki rentang yang lebih sempit dari 0 hingga 255. Sebagian besar waktu, kita menggunakan int t alih-alih byte untuk bilangan integral. Namun, jika Anda memprogram untuk mesin yang memiliki ruang memori terbatas, Anda harus menggunakan byte jika Anda yakin nilai variabel tidak akan melebihi rentang 0 hingga 255. Misalnya, jika Anda perlu menyimpan usia pengguna, Anda dapat menggunakan tipe data byte karena kecil kemungkinan usia pengguna akan melebihi 255 tahun.

Float

Float mengacu pada angka floating point, yang merupakan angka dengan tempat desimal seperti 12,43, 5,2 dan -9,12. float dapat menyimpan angka dari -3,4 x 1038 hingga +3,4 x 1038. Menggunakan penyimpanan 8 byte dan memiliki presisi sekitar 7 digit. Artinya,

jika Anda menggunakan float untuk menyimpan angka seperti 1.23456789 (10 digit), angka tersebut akan dibulatkan menjadi 1.234568 (7 digit).

Double

Double juga merupakan angka floating point, tetapi dapat menyimpan rentang angka yang jauh lebih luas. Ini dapat menyimpan angka dari (+/-)5,0 x 10-324 hingga (+/-)1,7 x 10308 dan memiliki presisi sekitar 15 hingga 16 digit. double adalah tipe data floating point default di C#. Dengan kata lain, jika Anda menulis angka seperti 2.34, C# memperlakukannya sebagai double secara default.

Decimal

Decimal menyimpan angka desimal tetapi memiliki rentang yang lebih kecil daripada float dan double. Namun, ia memiliki presisi yang jauh lebih besar sekitar 28-29 digit. Jika program Anda memerlukan tingkat presisi yang tinggi saat menyimpan bilangan non integral, Anda harus menggunakan tipe data decimal. Contohnya adalah ketika Anda menulis aplikasi keuangan di mana presisi sangat penting.

Char

Char adalah singkatan dari character dan digunakan untuk menyimpan satu karakter Unicode seperti 'A', '%', '@' dan 'p' dll.

Bool

Bool adalah singkatan dari boolean dan hanya dapat menampung dua nilai: true dan false. Ini biasanya digunakan dalam pernyataan aliran kontrol. Kita akan membahas pernyataan aliran kontrol di Bab 6.

3.3 PENAMAAN VARIABEL

Nama variabel dalam C# hanya boleh berisi huruf, angka, atau garis bawah (_). Namun, karakter pertama tidak boleh berupa angka. Oleh karena itu, Anda dapat memberi nama variabel Anda userName, user_nameatau userName2 tetapi bukan 2userName. Selain itu, ada beberapa kata yang dicadangkan yang tidak dapat Anda gunakan sebagai nama variabel karena sudah memiliki arti yang ditentukan sebelumnya dalam C#. Kata-kata yang dicadangkan ini mencakup kata-kata seperti Console, if, while dll. Kita akan mempelajari masing-masing kata tersebut di bab selanjutnya. Akhirnya, nama variabel peka huruf besar/kecil. username tidak sama dengan userName.

Ada dua konvensi saat memberi nama variabel dalam C#. Kita bisa menggunakan notasi kasus unta atau menggunakan garis bawah. Casing unta adalah praktik menulis kata majemuk dengan casing campuran, menggunakan huruf kapital pada huruf pertama setiap kata kecuali kata pertama (mis. thisIsAVariableName). Ini adalah konvensi yang akan kita gunakan di sisa buku ini. Atau, praktik umum lainnya adalah menggunakan garis bawah (_) untuk memisahkan kata-kata. Jika Anda mau, Anda dapat memberi nama variabel Anda seperti ini:

this_is_a_variable_name.

3.4 MENGINISIALISASI VARIABEL

Setiap kali Anda mendeklarasikan variabel baru, Anda harus memberinya nilai awal. Ini dikenal sebagai inisialisasi variabel. Anda dapat mengubah nilai variabel dalam program Anda nanti. Ada dua cara untuk menginisialisasi variabel. Anda dapat menginisialisasinya pada titik deklarasi atau menginisialisasinya dalam pernyataan terpisah.

Contoh di bawah ini menunjukkan bagaimana Anda dapat menginisialisasi variabel pada titik deklarasi:

Contoh 1

Contoh-contoh ini menunjukkan bagaimana Anda dapat menginisialisasi variabel byte dan int.

```
byte userAge = 20;
int numberOfEmployees = 510;
```

Karena byte dan int adalah untuk data tanpa tempat desimal, Anda akan mendapatkan kesalahan jika Anda menulis sesuatu seperti:

```
byte userAge2 = 20.0;
```

20.0 tidak sama dengan 20 di C#.

Contoh 2

Contoh berikut menunjukkan bagaimana Anda dapat menginisialisasi variabel double, float dan decimal dengan nilai integral. Meskipun tipe data ini untuk bilangan dengan bagian desimal, kita juga dapat menggunakannya untuk menyimpan nilai integral seperti yang ditunjukkan di bawah ini:

```
double intNumberOfHours = 5120;
float intHourlyRate = 60;
decimal intIncome = 25399;
```

Contoh 3

Contoh di bawah ini menunjukkan bagaimana Anda dapat menginisialisasi variabel double, float dan desimal dengan non integer.

```
double numberOfHours = 5120.5;
float hourlyRate = 60.5f;
decimal income = 25399.65m;
```

Seperti disebutkan sebelumnya, tipe data default untuk angka dengan tempat desimal adalah double. Oleh karena itu, dalam contoh di atas, ketika Anda menginisialisasi hourlyRate, Anda perlu menambahkan 'f' sebagai akhiran setelah 60.5 untuk secara eksplisit memberi tahu kompiler untuk mengubah 60.5 menjadi float. Demikian pula, ketika Anda menginisialisasi income, Anda perlu menambahkan 'm' sebagai akhiran mengubah 25399.65 menjadi tipe data decimal.

Contoh 4

Tipe data char hanya dapat berisi satu karakter. Saat kita menginisialisasi variabel char, kita perlu menyertakan karakter itu dalam tanda kutip tunggal. Contohnya adalah: char grade = ‘A’;

Contoh 5

Variabel bool hanya bisa true atau false. Contoh di bawah ini menunjukkan bagaimana Anda dapat menginisialisasi variabel bool.

```
bool promote = true;
```

Contoh 6

Selain menginisialisasi variabel satu per satu, Anda juga dapat menginisialisasi beberapa variabel dalam pernyataan yang sama asalkan memiliki tipe data yang sama. Contoh berikut menunjukkan bagaimana hal ini dapat dilakukan. Perhatikan bahwa kedua variabel dipisahkan dengan koma dan pernyataan diakhiri dengan titik koma.

```
byte level = 2, userExperience = 5;
```

Keenam contoh di atas menunjukkan bagaimana Anda dapat menginisialisasi variabel pada titik deklarasi. Atau, Anda dapat memilih untuk menginisialisasi variabel dalam pernyataan terpisah. Contoh ditunjukkan di bawah ini:

```
byte year; //declare the variable
year = 20; //initialize it
```

3.5 TANDA TUGAS

Tanda = dalam pemrograman memiliki arti yang berbeda dari tanda = yang kita pelajari di Matematika. Dalam pemrograman, tanda = dikenal sebagai tanda penugasan. Ini berarti kita menetapkan nilai di sisi kanan tanda = ke variabel di sebelah kiri. Cara yang baik untuk memahami pernyataan seperti year = 20adalah dengan menganggapnya sebagai year <- 20. Dalam pemrograman, pernyataan x = ydan y = xmemiliki arti yang sangat berbeda. Bingung? Sebuah contoh kemungkinan akan menjernihkan hal ini. Misalkan kita memiliki dua variabel x dan y dan x = 5; y = 10;

Jika Anda menulis:

```
x = y;
```

Guru Matematika Anda mungkin akan marah pada Anda karena x tidak sama dengan y.

Namun, dalam pemrograman, ini baik-baik saja. Pernyataan ini berarti kita menetapkan nilai y ke x (anggap saja sebagai x <- y). Tidak apa-apa untuk menetapkan nilai variabel ke variabel lain. Dalam contoh kita, nilai x sekarang diubah menjadi 10 sedangkan nilai y tetap tidak berubah. Dengan kata lain, x = 10 dan y = 10 sekarang.

Sekarang misalkan kita mengubah nilai x dan y kembali ke:

```
x = 5; y = 10;
```

Jika Anda menulis:

```
y = x;
```

itu berarti Anda menetapkan nilai x ke y (anggap saja sebagai y <- x). Secara matematis, x = y dan y = x memiliki arti yang sama. Namun, tidak demikian dalam pemrograman. Di sini, nilai y

diubah menjadi 5 sedangkan nilai x tetap tidak berubah. Dengan kata lain, x = 5 dan y = 5 sekarang.

3.6 OPERATOR DASAR

Selain menetapkan nilai awal ke variabel atau menetapkan variabel lain untuk itu, kita juga dapat melakukan operasi matematika biasa pada variabel. Operator dasar dalam C# termasuk +, -, *, / dan % yang masing-masing mewakili penambahan, pengurangan, perkalian, pembagian dan modulus.

Contoh

Misalkan x = 7, y = 2

Penjumlahan: x + y = 9
Pengurangan: x - y = 5
Perkalian: x*y = 14
Pembagian: x/y = 3 (membulatkan jawaban ke bilangan bulat terdekat)
Modulus: x%y = 1 (memberikan sisa jika 7 dibagi 2)

Dalam C#, pembagian memberikan jawaban bilangan bulat jika x dan y keduanya bilangan bulat. Namun, jika x atau y adalah non integer, kita akan mendapatkan jawaban non integer. Misalnya,

7/2 = 3
7,0 / 2 = 3,5
7 / 2.0 = 3.5
7.0 / 2.0 = 3.5

Dalam kasus pertama, ketika bilangan bulat dibagi dengan bilangan bulat lain, Anda mendapatkan bilangan bulat sebagai jawabannya. Bagian desimal dari jawaban, jika ada, dipotong. Oleh karena itu, kita mendapatkan 3 bukannya 3,5. Dalam semua kasus lain, hasilnya adalah non integer karena setidaknya salah satu operan adalah non integer.

3.7 LEBIH BANYAK OPERATOR PENUGASAN

Selain tanda =, ada beberapa operator penugasan lagi di C# (dan sebagian besar bahasa pemrograman). Ini termasuk operator seperti +=, -= dan *=. Misalkan kita memiliki variabel x, dengan nilai awal 10. Jika kita ingin menambah x dengan 2, kita dapat menulis x = x + 2;

Program pertama-tama akan mengevaluasi ekspresi di sebelah kanan (x + 2) dan menetapkan jawabannya di sebelah kiri. Sehingga akhirnya pernyataan di atas menjadi x <- 12. Alih-alih menulis x = x + 2, kita juga dapat menulis x += 2 untuk menyatakan arti yang sama. Tanda += sebenarnya adalah singkatan yang menggabungkan tanda penugasan dengan operator penjumlahan. Jadi, x += 2 berarti x = x + 2.

Demikian pula, jika kita ingin melakukan pengurangan, kita dapat menulis x = x - 2 atau x -= 2. Cara yang sama berlaku untuk 5 operator yang disebutkan pada bagian di atas. Sebagian

besar bahasa pemrograman juga memiliki operator ++ dan —. Operator ++ digunakan ketika Anda ingin meningkatkan nilai variabel sebesar 1. Misalnya, misalkan int x = 2;
Jika Anda menulis x++;
nilai x menjadi 3.

Tidak perlu menggunakan tanda = saat Anda menggunakan operator ++. Pernyataan x++; setara dengan x = x + 1;

Operator ++ dapat ditempatkan di depan atau di belakang nama variabel. Ini mempengaruhi urutan tugas yang dilakukan.
Misalkan kita memiliki bilangan bulat bernama counter r. Jika kita menulis:

```
Console.WriteLine(counter++);
```

Program pertama-tama mencetak nilai asli counter sebelum menambah counter sebesar 1. Dengan kata lain, program menjalankan tugas dalam urutan ini

```
Console.WriteLine(counter);
counter = counter + 1;
```

Sebaliknya, jika kita menulis

```
Console.WriteLine(++counter);
```

program pertama-tama menambah penghitung sebanyak 1 sebelum mencetak nilai baru dari counter. Dengan kata lain, ia menjalankan tugas dalam urutan ini:

```
counter = counter + 1; Console.WriteLine(counter);
```

Selain operator ++, kita juga memiliki operator — (dua tanda minus). Operator ini menurunkan nilai variabel sebesar 1.

3.8 KETIK PENGECORAN DALAM C#

Terkadang dalam program kita, kita perlu mengonversi dari satu tipe data ke tipe data lainnya, seperti dari double ke int. Ini dikenal sebagai tipe casting. Untuk mengkonversi satu tipe data numerik ke yang lain, kita hanya perlu menambahkan (new data type) di depan data yang ingin kita konversi. Misalnya, kita dapat memasukkan non integer menjadi integer seperti ini:

```
int x = (int) 20.9;
```

Saat kita memasukkan 20,9 ke dalam bilangan bulat, nilai yang dihasilkan adalah 20, bukan 21. Bagian desimal terpotong setelah konversi. Kita juga dapat melemparkan double ke dalam float atau decimal. Ingat bahwa kita sebutkan sebelumnya bahwa semua non integer diperlakukan sebagai double secara default di C#? Jika kita ingin menetapkan angka seperti 20,9 ke float atau decimal, kita perlu menambahkan sufiks 'f' dan 'm' masing-masing. Cara lain untuk melakukannya adalah dengan menggunakan gips, seperti ini:

```
float num1 = (float) 20.9; decimal
num2 = (decimal)20.9;
```

Nilai num1 dan num2 keduanya akan menjadi 20,9. Selain casting antar tipe numerik, kita juga dapat melakukan casting jenis lain. Kita akan menjelajahi beberapa dari konversi ini di bab-bab berikutnya.

BAB 4
ARRAY, STRING, DAN DAFTAR

Pada bab sebelumnya, kita telah membahas beberapa tipe data dasar yang umum digunakan dalam C#. Selain tipe data dasar ini, C# juga dilengkapi dengan beberapa tipe data lanjutan. Dalam bab ini, kita akan membahas tiga tipe data lanjutan: array, string, dan daftar. Selain itu, kita akan membahas perbedaan antara tipe data nilai dan tipe data referensi.

4.1 ARRAY

Array hanyalah kumpulan data yang biasanya terkait satu sama lain. Misalkan kita ingin menyimpan usia 5 pengguna. Alih-alih menyimpannya sebagai user1Age, user2Age, user3Age, user4Age dan user5Age, kita dapat menyimpannya sebagai array.
Sebuah array dapat dideklarasikan dan diinisialisasi sebagai berikut:

```
int[] userAge = {21, 22, 23, 24, 25};
```

int menunjukkan bahwa variabel ini menyimpan nilai int.
[] menunjukkan bahwa variabel adalah array, bukan variabel normal.
userAge adalah nama array.
{21, 22, 23, 24, 25}adalah lima bilangan bulat yang disimpan array.

Selain mendeklarasikan dan menginisialisasi array pada titik deklarasi, kita dapat mendeklarasikan array terlebih dahulu dan menginisialisasinya nanti. Untuk melakukan itu, kita perlu menggunakan operator baru:

```
int[] userAge = new int[5];
userAge = new [] {21, 22, 23, 24, 25};
```

Pernyataan pertama mendeklarasikan dan membuat larik untuk menyimpan 5 bilangan bulat. Pernyataan kedua menginisialisasi array. Nilai individu dalam array dapat diakses oleh indeksnya, dan indeks selalu dimulai dengan nilai NOL, bukan 1. Ini adalah praktik umum di hampir semua bahasa pemrograman, seperti Python dan Java. Nilai pertama dari array memiliki indeks 0, berikutnya memiliki indeks 1 dan seterusnya.
Jika kita mengetik:

```
Console.WriteLine(userAge[0]);
```

nilai '21' akan ditampilkan di layar.

Jika kita mengetik:

```
userAge[2] = userAge[2] + 20;
```

array menjadi {21, 22, 43, 24, 25}. Artinya, 20 ditambahkan ke elemen ketiga.

4.2 PROPERTI DAN METODE ARRAY

C# hadir dengan sejumlah properti dan metode berguna yang dapat kita gunakan dengan array.

Kita akan belajar lebih banyak tentang properti dan metode di Bab 7 saat kita membahas kelas. Untuk saat ini, yang perlu kita ketahui adalah bahwa untuk menggunakan properti atau metode, kita perlu menggunakan operator titik (.). Untuk menggunakan properti, kita ketik nama properti setelah titik. Untuk menggunakan metode, kita ketik nama metode setelah operator titik, diikuti dengan sepasang tanda kurung ().

Length

Properti Length dari sebuah array memberi tahu kita jumlah item yang dimiliki array.
Misalnya, jika kita memiliki

```
int [] userAge = {21, 22, 26, 32, 40};
```

userAge.Length. Length sama dengan 5 karena ada 5 angka dalam array.

Copy()

Metode Copy()memungkinkan Anda untuk menyalin konten dari satu larik ke larik lain, mulai dari elemen pertama. Dalam C#, sebuah metode mungkin memiliki banyak variasi yang berbeda. Misalnya, metode Copy()hadir dalam empat variasi berbeda. Contoh di bawah ini membahas salah satu dari empat variasi. Jika Anda mempelajari cara menggunakan satu variasi, Anda dapat mengetahui cara menggunakan metode Copy()lainnya dengan relatif mudah. Setiap kali kita menggunakan suatu metode, kita perlu meletakkan sepasang tanda kurung () setelah nama metode. Beberapa metode memerlukan data tertentu agar dapat berfungsi. Data ini dikenal sebagai argumen. Kita menyertakan argumen ini dalam pasangan kurung. Metode Copy()membutuhkan tiga argumen.
Misalkan Anda memiliki:

```
int [] source = {12, 1, 5, -2, 16, 14};
```

dan

```
int [] dest = {1, 2, 3, 4};
```

Anda dapat menyalin tiga elemen pertama dari source ke dest dengan menggunakan pernyataan di bawah ini:

```
Array.Copy(source, dest, 3);
```

Argumen pertama adalah larik yang menyediakan nilai yang akan disalin. Yang kedua adalah larik tempat nilai akan disalin. Argumen terakhir menentukan jumlah item yang akan disalin. Dalam contoh kita, dest tujuan kita menjadi {12, 1, 5, 4} sedangkan larik sumber tetap tidak berubah.

Sort()

Metode Sort()memungkinkan kita untuk mengurutkan array kita. Dibutuhkan dalam array sebagai argumen.
Misalkan Anda memiliki

```
int [] numbers = {12, 1, 5, -2, 16, 14};
```

Anda dapat mengurutkan array ini dengan menulis
Array.Sort(numbers);

Array akan diurutkan dalam urutan menaik. Jadi, numbers menjadi {-2, 1, 5,12, 14, 16}.

IndexOf()

Kita menggunakan metode IndexOf()untuk menentukan apakah ada nilai tertentu dalam array. Jika ada, metode mengembalikan indeks kemunculan pertama dari nilai tersebut. Jika tidak ada, metode mengembalikan -1.
Misalnya, jika Anda memiliki
int [] numbers = {10, 30, 44, 21, 51, 21, 61, 24, 14};

Anda dapat menemukan apakah nilai 21 ada dalam array dengan menulis
Array.IndexOf(numbers, 21);

Metode mengembalikan indeks dari nilai pertama yang ditemukan, yaitu 3 dalam hal ini karena 21 adalah elemen keempat dalam larik. Anda kemudian dapat menetapkan jawaban ke variabel seperti ini:
int ans = Array.IndexOf(numbers, 21);

Jadi, nilai ans adalah 3. Jika ditulis
ans = Array.IndexOf(numbers, 100);

nilai ans adalah -1 karena 100 tidak ada dalam array numbers.

Kita telah membahas beberapa metode array yang lebih umum digunakan di bagian ini. Untuk daftar lengkap semua metode array yang tersedia di C#, lihat halaman ini
https://msdn.microsoft.com/en-us/library/system.array_methods(v=vs.110).aspx

String

Selanjutnya, mari kita lihat tipe data string. String adalah sepotong teks. Contoh string adalah teks "Hello World". Untuk mendeklarasikan dan menginisialisasi variabel string, Anda menulis:
string pesan = “Halo Dunia”;

Di mana message adalah nama variabel string dan "Hello World" adalah string yang ditugaskan padanya. Perhatikan bahwa Anda perlu menyertakan string dalam tanda kutip ganda (“). Anda juga dapat menetapkan string kosong ke variabel, seperti ini:
string message = “Hello World”;
Terakhir, kita dapat menggabungkan dua atau lebih string menggunakan tanda gabungan (+) dan menetapkannya ke variabel. Misalnya, kita dapat menulis
string myName = “Hello World, ” + “my name is Jamie”;

Ini sama dengan
string myName = “Hello World, my name is Jamie”;

4.3 PROPERTI DAN METODE STRING

Seperti array, string datang dengan sejumlah properti dan metode.

Length

Properti Length dari sebuah string memberi tahu kita jumlah total karakter yang dikandung string. Untuk menemukan panjang string "Hello World", kita menulis:
“Hello World”.Length

Kita akan mendapatkan nilai 11 sebagai "Halo" dan "Dunia" keduanya memiliki masing-masing 5 karakter. Saat Anda menambahkan spasi di antara dua kata, Anda mendapatkan total panjang 11.

Substring()

Metode Substring() digunakan untuk mengekstrak substring dari string yang lebih panjang. Ini membutuhkan dua argumen. Yang pertama memberi tahu kompiler indeks posisi awal untuk mengekstrak dan yang kedua memberi tahu kompiler panjangnya.

Misalkan kita mendeklarasikan message variabel string dan menetapkan string "Hello World" padanya.
string message = “Hello World”;

Kita kemudian dapat menggunakan message untuk memanggil metode Substring() seperti yang ditunjukkan di bawah ini.
string newMessage = message.Substring(2, 5);

Substring(2, 5)mengekstrak substring dari 5 karakter dari pesan, mulai dari indeks 2 (yang merupakan huruf ketiga karena indeks selalu dimulai dari 0). Substring yang dihasilkan kemudian ditugaskan ke newMessage. newMessage dengan demikian sama dengan "llo W". message, di sisi lain, tidak berubah. Itu tetap sebagai "Halo Dunia".

Equals()

Kita dapat menggunakan metode Equals() untuk membandingkan jika dua string identik. Jika kita memiliki dua string seperti yang ditunjukkan di bawah ini:
string firstString = “This is Jamie”;
string secondString = “Hello”;

firstString.Equals(“This is Jamie”);
mengembalikan true while
firstString.Equals(secondString);
mengembalikan false karena dua string (firstStringdan secondString) tidak sama.

Split()

Metode Split() membagi string menjadi substring berdasarkan larik pemisah yang ditentukan pengguna. Setelah memisahkan string, metode Split()mengembalikan array yang berisi substring yang dihasilkan. Metode Split()memerlukan dua argumen - larik string yang bertindak sebagai pemisah dan argumen kedua untuk menentukan apakah Anda ingin menghapus string kosong dari hasilnya.

Misalkan Anda ingin membagi string "Peter, John; Andy, ,David" menjadi substring, Anda dapat melakukannya sebagai berikut (nomor baris ditambahkan untuk referensi):

```
1 string [] separator = {", ", "; "};
2 string names = "Peter, John; Andy, , David";
3 string [] substrings = names.Split(separator,
StringSplitOptions.None);
```

Pada Baris 1, pertama-tama kita mendeklarasikan larik dua string untuk bertindak sebagai pemisah. String pertama adalah koma diikuti dengan spasi dan string kedua adalah titik koma diikuti spasi. Pada Baris 2, kita menetapkan string yang ingin kita pisahkan ke variabel names. Pada Baris 3, kita menggunakan variabel names untuk memanggil metode Split dan menetapkan hasilnya ke larik substrings.
Hasil dari kode di atas adalah array berikut:
{"Peter", "John", "Andy", "" , "David"}

Array ini berisi string kosong karena ada spasi antara koma setelah "Andy" dan koma sebelum "David" di string asli. Jika Anda ingin menghapus string kosong dari hasil, Anda harus mengubah Baris 3 menjadi:

```
string [] substrings = names.Split(separator,
StringSplitOptions.RemoveEmptyEntries);
```

Array substrings dengan demikian menjadi
{"Peter", "John", "Andy", "David"}

Seperti biasa, kita hanya membahas sejumlah metode string yang lebih umum digunakan. Untuk daftar lengkap semua metode string yang tersedia di C#, lihat halaman ini https://msdn.microsoft.com/en-us/library/system.string_methods(v=vs.110).aspx

4.4 DAFTAR

Sekarang, mari kita lihat tipe data terakhir dalam bab ini – daftar. Daftar menyimpan nilai seperti array, tetapi elemen dapat ditambahkan atau dihapus sesuka hati. Array hanya dapat menampung sejumlah nilai tetap. Jika Anda menyatakan:

```
int [] myArray = new int[10];
```

myArray hanya dapat menampung 10 nilai. Jika Anda menulis myArray[10] (yang mengacu pada nilai ke-11 sejak indeks array dimulai dari nol), Anda akan mendapatkan kesalahan. Jika Anda membutuhkan fleksibilitas yang lebih besar dalam program Anda, Anda dapat menggunakan daftar. Untuk mendeklarasikan daftar bilangan bulat, kita menulis:

```
List<int> userAgeList = new List<int>();
```

userAgeList adalah nama daftar.

List adalah kata kunci untuk menunjukkan bahwa Anda mendeklarasikan daftar. Tipe data diapit oleh kurung sudut < >.

Anda dapat memilih untuk menginisialisasi daftar pada titik deklarasi seperti ini:

```
List<int> userAgeList = new List<int> {11, 21, 31, 41};
```

Untuk mengakses elemen individual dalam daftar, kita menggunakan notasi yang sama seperti ketika kita mengakses elemen dalam array. Misalnya, untuk mengakses elemen pertama, Anda menulis userAgeList[0]. Untuk mengakses elemen ketiga, Anda menulis userAgeList[2]..

4.5 DAFTAR PROPERTI DAN METODE

Tipe data daftar juga dilengkapi dengan sejumlah besar properti dan metode.

Add()

Anda dapat menambahkan anggota ke daftar menggunakan metode Add().

```
userAgeList.Add(51);
userAgeList.Add(61);
```

userAgeList sekarang memiliki 6 anggota: {11, 21, 31, 41, 51, 61}.

Count

Untuk mengetahui jumlah elemen dalam daftar, gunakan properti Count. userAgeList.Count memberi kita 6 karena ada 6 elemen dalam daftar saat ini.

Insert()

Untuk menambahkan anggota pada posisi tertentu, gunakan metode Insert(). Untuk menyisipkan anggota di posisi ke-3, Anda menulis userAgeList.Insert(2, 51); di mana 2 adalah indeks dan 51 adalah nilai yang ingin Anda masukkan. userAgeList sekarang menjadi {11, 21, 51, 31, 41, 51, 61}.

Remove()

Untuk menghapus anggota dari daftar, gunakan metode Remove(). Metode Remove() mengambil satu argumen dan menghapus kemunculan pertama dari argumen itu.

Misalnya, jika kita menulis
userAgeList.Remove(51);
userAgeLis menjadi {11, 21, 31, 41, 51, 61}. Hanya '51' pertama yang dihapus.

RemoveAt()

Untuk menghapus anggota di lokasi tertentu, gunakan metode RemoveAt(). Misalnya, untuk menghapus item ke-4 (indeks 3), Anda menulis userAgeList.RemoveAt(3);
Dimana 3 adalah indeks item yang akan dihapus. userAgeList sekarang menjadi { {11, 21, 31, 51, 61}.

Contains()

Untuk memeriksa apakah daftar berisi anggota tertentu, gunakan metode Contains(). Untuk memeriksa apakah userAgeList berisi '51', kita menulis userAgeList.Contains(51);
Kita akan mendapatkan true sebagai hasilnya.

Clear()

Untuk menghapus semua item dalam daftar, gunakan metode Clear(). Jika kita menulis userAgeList.Clear();
kita tidak akan memiliki elemen yang tersisa dalam daftar.

Untuk daftar lengkap semua metode daftar yang tersedia di C#, lihat halaman ini
https://msdn.microsoft.com/en-us/library/s6hkc2c4(v=vs.110).aspx

4.6 JENIS NILAI VS JENIS REFERENSI

Sekarang kita sudah familiar dengan string, array dan daftar, mari kita bahas konsep penting mengenai tipe data dalam C#. Semua tipe data dalam C# dapat diklasifikasikan sebagai tipe nilai atau tipe referensi. Tipe data yang dibahas dalam Bab 3 adalah tipe nilai. Yang dibahas dalam bab ini adalah jenis referensi. Tipe data nilai adalah variabel yang menyimpan datanya sendiri.
Saat kita menulis:
int myNumber = 5;
variabel myNumber menyimpan nilai aktual 5.

Jenis referensi, di sisi lain, tidak menyimpan data aktual. Sebaliknya, itu menyimpan referensi ke data. Itu tidak memberi tahu kompiler berapa nilai datanya; itu memberitahu kompiler di mana menemukan data aktual. Contoh dari tipe referensi adalah string. Saat Anda menulis pernyataan seperti string message = “Hello”;
message variabel sebenarnya tidak menyimpan string "Halo".

Sebaliknya, string "Halo" dibuat dan disimpan di tempat lain di memori komputer. message variabel menyimpan alamat lokasi memori tersebut. Itu saja yang perlu kita ketahui tentang tipe referensi saat ini. Karena ini adalah buku untuk pemula, kita tidak akan membahas secara rinci mengapa jenis referensi diperlukan. Perlu diketahui bahwa ada perbedaan antara tipe nilai dan tipe referensi; yang pertama menyimpan nilai sementara yang terakhir menyimpan alamat.

BAB 5
MENJADIKAN PROGRAM KITA INTERAKTIF

Sekarang kita telah membahas dasar-dasar variabel dan tipe data, mari kita menulis program yang memanfaatkannya. Dalam bab ini, kita akan belajar bagaimana menerima input dari pengguna, menyimpan data dalam variabel dan menampilkan pesan kepada pengguna kita. Siap?

5.1 MENAMPILKAN PESAN KEPADA PENGGUNA

Untuk menampilkan pesan kepada pengguna kita, kita menggunakan metode Write()atau WriteLine()yang disediakan oleh C#, tersedia di ruang nama System.

Perbedaan antara WriteLine()dan Write()adalah bahwa Writeline()memindahkan kursor ke baris berikutnya setelah menampilkan pesan sedangkan Write() tidak.

Jika kita menulis:

```
Console.WriteLine("Hello ");
Console.WriteLine("How are you?");
```

kita akan mendapatkan:

```
Hello
How are you?
```

Jika kita menulis:

```
Console.Write("Hello ");
Console.Write("How are you?");
```

kita akan mendapatkan:

```
Hello How are you?
```

Perhatikan bahwa dalam contoh di atas, kita menambahkan kata Console di depan nama metode setiap kali kita memanggil metode WriteLine() atau Write(). Ini karena kedua metode tersebut adalah metode statis dari kelas Console. Kita akan berbicara lebih banyak tentang metode statis di Bab 7.

Jika Anda merasa kesulitan untuk menambahkan kata Console setiap kali Anda menggunakan dua metode ini, Anda dapat menambahkan arahan:

```
using static System.Console;
```

ke awal program Anda. Jika Anda melakukannya, Anda cukup menulis:

```
WriteLine("Hello World");
```

alih-alih

```
Console.WriteLine("Hellow World");
```

Setiap kali Anda menggunakan salah satu metode statis di kelas Console. Ini adalah fitur baru di C# 6 (versi terbaru C#) dan hanya tersedia jika Anda menggunakan IDE terbaru (yaitu Visual Studio 2015). Untuk sisa contoh kita, kita akan tetap menggunakan metode pertama untuk kompatibilitas mundur.

Kita telah melihat contoh bagaimana kita dapat menggunakan metode WriteLine(ketika kita menulis program "Hello World" di Bab 2. Sekarang mari kita lihat contoh lainnya. Dalam contoh di bawah ini, kita akan fokus pada metode WriteLine(. Metode Write() bekerja dengan cara yang persis sama.

Contoh 1

Untuk menampilkan string sederhana, kita menulis
Console.WriteLine("Hello, how are you?");

Output
Hello, how are you?

Contoh 2

Untuk menampilkan nilai suatu variabel, kita memasukkan nama variabel sebagai argumen. Misalnya, misalkan kita memiliki:
int userAge = 30;

kita menampilkan nilai userAge dengan menulis:
Console.WriteLine(userAge);

Output:
30

Perhatikan bahwa kita tidak menyertakan nama variabel (userAge) dalam tanda kutip ganda. Jika kita menulis:
Console.WriteLine("userAge");

kita akan mendapatkan:
userAge
sebagai output sebagai gantinya.

Contoh 3

Untuk menggabungkan dua atau lebih string dan menampilkannya, kita menggunakan tanda penggabungan (+) yang disebutkan dalam bab sebelumnya. Misalnya, jika kita menulis:
Console.WriteLine("Hello, " + "how are you?" + " I love C#.");

kita akan mendapatkan:
Hello, how are you? I love C#.

Contoh 4

Kita juga dapat menggunakan tanda penggabungan untuk menggabungkan string dan variabel.
Misalkan kita memiliki:
int results = 79;

Pernyataan:
Console.WriteLine("You scored " + results + " marks for your test.");

memberi kita:
You scored 79 marks for your test.

Sekali lagi, kita tidak menyertakan nama variabel dalam tanda kutip ganda. Jika tidak, kita akan mendapatkan:
You scored results marks for your test.

Contoh 5

Selain menggunakan tanda penggabungan untuk menggabungkan string dan variabel, kita dapat menggunakan placeholder. Misalkan kita memiliki:
int results = 79;

Jika kita menulis:
Console.WriteLine("{0}! You scored {1} marks for your test.", "Good morning", results);

kita akan mendapatkan:
Good morning! You scored 79 marks for your test.

Dalam contoh ini, kita meneruskan tiga argumen ke metode WriteLine(), dipisahkan dengan koma. Ketiga argumen tersebut adalah:

1) "{0}! You scored {1} marks for your test."
2) "Good morning"
3) results

Yang pertama adalah string yang akan ditampilkan. Di dalam string, kurung kurawal bertindak sebagai pengganti dan akan digantikan oleh argumen yang mengikutinya.
{0} adalah pengganti untuk argumen berikutnya, yang merupakan string "Selamat pagi" dalam kasus ini.
{1} adalah pengganti untuk results variabel.

Oleh karena itu output adalah:
Good morning! You scored 79 marks for your test.

Jika Anda menulis:
Console.WriteLine("{1}! You scored {0} marks for your test.", "Good morning", results);

Anda akan mendapatkan:
79! You scored Good morning marks for your test.

Tentu saja pernyataan seperti itu tidak masuk akal. Namun, ini menunjukkan bagaimana placeholder digantikan oleh argumen yang sesuai. Kita dapat menentukan bagaimana kita ingin nilai numerik ditampilkan saat menggunakan placeholder. Ini dilakukan dengan menggunakan penentu format, seperti penentu C dan F. Penentu F menentukan jumlah tempat desimal yang harus ditampilkan dengan angka.

Jika kita menulis:
Console.WriteLine("The number is {0:F3}.", 123.45678);

kita akan mendapatkan:
The number is 123.457.

Penentu F3 membulatkan angka 123.45678 menjadi 123.457. Perhatikan bahwa tidak boleh ada spasi sebelum specifier. Dengan kata lain, itu harus {0: F3} dan bukan {0: F3}. Penentu C adalah untuk memformat mata uang; itu menambahkan simbol "$" di depan nomor dan menampilkan nomor dengan 2 tempat desimal. Selain itu, ia memisahkan setiap seribu dengan koma.

Jika Anda menulis:
Console.WriteLine("Deposit = {0:C}. Account balance = {1:C}.",2125, 12345.678);

Anda akan mendapatkan:
Deposit = $2,125.00. Account balance = $12,345.68

Contoh 6
Kita juga dapat menggunakan Console.WriteLine() untuk mencetak hasil dari suatu metode.

Di Bab 4, kita mempelajari cara menggunakan metode Substring()untuk mengekstrak substring dari string yang lebih panjang. Dalam contoh itu, kita menetapkan hasilnya ke string lain. Atau, kita dapat menggunakan Console.WriteLine() untuk menampilkan hasilnya tanpa menetapkannya sebagai variabel.
Misalnya, jika Anda menulis:
Console.WriteLine("Microsoft".Substring(1, 3));

Hasil:
icr
akan ditampilkan di layar.

Selain menampilkan hasil suatu metode, Console.WriteLine() juga dapat digunakan untuk menampilkan nilai suatu properti. Jika kita menulis Console.WriteLine(“Hello World”.Length);

Nilai:
11
akan ditampilkan di layar.

5.2 URUTAN MELARIKAN DIRI

Terkadang dalam program kita, kita mungkin perlu mencetak beberapa karakter khusus yang "tidak dapat dicetak" seperti tab atau baris baru. Dalam hal ini, Anda perlu menggunakan karakter \ (garis miring terbalik) untuk menghindari karakter yang memiliki arti berbeda. Misalnya untuk mencetak tab, kita ketik karakter backslash sebelum huruf t, seperti ini: \t. Tanpa karakter \, huruf “t” akan tercetak. Dengan itu, sebuah tab dicetak. Oleh karena itu, jika Anda mengetik
Console.WriteLine(“Hello\tWorld”);

Anda akan mendapatkan:
Hello World

Penggunaan umum lainnya dari karakter garis miring terbalik meliputi:

Untuk mencetak baris baru (\n)

Contoh:
Console.WriteLine(“Hello\nWorld”);

Output:
Hello
World

Untuk mencetak karakter garis miring terbalik itu sendiri (\\)

Contoh:
Console.WriteLine(“\\”);

Output:
\

Untuk mencetak tanda kutip ganda (\”) agar tanda kutip ganda tidak mengakhiri string

Contoh:

```
Console.WriteLine("I am 5'9\" tall");
```

Output:
I am 5'9" tall

5.3 MENERIMA MASUKAN PENGGUNA

Sekarang setelah kita mengetahui cara menampilkan pesan kepada pengguna, mari kita lihat bagaimana kita dapat menerima masukan dari mereka. Untuk menerima input pengguna, kita dapat menggunakan metode Read() atau ReadLine().

Read() membaca karakter berikutnya dari input standar sementara ReadLine() membaca baris karakter. Input standar mengacu pada perangkat standar yang digunakan pengguna untuk memasukkan data, yang biasanya berupa keyboard.

Contoh di bawah ini menunjukkan bagaimana kita dapat menggunakan metode ReadLine() untuk membaca input dari pengguna. Metode Read() bekerja dengan cara yang sama.

```
string userInput = Console.ReadLine();
```

Baik metode Read() dan ReadLine() membaca input pengguna sebagai string. Oleh karena itu, dalam contoh di atas, kita menetapkan hasil Console.ReadLine() ke variabel string yang disebut userInput.

Kita kemudian dapat menggunakan:

```
Console.WriteLine(userInput);
```

untuk mencetak input yang dimasukkan pengguna.

5.4 MENGUBAH STRING MENJADI ANGKA

Terkadang, input yang dimasukkan pengguna perlu diubah menjadi tipe data numerik sehingga Anda dapat melakukan perhitungan di dalamnya. C# memberi kita sejumlah metode untuk melakukan konversi. Metode yang kita gunakan ditemukan di kelas Convert, yang juga dikelompokkan di bawah ruang nama System. Untuk mengonversi string menjadi integer, kita menggunakan metode ToInt32(). Misalnya, jika kita memiliki:

```
string userInput = Console.ReadLine();
```

dan kunci pengguna di 20, userInput akan sama dengan "20" (yang merupakan string dan bukan bilangan bulat karena tanda kutip ganda).

Kita kemudian dapat menggunakan:

```
int newUserInput = Convert.ToInt32(userInput);
```

untuk mengonversi string menjadi bilangan bulat 20 dan menetapkannya ke variabel int. Kita sekarang dapat melakukan operasi matematika biasa pada variabel int baru ini.

Selain mengubah string menjadi integer, kita juga dapat mengonversi string menjadi decimal, float atau double menggunakan ToDecimal(), ToSingle() dan ToDouble() metode masing-masing.

5.5 MENYATUKAN SEMUANYA

Sekarang mari kita gabungkan semua yang telah kita pelajari untuk menulis program yang lengkap. Kita akan memodifikasi program "Hello World" yang kita tulis di Bab 2. Daripada hanya menyapa dunia, kita ingin dunia tahu nama dan usia kita juga. Pertama, jalankan Komunitas Visual Studio dan buat proyek Aplikasi Konsol Visual C# baru. Beri nama proyek "HelloWorldAgain". Ketik segmen kode berikut ke dalam metode Main() (nomor baris ditambahkan untuk referensi).

```
1  string userName = "";
2  int userAge = 0;
3  int currentYear = 0;4
5  Console.Write("Please enter your name: ");
6  userName = Console.ReadLine();
7  Console.Write("Please enter your age: ");
8  userAge = Convert.ToInt32(Console.ReadLine());
9  Console.Write("Please enter the current year: ");
10  currentYear = Convert.ToInt32(Console.ReadLine());
11
12 Console.WriteLine("Hello World! My name is {0} and I am {1} years old. I was born
in {2}.", userName, userAge, currentYear -userAge);
```

Jalankan program dan masukkan informasi berikut:
Please enter your name: JamiePlease
enter your age: 39
Please enter the current year: 2015

Program akan memberi Anda output berikut:
Hello World! My name is Jamie and I am 39 years old. I was born in1976.

Program ini seharusnya cukup mudah dipahami. Namun, ada dua hal yang perlu disebutkan tentang program ini.

Pertama, Baris 10 menunjukkan contoh bagaimana kita dapat menggunakan dua metode dalam pernyataan yang sama. Saat kita menulis:
userAge = Convert.ToInt32(Console.ReadLine());

metode Console.ReadLine() dijalankan terlebih dahulu karena berada dalam sepasang tanda kurung (). Ini mirip dengan bagaimana operasi dalam tanda kurung memiliki urutan prioritas yang lebih tinggi ketika kita mengevaluasi ekspresi matematika. Misalnya, ketika kita mengevaluasi 3 (5 + 9), kita harus menjumlahkan 5 dengan 9 terlebih dahulu sebelum mengalikan jawabannya dengan 3 (yaitu 314).

Setelah Console.ReadLine(dijalankan, nilai yang dimasukkan oleh pengguna diubah menjadi integer menggunakan Convert.ToInt32().
Misalkan pengguna memasukkan 39.

Convert.ToInt32(Console.ReadLine())

Menjadi:
Convert.ToInt32(“39”).

Hasil dari Convert.ToInt32(“39”) adalah integer 39. Integer ini kemudian ditetapkan ke variabel userAge.

Hal berikutnya yang perlu diperhatikan tentang program ini adalah Baris 12 seperti yang ditunjukkan di bawah ini:
Console.WriteLine(“Hello World! My name is {0} and I am {1} yearsold. I was born in {2}.”, userName, userAge, currentYear - userAge);

Perhatikan bahwa argumen terakhir (currentYear - userAge) melibatkan operasi Matematika? Ini diperbolehkan di C#. WriteLine() akan melakukan pengurangan dan menampilkan hasil perhitungan.

BAB 6
MEMBUAT PILIHAN DAN KEPUTUSAN

Selamat telah berhasil sejauh ini. Kita telah menempuh perjalanan jauh. Anda sekarang mengetahui berbagai tipe data dalam C# dan dapat membuat kode program sederhana yang berinteraksi dengan pengguna. Dalam bab ini, kita akan membahas konsep dasar lain dalam pemrograman; kita akan belajar bagaimana mengontrol aliran program menggunakan pernyataan aliran kontrol. Secara khusus, kita akan belajar tentang pernyataan if, pernyataan if sebaris, pernyataan switch, perulangan for, perulangan foreach, perulangan while dan perulangan do while. Selain itu, kita juga akan mempelajari pernyataan try-catch-finally yang mengontrol alur program ketika terjadi kesalahan. Namun, sebelum kita masuk ke alat kontrol ini, kita harus terlebih dahulu melihat pernyataan kondisi.

6.1 PERNYATAAN KONDISI

Sebagian besar pernyataan aliran kontrol melibatkan evaluasi pernyataan kondisi. Program akan berjalan secara berbeda tergantung pada apakah kondisi terpenuhi. Pernyataan kondisi yang paling umum adalah pernyataan perbandingan. Jika kita ingin membandingkan apakah dua variabel itu sama, kita menggunakan tanda == (ganda =). Misalnya, jika Anda menulis x == y, Anda meminta program untuk memeriksa apakah nilai x sama dengan nilai y. Jika keduanya sama, kondisi terpenuhi dan pernyataan bernilai true. Jika tidak, pernyataan tersebut bernilai false. Selain mengevaluasi apakah satu nilai sama dengan yang lain, ada operator perbandingan lain yang dapat kita gunakan dalam pernyataan kondisi kita.

Tidak sama (!=)

Mengembalikan nilai true jika kiri tidak sama dengan kanan

5 != 2 benar

6 != 6 salah

Lebih besar dari (>)

Mengembalikan nilai true jika kiri lebih besar dari kanan

5 > 2 benar

3 > 6 salah

Lebih kecil dari (<)

Mengembalikan nilai true jika kiri lebih kecil dari kanan

1 < 7 benar

9 < 6 salah

Lebih besar dari atau sama dengan (>=)

Mengembalikan nilai true jika kiri lebih besar dari atau sama dengan kanan

5 >= 2 benar

5 >= 5 benar

3 >= 6 salah

Lebih kecil dari atau sama dengan (<=)

Mengembalikan nilai true jika kiri lebih kecil dari atau sama dengan kanan

11 <= 7 benar

7 <= 7 benar

9 <= 6 salah

Kita juga memiliki tiga operator logika (&&, ||, !) yang berguna jika kita ingin menggabungkan beberapa kondisi.

Operator AND (&&)

Mengembalikan nilai true jika semua kondisi terpenuhi

5==5 && 2>1 && 3!=7 benar

5==5 && 2<1 && 3!=7 salah karena kondisi kedua (2<1) salah

Operator ATAU (||)

Mengembalikan nilai true jika setidaknya satu kondisi terpenuhi.

5==5 || 2<1 || 3==7 benar karena kondisi pertama (5==5) benar 5==6 || 2<1 || 3==7 salah karena semua kondisi salah

6.2 PERNYATAAN KONTROL ALIRAN

Sekarang kita sudah familiar dengan pernyataan kondisi, mari kita lanjutkan untuk mempelajari bagaimana kita dapat menggunakan pernyataan ini untuk mengontrol aliran program.

Pernyataan IF

Pernyataan if adalah salah satu pernyataan aliran kontrol yang paling umum digunakan. Hal ini memungkinkan program untuk mengevaluasi jika kondisi tertentu terpenuhi, dan untuk melakukan tindakan yang sesuai berdasarkan hasil evaluasi. Struktur pernyataan if adalah sebagai berikut (nomor baris ditambahkan untuk referensi):

```
1 if (condition 1 is met)
2 {
3 do Task A
4 }
5 else if (condition 2 is met)
6 {
7 do Task B
8 }
9 else if (condition 3 is met)
10 {
11 do Task C
12 }
13 else
14 {
15 do Task E
16 }
```

Baris 1 menguji kondisi pertama. Jika kondisi terpenuhi, semua yang ada di dalam pasangan kurung kurawal yang mengikuti (baris 2 hingga 4) akan dieksekusi. Pernyataan if lainnya (dari baris 5 hingga 16) akan dilewati. Jika kondisi pertama tidak terpenuhi, Anda dapat menggunakan pernyataan else if yang mengikutinya untuk menguji kondisi lainnya (baris 5 hingga 12). Mungkin ada beberapa pernyataan if lainnya. Terakhir, Anda dapat menggunakan pernyataan else (baris 13 hingga 16) untuk mengeksekusi beberapa kode jika tidak ada kondisi sebelumnya yang terpenuhi.

Untuk memahami sepenuhnya bagaimana pernyataan if bekerja, tambahkan kode berikut ke Main() program dalam template VSC.

```
int userAge;

Console.Write("Please enter your age: ");
userAge = Convert.ToInt32(Console.ReadLine());

if (userAge < 0 || userAge > 100)
{
    Console.WriteLine("Invalid Age");
    Console.WriteLine("Age must be between 0 and 100");
}
else if (userAge < 18)
    Console.WriteLine("Sorry you are underage");
else if (userAge < 21)
    Console.WriteLine("You need parental consent");
else
{
    Console.WriteLine("Congratulations!");
    Console.WriteLine("You may sign up for the event!");
}
```

Program pertama-tama meminta pengguna untuk usianya dan menyimpan hasilnya di variabel userAge.

Selanjutnya pernyataan:
if (userAge < 0 || userAge > 100)

memeriksa apakah nilai userAge lebih kecil dari nol atau lebih besar dari 100. Jika salah satu dari kondisi tersebut benar, program akan mengeksekusi semua pernyataan dalam kurung kurawal yang mengikutinya. Dalam contoh ini, itu akan mencetak "Usia Tidak Valid", diikuti oleh "Usia harus antara 0 dan 100".

Di sisi lain, jika kedua kondisi false, program akan menguji kondisi berikutnya - else if (userAge < 18). Jika userAge kurang dari 18 (tetapi lebih dari atau sama dengan 0 karena kondisi pertama tidak terpenuhi), program akan mencetak "Maaf Anda di bawah umur". Anda mungkin memperhatikan bahwa kita tidak menyertakan pernyataan:
Console.WriteLine(“Sorry you are underage”);

dalam kurung kurawal. Ini karena kurung kurawal adalah opsional jika hanya ada satu pernyataan yang akan dieksekusi.

Jika pengguna tidak memasukkan nilai yang lebih kecil dari 18, tetapi memasukkan nilai yang lebih besar atau sama dengan 18 tetapi lebih kecil dari 21, pernyataan if else berikutnya akan dieksekusi. Dalam hal ini, pesan "Anda memerlukan persetujuan orang tua" akan dicetak. Terakhir, jika pengguna memasukkan nilai yang lebih besar atau sama dengan 21 tetapi lebih kecil dari atau sama dengan 100, program akan mengeksekusi kode di blok else. Dalam hal ini, itu akan mencetak "Selamat" diikuti oleh "Anda dapat mendaftar untuk acara itu!".

Jalankan program lima kali dan masukkan -1, 8, 20, 23 dan 121 masing-masing untuk setiap proses. Anda akan mendapatkan output berikut:

```
Please enter your age: -1
Invalid Age
Age must be between 0 and 100

Please enter your age: 8
Sorry you are underage

Please enter your age: 20
You need parental consent

Please enter your age: 23
Congratulations!
You may sign up for the event!

Please enter your age: 121
Invalid Age
Age must be between 0 and 100
```

If sebaris

Pernyataan if sebaris adalah bentuk sederhana dari pernyataan if yang sangat nyaman jika Anda ingin menetapkan nilai ke variabel tergantung pada hasil suatu kondisi. Sintaksnya adalah:

condition ? value if condition is true : value if condition isfalse;

Misalnya, pernyataan:

3>2 ? 10 : 5;

mengembalikan nilai 10 karena 3 lebih besar dari 2 (yaitu kondisi 3 > 2 benar). Nilai ini kemudian dapat diberikan ke variabel.

Jika kita menulis:

int myNum = 3>2 ? 10 : 5;

myNum akan diberi nilai 10.

Pernyataan Switch

Pernyataan switch mirip dengan pernyataan if kecuali bahwa itu tidak bekerja dengan rentang nilai. Pernyataan switch mengharuskan setiap kasus didasarkan pada satu nilai. Bergantung pada nilai variabel yang digunakan untuk berpindah, program akan mengeksekusi blok kode yang benar.

Sintaks dari pernyataan switch adalah sebagai berikut:

```
switch (variable used for switching)
{
    case firstCase:
    do A;
    break (or other jump statements);

    case secondCase:
    do B;
    break (or other jump statements);

    case default:
    do C;
    break (or other jump statements);
}
```

Anda dapat memiliki kasus sebanyak yang Anda inginkan saat menggunakan pernyataan switch. Itu kasus default adalah opsional dan dijalankan jika tidak ada kasus lain yang berlaku. Ketika kasus tertentu terpenuhi, semuanya mulai dari baris berikutnya dijalankan sampai pernyataan lompat tercapai. Pernyataan lompat adalah pernyataan yang menginstruksikan kompiler untuk melompat ke baris lain dalam program. Kita akan melihat pernyataan lompatan secara lebih mendalam nanti. Pernyataan lompatan yang paling umum digunakan adalah break; penyataan.

Mari kita lihat contoh bagaimana pernyataan switch bekerja.

```
1 Console.Write("Enter your grade: ");
2 string userGrade = Console.ReadLine();
3
4 switch (userGrade)
5 {
6 case "A+":
7 case "A":
8 Console.WriteLine("Distinction");
9 break;
10 case "B":
11 Console.WriteLine("B Grade");
12 break;
13 case "C":
14 Console.WriteLine("C Grade");
15 break;
16 default:
17 Console.WriteLine("Fail");
18 break;
19 }
```

Program pertama meminta pengguna untuk nilainya. Jika nilai adalah “A+” (Baris 6), program akan mengeksekusi pernyataan berikutnya hingga mencapai jeda; penyataan. Ini berarti akan mengeksekusi Baris 7 hingga 9. Jadi outputnya adalah "Distinction". Jika grade adalah “A” (Baris 7), program akan mengeksekusi Baris 8 dan 9. Demikian pula, outputnya adalah “Distinction”.

Jika nilai bukan “A+” atau “A”, program akan memeriksa kasus berikutnya. Itu terus memeriksa dari atas ke bawah sampai kasus puas. Jika tidak ada kasus yang berlaku, kasus default akan dieksekusi. Jika Anda menjalankan kode di atas, Anda akan mendapatkan output berikut untuk setiap input yang ditampilkan:

```
Enter your grade: A+
Distinction

Enter your grade: A
Distinction

Enter your grade: B

B Grade

Enter your grade: C
C Grade

Enter your grade: D
Fail

Enter your grade: Hello
Fail
```

Loop For

Loop for mengeksekusi blok kode berulang kali hingga kondisi pengujian tidak lagi valid. Sintaks untuk perulangan for adalah sebagai berikut:

```
for (initial value; test condition;  modification to value)
{
        //Do Some Task
}
```

Untuk memahami cara kerja for loop, mari perhatikan contoh di bawah ini:

```
1  for (int i = 0; i < 5; i++)
2  {
3  Console.WriteLine(i);
4  }
```

Fokus utama dari for loop adalah Baris 1:

for (int i = 0; i < 5; i++)

Ada tiga bagian untuk itu, masing-masing dipisahkan oleh titik koma.

Bagian pertama mendeklarasikan dan menginisialisasi variabel int i ke nol. Variabel ini berfungsi sebagai penghitung loop. Bagian kedua menguji apakah i lebih kecil dari 5. Jika ya, pernyataan di dalam kurung kurawal akan dieksekusi. Dalam contoh ini, kurung kurawal adalah opsional karena hanya ada satu pernyataan. Setelah mengeksekusi pernyataan WriteLine(), program kembali ke segmen terakhir di Baris 1. i++ menambah nilai i dengan 1. Oleh karena itu, i meningkat dari 0 menjadi 1. Setelah kenaikan, program menguji apakah nilai baru i masih lebih kecil dari 5. Jika ya, ia akan mengeksekusi pernyataan WriteLine() sekali lagi.

Proses pengujian dan penambahan penghitung loop ini diulangi sampai kondisi i < 5 tidak lagi benar. Pada titik ini, program keluar dari perulangan for dan terus mengeksekusi perintah lain setelah perulangan for. Output untuk segmen kode adalah:

0

1

2

3
4
Output berhenti pada 4 karena ketika i adalah 5, pernyataan WriteLine() tidak dieksekusi karena 5 tidak lebih kecil dari 5. Perulangan for biasanya digunakan untuk mengulang array atau daftar. Misalnya, jika kita memiliki:
int[] myNumbers = { 10, 20, 30, 40, 50 };
kita dapat menggunakan for loop dan properti Length dari array untuk mengulang array seperti yang ditunjukkan di bawah ini.

```
for (int i = 0; i < myNumbers.Length; i++)
{
        Console.WriteLine(myNumbers[i]);
}
```

Karena myNumbers.Length sama dengan 5, kode ini berjalan dari i = 0 hingga i = 4. Jika kita menjalankan kode, kita akan mendapatkan output berikut:
10
20
30
40
50

Loop Foreach

Selain loop for, kita juga dapat menggunakan loop foreach saat bekerja dengan array dan daftar. Loop foreach sangat berguna jika Anda ingin mendapatkan informasi dari larik atau daftar, tanpa membuat perubahan apa pun.
Misalkan Anda memiliki:
char[] message = { ‘H’, ‘e’, ‘l’, ‘l’, ‘o’ };

Anda dapat menggunakan kode berikut untuk menampilkan elemen array.
foreach (char i in message)Console.Write(i);

Pada kode di atas, kita memiliki variabel char i yang digunakan untuk perulangan. Setiap kali loop berjalan, sebuah elemen dalam array pesan ditugaskan ke variabel i. Misalnya, pertama kali loop berjalan, karakter 'H' ditugaskan ke i.
Garis
Console.Write(i);
kemudian mencetak huruf 'H'.

Kali kedua loop berjalan, karakter 'e' ditugaskan ke i. Garis
Console.Write(i);
mencetak huruf 'e'.
Ini berlanjut sampai semua elemen dalam array telah dicetak.

While Loop

Seperti namanya, while loop berulang kali mengeksekusi instruksi di dalam loop sementara kondisi tertentu tetap valid. Struktur pernyataan while adalah sebagai berikut:

```
while (condition is true)
{
        do A
}
```

Sebagian besar waktu saat menggunakan perulangan while, kita harus mendeklarasikan variabel terlebih dahulu ke berfungsi sebagai penghitung putaran. Sebut saja penghitung variabel ini. Kode di bawah ini menunjukkan contoh cara kerja while loop:

```
int counter = 5;
while (counter > 0)
  {
        Console.WriteLine("Counter = {0}", counter); counter =
        counter – 1;
  }
```

Jika Anda menjalankan kode, Anda akan mendapatkan output berikut:

```
Counter   = 5
Counter   = 4
Counter   = 3
Counter   = 2
Counter   = 1
```

Pernyataan while memiliki sintaks yang relatif sederhana. Pernyataan di dalam kurung kurawal dieksekusi selama penghitung > 0. Perhatikan bahwa kita memiliki penghitung garis = penghitung – 1 di dalam kurung kurawal? Garis ini sangat penting. Ini mengurangi nilai penghitung sebesar 1 setiap kali loop dijalankan. Kita perlu mengurangi nilai counter sebesar 1 sehingga kondisi loop (counter > 0) pada akhirnya akan bernilai false. Jika kita lupa melakukannya, loop akan terus berjalan tanpa henti, menghasilkan infinite loop. Program akan terus mencetak counter = 5 sampai Anda entah bagaimana mematikan program tersebut. Bukan pengalaman yang menyenangkan terutama jika Anda memiliki program besar dan Anda tidak tahu segmen kode mana yang menyebabkan infinite loop.

Do while

Perulangan do while mirip dengan perulangan while dengan satu perbedaan utama - kode di dalam kurung kurawal dari perulangan do while dieksekusi setidaknya sekali. Berikut adalah contoh cara kerja do while loop:

```
int counter = 100;
do {
        Console.WriteLine("Counter = {0}", counter)
counter++;
} while (counter<0);
```

Karena kondisi pengujian (while (counter<0)) ditempatkan setelah kurung kurawal penutup, itu diuji setelah kode di dalam kurung kurawal dieksekusi setidaknya sekali. Jika Anda menjalankan kode di atas, Anda akan mendapatkan:
Counter = 100;

Setelah pernyataan WriteLine() dijalankan untuk pertama kalinya, pencacah bertambah 1. Nilai pencacah sekarang 101. Ketika program mencapai kondisi pengujian, pengujian gagal karena pencacah tidak lebih kecil dari 0. Program kemudian akan keluar dari lingkaran. Meskipun nilai awal pencacah tidak memenuhi kondisi pengujian (penghitung < 0), kode di dalam kurung kurawal tetap dieksekusi satu kali. Perhatikan bahwa untuk pernyataan do while, titik koma (;) diperlukan setelah kondisi pengujian.

6.3 PERNYATAAN LANGSUNG

Kita sekarang telah membahas sebagian besar pernyataan aliran kontrol di C#. Selanjutnya, mari kita lihat pernyataan jump. Pernyataan lompat adalah pernyataan yang menginstruksikan program untuk menyimpang dari urutan aliran normal dan melompat ke baris kode lain. Pernyataan lompat biasanya digunakan dalam perulangan dan pernyataan aliran kontrol lainnya.

Break

Kata kunci break menyebabkan program keluar dari loop sebelum waktunya ketika kondisi tertentu terpenuhi. Kita telah melihat bagaimana kata kunci break dapat digunakan dalam pernyataan switch. Sekarang, mari kita lihat contoh bagaimana kata kunci break dapat digunakan dalam perulangan for.
Perhatikan segmen kode di bawah ini:

```
1 1 int i =
  0;
2
3 for (i = 0; i < 5; i++
4 {
5 Console.WriteLine("i = {0}", i);
6 if (i == 2)
7 break;
8 }
```

Dalam contoh ini, kita menggunakan pernyataan if di dalam perulangan for. Sangat umum bagi kita untuk 'mencampur-dan-mencocokkan' berbagai alat kontrol dalam pemrograman, seperti menggunakan perulangan while di dalam pernyataan if atau menggunakan perulangan for di dalam perulangan while. Ini dikenal sebagai pernyataan kontrol bersarang. Jika Anda menjalankan segmen kode di atas, Anda akan mendapatkan output berikut:
i = 0
i = 1
i = 2

Perhatikan bahwa loop berakhir sebelum waktunya di i = 2? Tanpa kata kunci break, loop harus dijalankan dari i = 0 hingga i = 4 karena kondisi loop adalah i < 5. Namun dengan kata kunci break, ketika i = 2, kondisi pada baris 6 bernilai true. Kata kunci break pada baris 7 kemudian menyebabkan loop berakhir sebelum waktunya.

Continue

Kata kunci lompat lain yang umum digunakan adalah kata kunci lanjutan. Saat kita menggunakan lanjutkan, sisa loop setelah kata kunci dilewati untuk iterasi itu. Sebuah contoh akan membuatnya lebih jelas. Jika Anda menjalankan segmen kode di bawah ini

```
  for (int i = 0; i<5; i++)
  {
        Console.WriteLine(“i = {0}”, i);if (i == 2)
              continue;
        Console.WriteLine(“I will not be printed if i=2.\n”);
}
```

Anda akan mendapatkan output berikut:

```
i = 0
I will not be printed if i=2.

i = 1
I will not be printed if i=2.

i = 2
i = 3
I will not be printed if i=2.

i = 4
I will not be printed if i=2.
```

Ketika i = 2, baris setelah kata kunci continue tidak dieksekusi. Selain itu, semuanya berjalan seperti biasa.

6.4 PENANGANAN PENGECUALIAN

Kita sekarang telah belajar bagaimana mengontrol aliran program dalam keadaan 'normal' menggunakan pernyataan aliran kontrol dan pernyataan lompat. Sebelum kita mengakhiri bab ini, kita perlu melihat satu pernyataan kontrol terakhir, pernyataan try-catch-final. Pernyataan try-catch-finally mengontrol bagaimana program berjalan ketika terjadi kesalahan. Sintaksnya adalah sebagai berikut:

```
try
{
    do something
}
catch (type of error)
{
    do something else when an error occurs
}
finally
{
    do this regardless of whether the try or catch condition is
met.
}
```

Anda dapat memiliki lebih dari satu blok tangkapan. Selain itu, blok terakhir adalah opsional. Mari kita pertimbangkan sebuah contoh.

```
int numerator, denominator;

Console.Write("Please enter the numerator: ");
numerator = Convert.ToInt32(Console.ReadLine());

Console.Write("Please enter the denominator: ");
denominator = Convert.ToInt32(Console.ReadLine());

try
{
    Console.WriteLine("The result is {0}.", numerator/denominator);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
finally
{
    Console.WriteLine(“---- End of Error Handling Example ----”);
}
```

Jika Anda menjalankan kode dan memasukkan 12 dan 4, Anda akan mendapatkan pesan:

The result is 3.

---- End of Error Handling Example ----

Dalam contoh ini, kode di blok try berhasil dieksekusi. Setelah kode di blok try dieksekusi, kode di blok akhirnya dieksekusi. Sekarang misalkan Anda memasukkan 12 dan 0 sebagai gantinya. Anda akan mendapatkan:

Mencoba membagi dengan nol.

Attempted to divide by zero.

---- End of Error Handling Example ----

Dalam hal ini, kode di blok catch dieksekusi sebagai gantinya. Ini karena ketika program mencoba mengeksekusi pernyataan di blok try, terjadi kesalahan karena Anda tidak dapat membagi angka dengan nol. Oleh karena itu, pernyataan di blok catch dieksekusi. Selain itu, kode di blok akhirnya juga dieksekusi. Blok akhirnya selalu dieksekusi terlepas dari apakah coba atau tangkap blok dieksekusi. Blok catch memungkinkan kita untuk menentukan jenis

kesalahan yang harus ditangkap. Dalam hal ini, kita mencoba menangkap kesalahan umum. Oleh karena itu, kita menulis:
catch (Exception e)

di mana Pengecualian merujuk ke kelas tempat kesalahan itu berada dan e adalah namanya diberikan pada kesalahan. Kelas Pengecualian menangani semua kesalahan umum dan memiliki properti yang disebut Pesan yang menjelaskan alasan pengecualian. Untuk menampilkan properti tersebut, kita menulis Console.WriteLine(e.Message);

6.5 KESALAHAN SPESIFIK

Selain kelas Pengecualian yang menangani kesalahan umum, kita juga memiliki kelas lain yang dapat menangani kesalahan yang lebih spesifik. Ini berguna jika Anda ingin melakukan tugas tertentu tergantung pada kesalahan yang tertangkap. Misalnya, Anda mungkin ingin menampilkan pesan kesalahan Anda sendiri.
Coba jalankan kode di bawah ini:

```
int choice = 0;
int[] numbers = { 10, 11, 12, 13, 14, 15 };
Console.Write("Please enter the index of the array: ");

try
{
    choice = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("numbers[{0}] = {1}", choice,
numbers[choice]);
}catch (IndexOutOfRangeException)
{
    Console.WriteLine("Error: Index should be from 0 to 5.");
}catch (FormatException)
{
    Console.WriteLine("Error: You did not enter an integer.");
}catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

Jika Anda masuk:
10

Kamu akan mendapatkan:
Index was outside the bounds of the array.
Index should be from 0 to 5.
Jika Anda enter:
Hello

Kamu akan mendapatkan:
Input string was not in a correct format.
You did not enter an integer.

Kesalahan pertama adalah pengecualian IndexOutOfRangeException dan ditangani oleh blok tangkapan pertama. Pengecualian ini terjadi saat Anda mencoba mengakses elemen larik dengan indeks yang berada di luar batasnya.

Kesalahan kedua adalah pengecualian FormatException dan ditangani oleh blok tangkapan kedua. Pengecualian FormatException terjadi ketika format argumen tidak valid. Dalam contoh kita, Convert.ToInt32("Halo") menghasilkan pengecualian FormatException karena argumen "Halo" tidak dapat diubah menjadi bilangan bulat. Sebaliknya, jika Anda memasukkan 4, Convert.ToInt32("4") tidak akan menghasilkan pengecualian FormatException karena string "4" dapat diubah menjadi bilangan bulat. Setelah dua blok tangkap tertentu, kita memiliki satu blok tangkap lagi untuk menangkap kesalahan umum apa pun yang tidak kita hapus sebelumnya. Contoh di atas menunjukkan dua dari banyak pengecualian dalam C#. Untuk daftar lengkap semua pengecualian, lihat https://msdn.microsoft.com/en-us/library/system.systemexception.aspx.

BAB 7
PEMROGRAMAN BERORIENTASI OBJEK BAGIAN 1

Kita telah membahas sedikit sejauh ini. Dalam dua bab berikutnya, kita akan melihat konsep penting lainnya dalam pemrograman – konsep pemrograman berorientasi objek. Dalam bab ini, kita akan mempelajari apa itu pemrograman berorientasi objek dan bagaimana menulis kelas kita sendiri dan membuat objek darinya. Selain itu, kita juga akan membahas konsep bidang, properti, konstruktor, dan metode.

7.1 APA ITU PEMROGRAMAN BERORIENTASI OBJEK?

Secara sederhana, pemrograman berorientasi objek adalah pendekatan pemrograman yang memecah masalah pemrograman menjadi objek yang berinteraksi satu sama lain. Objek dibuat dari template yang dikenal sebagai kelas. Anda dapat menganggap kelas sebagai cetak biru sebuah bangunan. Objek adalah "bangunan" aktual yang kita bangun berdasarkan cetak biru.

7.2 MENULIS KELAS KITA SENDIRI

Untuk menulis kelas kita sendiri, kita menggunakan kata kunci class, diikuti dengan nama kelasnya.

Misalnya, untuk membuat kelas Staf, kita menulis:

```
class Staff {
        //Contents of the class
        //including fields, properties and methods
}
```

Ini adalah praktik umum untuk menggunakan PascalCasing saat menamai kelas kita. PascalCasing mengacu pada praktik penggunaan huruf kapital pada huruf pertama setiap kata, termasuk kata pertama (mis. ThisIsAClassName). Ini adalah konvensi yang akan kita ikuti dalam buku ini.

Isi kelas diapit oleh sepasang kurung kurawal yang mengikuti nama kelas. Isi kelas termasuk konstruktor, destruktor, konstanta, bidang, metode, properti, pengindeks, operator, acara, delegasi, antarmuka, struct, dan kelas lainnya. Kita akan membahas beberapa elemen kelas yang lebih umum dalam bab ini, yaitu bidang, metode, properti, dan konstruktor. Untuk memahami apa ini, mari buat kelas dari awal. Pertama, buat proyek Aplikasi Konsol baru di Komunitas Visual Studio dan beri nama proyek ini "ClassDemo". Pelajari kode yang dibuat secara otomatis untuk Anda. Perhatikan bahwa di dalam namespace ClassDemo, VSC telah membuat kelas bernama Program untuk Anda? Di dalam kelas Program, kita memiliki metode Main().

Secara default, metode Main() (yang merupakan titik awal untuk semua aplikasi C#) dimasukkan ke dalam kelas Program yang dibuat oleh VSC. Jika kita mau, kita bisa mengubah nama kelas Program menjadi sesuatu yang lain, tetapi metode Main() harus dipanggil Main().

Metode Main() harus ada di semua program C#. Dalam bab ini, kita akan menambahkan kelas kedua ke namespace ClassDemo. Kita akan memanggil kelas baru ini Staf dan menambahkan bidang, properti, dan metode ke kelas.

Kode lengkap untuk bab ini dapat diunduh di http://www.learncodingfast.com/csharp. Mari kita mendeklarasikan kelas terlebih dahulu. Tambahkan kode berikut tepat sebelum baris kelas Program dalam kode yang dibuat secara otomatis.

```
class Staff
{
}
```

Kita sekarang memiliki dua kelas dalam proyek kita: Staf dan Program.

7.3 FIELDS

Di dalam kelas Staff, tambahkan baris berikut:

```
private string nameOfStaff;
private const int hourlyRate = 30;
private int hWorked;
```

Di sini, kita mendeklarasikan satu variabel string (nameOfStaff) dan dua variabel int (hourlyRate dan hWorked). Variabel ini dikenal sebagai bidang kelas. Bidang hanyalah variabel yang dideklarasikan di dalam kelas. Seperti variabel lainnya, mereka digunakan untuk menyimpan data. Perhatikan bahwa ada kata pribadi di depan setiap pernyataan deklarasi? Ini dikenal sebagai pengubah akses. Pengubah akses seperti penjaga gerbang, mereka mengontrol siapa yang memiliki akses ke bidang itu (yaitu siapa yang dapat membaca dan mengubah nilai bidang itu). Bidang dapat berupa pribadi, publik, dilindungi atau internal. Dalam kasus kita, kita mendeklarasikan tiga bidang sebagai pribadi. Ini berarti mereka hanya dapat diakses dari dalam kelas Staf itu sendiri.

Ada dua alasan mengapa kita tidak ingin ketiga bidang dapat diakses di luar kelas. Alasan pertama adalah karena kelas lain tidak perlu tahu tentang bidang tersebut. Dalam kasus kita, field hourlyRate hanya diperlukan dalam kelas Staff. Kita memiliki metode di dalam kelas Staff yang menggunakan field hourlyRate untuk menghitung gaji bulanan seorang karyawan. Kelas lain tidak menggunakan bidang hourlyRate sama sekali. Oleh karena itu, adalah tepat untuk mendeklarasikan hourlyRate sebagai private untuk menyembunyikan bidang ini dari kelas lain. Ini dikenal sebagai enkapsulasi. Enkapsulasi memungkinkan objek untuk menyembunyikan data dan perilaku dari kelas lain yang tidak perlu mengetahuinya. Ini memudahkan kita untuk membuat perubahan pada kode kita di masa mendatang jika perlu. Kita dapat dengan aman mengubah nilai hourlyRate di dalam kelas Staff tanpa mempengaruhi kelas lain.

Alasan kedua untuk mendeklarasikan sebuah field sebagai private adalah karena kita tidak ingin class lain memodifikasinya secara bebas. Ini membantu untuk mencegah bidang dari yang rusak. Kita akan berbicara lebih banyak tentang pengubah akses di bab berikutnya.

Selain kata kunci pribadi, kita juga menambahkan kata kunci const ketika kita mendeklarasikan hourlyRate field.
private const int hourlyRate = 30;

Kata kunci const menunjukkan bahwa nilai tidak dapat diubah setelah dibuat. Setiap variabel yang dideklarasikan sebagai const harus diinisialisasi pada titik deklarasi. Dalam contoh kita, kita menginisialisasi hourlyRate menjadi 30. Nilai ini tidak dapat diubah selanjutnya di mana pun dalam kode.

7.4 PROPERTI

Selanjutnya, mari kita lihat properti. Properti biasanya digunakan untuk menyediakan akses ke bidang pribadi dalam kasus di mana bidang tersebut dibutuhkan oleh kelas lain. Ini mungkin terdengar seperti kontradiksi. Sebelumnya, kita menyebutkan bahwa kita menggunakan bidang pribadi sehingga kelas lain tidak memiliki akses ke sana. Jika demikian, mengapa kita mengizinkan akses ke mereka melalui properti? Salah satu alasan utama adalah bahwa menggunakan properti memberi kita kontrol lebih besar atas hak apa yang dimiliki kelas lain saat menilai bidang pribadi ini. Kita akan melihat bagaimana melakukannya nanti.

Untuk saat ini, pertama-tama mari kita pelajari cara mendeklarasikan properti. Tambahkan baris kode berikut ke kelas Staff kita, tepat setelah baris privateint hWorked;.

```
public int HoursWorked
{
    get
    {
        return hWorked;
    }
    set
    {
      if (value > 0)
      hWorked = value;
      else
      hWorked = 0;
    }
}
```

Kita mendeklarasikan properti kita sebagai
public int HoursWorked
{
}

Pengubah akses bersifat publik karena kita ingin kelas lain memiliki akses ke properti ini. Tipe datanya adalah int karena properti ini digunakan untuk menyediakan akses ke private int bidang hWorked. hWorked dikenal sebagai bidang pendukung properti. Nama propertinya adalah HoursWorked. Kita biasanya menggunakan PascalCasing untuk nama properti. Properti berisi dua metode khusus yang dikenal sebagai pengakses. Accessor pertama adalah getter dan yang kedua adalah setter. Pengambil dasar hanya mengembalikan nilai bidang pribadi. Oleh karena itu, kita menulis:
Get

```
{
        Return hWorked;
}
```

di mana return adalah kata kunci dan hWorked adalah nama bidang pendukung. Setter menetapkan nilai bidang pribadi. Kita menulis:

```
set
{
        If (value > 0)
                hWorked = value;
        else
                hWorked = 0;
}
```

value adalah kata kunci ketika digunakan di dalam setter. Ini mengacu pada nilai yang ada di sisi kanan pernyataan penetapan saat pengguna menggunakan properti untuk mengatur nilai bidang pribadi. Kita akan belajar bagaimana melakukannya nanti.

Di dalam setter, kita melakukan pemeriksaan sederhana menggunakan pernyataan if. Kita memeriksa apakah nilainya lebih dari nol. Jika ya, kita menetapkannya ke hWorked. Lain, kita mengatur hWorked ke nol. Setter ini menunjukkan bagaimana kita dapat menggunakan properti untuk mengontrol nilai apa yang dapat ditetapkan ke bidang pribadi kita. Secara default, pengambil dan penyetel memiliki tingkat akses yang sama dengan properti itu sendiri (dalam hal ini publik). Oleh karena itu, kita tidak perlu menentukannya. Namun, jika Anda tidak ingin penyetel memiliki tingkat akses yang sama dengan properti, Anda dapat mendeklarasikan penyetel sebagai pribadi sehingga kelas lain tidak dapat mengubah bidang pribadi Anda:

```
private set
  {
}
```

Properti ini kemudian menjadi properti baca-saja di luar kelas Staf. Nilainya hanya dapat diatur dalam kelas Staf itu sendiri.

Properti yang diterapkan secara otomatis

Perhatikan bahwa dalam kasus di mana tidak ada logika tambahan yang diperlukan dalam pengambil dan penyetel, C# memberi kita singkatan untuk mendeklarasikan properti. Ini dikenal sebagai properti yang diterapkan secara otomatis. Untuk mendeklarasikan properti yang diimplementasikan secara otomatis, kita menulis:

```
public int HoursWorked { get; set; }
```

Ini setara dengan:

```
private int hWorked;
public int HoursWorked
{
    get
    {
        return hWorked;
    }
    set
    {
        hWorked = value;
    }
}
```

Saat Anda menggunakan singkatan ini, Anda tidak perlu mendeklarasikan bidang pribadi. Kompiler akan membuat bidang dukungan pribadi anonim untuk Anda secara otomatis.

Jika Anda ingin menjadikan properti hanya-baca, Anda dapat menyetel penyetel ke pribadi seperti ini:

public int HoursWorked { get; private set; }

7.5 METODE

Selanjutnya, mari kita lihat metode. Metode adalah blok kode yang melakukan tugas tertentu. Mari tambahkan metode sederhana ke kelas Staff kita.

```
public void PrintMessage()
{
    Console.WriteLine("Calculating Pay…");
}
```

Metode ini dinyatakan sebagai:

```
public void PrintMessage()
{
}
```

Deklarasi metode pertama-tama menyatakan tingkat aksesibilitas metode. Di sini kita mendeklarasikan metode sebagai publik sehingga metode tersebut dapat diakses di mana saja dalam program (tidak hanya di dalam kelas Staf). Selanjutnya, kita menyatakan tipe pengembalian metode. Sebuah metode dapat mengembalikan hasil tertentu setelah melakukan tugasnya. Jika metode tidak mengembalikan hasil apa pun, kita menggunakan kata kunci void seperti pada contoh kita. Akhirnya, kita menyatakan nama metode (PrintMessage dalam contoh kita).

Tanda kurung () setelah nama metode adalah tempat kita menyertakan parameter metode. Parameter adalah nama yang diberikan pada data yang kita berikan ke metode agar dapat melakukan tugasnya. Jika metode tidak memerlukan data (seperti dalam contoh kita), kita cukup menambahkan sepasang tanda kurung kosong setelah nama metode. Setelah mendeklarasikan metode, kita mendefinisikan apa yang dilakukannya di dalam pasangan

kurung kurawal yang mengikutinya. Ini dikenal sebagai menerapkan metode. Dalam contoh kita, metode PrintMessage() hanya mencetak baris "Calculating Pay…". Itu saja yang ada pada metode PrintMessage().

Selanjutnya, mari kita beralih ke metode yang lebih kompleks. Metode kedua ini menghitung gaji setiap karyawan dan mengembalikan hasil perhitungannya. Tambahkan baris kode berikut ke Staff.

```
{

     PrintMessage();

     int staffPay;
     staffPay = hWorked * hourlyRate ;

     if (hWorked > 0)
         return staffPay;
     else
       return 0;
}
```

Metode ini dinyatakan sebagai

```
public int CalculatePay()
{
}
```

Kata kunci int menunjukkan bahwa metode ini mengembalikan nilai yang bertipe int. Di dalam kurung kurawal, kita memiliki pernyataan:

```
PrintMessage();
```

Ini dikenal sebagai memanggil metode PrintMessage(). Saat program mencapai pernyataan ini, program akan mengeksekusi metode PrintMessage() terlebih dahulu dan mencetak baris "Calculating Pay…" sebelum menjalankan metode HitungPay() lainnya. Contoh ini menunjukkan bagaimana Anda dapat memanggil satu metode di dalam metode lain. Selanjutnya, kita mendeklarasikan variabel lokal yang disebut staffPay dan menetapkan produk dari bidang pribadi HourlyRate dan hWorked ke dalamnya.

Sebuah metode dapat mengakses semua bidang dan properti yang dideklarasikan di dalam kelas. Selain itu, ia dapat mendeklarasikan variabelnya sendiri. Ini dikenal sebagai variabel lokal dan hanya ada di dalam metode. Contohnya adalah variabel staffPay dalam contoh kita. Setelah menetapkan variabel staffPay, kita menggunakan pernyataan if untuk menentukan hasil apa yang harus dikembalikan oleh metode.

Sebuah metode biasanya memiliki setidaknya satu pernyataan kembali. return adalah kata kunci yang digunakan untuk mengembalikan jawaban dari metode. Mungkin ada lebih dari satu pernyataan pengembalian dalam suatu metode. Namun, setelah metode mengeksekusi pernyataan kembali, metode akan keluar. Dalam contoh kita, jika hWorked lebih besar dari nol, program akan mengeksekusi pernyataan:

```
return staffPay;
```

dan keluar dari metode. Nilai kembali ini kemudian dapat diberikan ke variabel. Misalnya, jika hWorked adalah 10 dan hourlyRate adalah 20, kita dapat menggunakan pernyataan int pay = CalculatePay();

untuk menetapkan hasil CalculatePay() ke pembayaran variabel. Nilai pembayaran kemudian akan menjadi 200.

Di sisi lain, jika hWorked kurang dari atau sama dengan nol, program akan mengeksekusi pernyataan
return 0;

dan keluar dari metode. Nilai pembayaran akan menjadi 0.

kelebihan beban

Dalam C# (dan sebagian besar bahasa lainnya), Anda dapat membuat dua metode dengan nama yang sama asalkan memiliki tanda tangan yang berbeda. Ini dikenal sebagai kelebihan beban. Tanda tangan suatu metode mengacu pada nama metode dan parameter yang dimilikinya.

Tambahkan metode berikut di bawah metode CalculatePay() sebelumnya.

```
public int CalculatePay(int bonus, int allowance)
{
        PrintMessage();
        if (hWorked > 0)
                return hWorked * hourlyRate + bonus + allowance;
        else
                return 0;
}
```

Tanda tangan dari metode pertama adalah CalculatePay() sedangkan metode kedua adalah CalculatePay(int bonus, int allowance).

Metode kedua ini memiliki dua parameter - bonus dan tunjangan. Ini menghitung gaji karyawan dengan menambahkan nilai dari dua parameter ini ke produk hWorked dan hourlyRate. Dalam contoh ini, kita tidak menggunakan variabel lokal untuk menyimpan hasil hWorked * hourlyRate + bonus + tunjangan. Kita hanya mengembalikan hasil perhitungan secara langsung. Ini baik-baik saja. Kita akan belajar bagaimana menggunakan metode ini nanti.

Metode ToString()

Terakhir, sebelum kita melanjutkan ke bagian berikutnya, kita perlu menulis satu metode lagi – metode ToString(). Metode ToString() adalah metode khusus yang mengembalikan string yang mewakili kelas saat ini. Dalam C#, semua kelas dilengkapi dengan metode ToString() yang telah ditentukan sebelumnya. Namun, merupakan kebiasaan (dan diharapkan dari kita) untuk mengesampingkan metode ini. Mengganti metode berarti menulis metode versi kita sendiri.

Biasanya, metode ToString() yang kita tulis menampilkan nilai bidang dan properti kelas. Tambahkan kode berikut ke kelas Staf:

```
string timpa publik ToString()
  public override string ToString()
  {
        return "Name of Staff = " + nameOfStaff + ", hourlyRate = " +hourlyRate + ",
  hWorked = " + hWorked;
}
```

Seperti yang Anda lihat, metode ToString() mengembalikan tipe string. String yang dikembalikannya berisi informasi tentang kelas Staff. Kata kunci override dalam deklarasi metode menunjukkan bahwa metode ini menimpa metode default. Kita akan membahas lebih lanjut tentang kata kunci override di bab berikutnya.

7.6 KONSTRUKTOR

Sekarang, mari kita lihat konstruktor. Konstruktor adalah metode khusus yang digunakan untuk 'membangun' objek dari templat kelas. Ini adalah metode pertama yang dipanggil setiap kali kita membuat objek dari kelas kita. Konstruktor biasanya digunakan untuk menginisialisasi bidang kelas. Konstruktor selalu memiliki nama yang sama dengan kelas (Staf dalam kasus kita) dan tidak mengembalikan nilai apa pun. Kita tidak perlu menggunakan kata kunci void saat mendeklarasikan konstruktor.

Tambahkan baris berikut ke kelas Staf kita.

```
  public Staff(string name)
  {
        nameOfStaff = name; Console.WriteLine("\n" +
        nameOfStaff);
        Console.WriteLine("----------");
}
```

Dalam konstruktor ini, pertama-tama kita menginisialisasi bidang nameOfStaff dengan string yang diteruskan ke konstruktor (nama). Kita kemudian menampilkan nilai nameOfStaff di layar dan menggarisbawahinya dengan serangkaian tanda hubung. Seperti metode lainnya, kita dapat memiliki lebih dari satu konstruktor selama tanda tangannya berbeda. Kita dapat menambahkan konstruktor lain ke kelas kita.

```
  public Staff(string firstName, string lastName)
  {
        nameOfStaff = firstName + " " + lastName;
        Console.WriteLine("\n" + nameOfStaff); Console.WriteLine("-----");
}
```

Konstruktor ini memiliki dua parameter - firstName dan lastName. Baris pertama menggabungkan dua string dan menetapkan string yang dihasilkan ke nameOfStaff. Dua baris berikutnya mencetak nameOfStaff di layar dan menggarisbawahinya dengan serangkaian

tanda hubung. Mendeklarasikan konstruktor adalah opsional. Jika Anda tidak mendeklarasikan konstruktor Anda sendiri, C# membuatnya untuk Anda secara otomatis. Konstruktor default hanya menginisialisasi semua bidang di kelas ke nilai default, yang biasanya nol untuk bidang angka dan string kosong untuk bidang string.

7.7 MEMBUAT INSTANSI OBJEK

Sekarang kita tahu cara membuat kelas, mari kita lihat bagaimana kita bisa menggunakan kelas untuk membuat objek. Proses ini dikenal sebagai instantiating objek. Sebuah objek juga dikenal sebagai instance.

Untuk rekap, kelas Staf kita memiliki komponen berikut:

Fields

```
private const int hourlyRate
private string nameOfStaff
private int hWorked
```

Properti

```
public int HoursWorked
```

Metode

```
public void PrintMessage()
public int CalculatePay()
public int CalculatePay(int bonus, int allowance)
public override string ToString()
```

Konstruktor

```
public Staff(string name)
public Staff(string firstName, string lastName)
```

Kita akan membuat instance objek Staff dalam metode Main() di dalam kelas Program. Sintaks untuk membuat instance objek adalah:

```
ClassName objectName = new ClassName(arguments);
```

Tambahkan baris berikut di dalam kurung kurawal metode Main() di Kelas program.

```
int pay;
Staff staff1 = new Staff("Peter");
staff1.HoursWorked = 160;
pay = staff1.CalculatePay(1000, 400);
Console.WriteLine("Pay = {0}", pay);
```

Di sini, kita menggunakan konstruktor pertama (dengan satu parameter) untuk membuat objek staff1 kita. Setelah kita membuat objek, kita dapat menggunakan operator titik setelah nama objek untuk mengakses bidang publik, properti, atau metode apa pun di kelas Staf. Perhatikan bahwa kita perlu menggunakan operator titik di sini saat kita mencoba mengakses anggota kelas Staf dari kelas Program. Operator titik diperlukan setiap kali kita ingin mengakses bidang, properti, atau metode dari kelas lain.

Jika Anda mengakses anggota dari kelas yang sama, Anda tidak perlu menggunakan operator titik. Contohnya adalah ketika kita memanggil metode PrintMessage() dari metode HitungPay() tadi. Kita tidak menggunakan operator titik karena kedua metode berasal dari kelas yang sama. Setelah membuat objek staff1 kita, baris berikutnya menunjukkan bagaimana kita dapat menggunakan public Properti EmployeeType untuk menetapkan nilai ke bidang hWorked.

```
staff1.HoursWorked = 160;
```

Jika kita mencoba mengakses bidang pribadi hWorked secara langsung dengan menulis

```
staff1.hWorked = 160;
```

kita akan mendapatkan kesalahan karena hWorked adalah bidang pribadi dan oleh karena itu hanya dapat diakses di dalam kelas Staf. Untuk memanggil metode CalculatePay(), kita tulis:

```
staff1.CalculatePay(1000, 400);
```

Dalam contoh ini, karena kita memiliki angka 1000 dan 400 di dalam tanda kurung, kita menggunakan metode HitungPay() kedua. Kita meneruskan nilai 1000 dan 400 ke parameter bonus dan tunjangan masing-masing. Nilai-nilai yang kita lewati dikenal sebagai argumen. Program kemudian menggunakan metode tersebut untuk menghitung pembayaran dan mengembalikan jawabannya. Jawaban ini diberikan untuk pembayaran variabel. Terakhir, kita menggunakan metode Console.WriteLine() untuk menampilkan nilai pembayaran di layar.

Jika Anda menjalankan kode di atas, Anda akan mendapatkan:

```
Peter
------------------------------
Calculating Pay...Pay = 6200
```

Bermain-main dengan kode sedikit untuk mendapatkan perasaan yang lebih baik tentang bagaimana kelas bekerja. Coba tambahkan baris kode berikut:

```
Staff staff2 = new Staff("Jane", "Lee");
staff2.HoursWorked = 160;
pay = staff2.CalculatePay();
Console.WriteLine("Pay = {0}", pay);
```

Jika Anda menjalankan kode di atas, Anda akan mendapatkan:

```
Jane Lee
------------------------------
Calculating Pay...Pay = 4800
```

Terakhir, mari buat objek ketiga untuk mendemonstrasikan cara kerja validasi data saat kita menggunakan properti. Tambahkan baris kode berikut:

```
Staff staff3 = new Staff("Carol");
staff3.HoursWorked = -10;
pay = staff3.CalculatePay();
```

```
Console.WriteLine("Pay = {0}", pay);
```

Di sini, kita mencoba menyetel properti HoursWorked ke -10, yang merupakan nilai yang tidak valid. Penyetel properti itu menetapkan nilai ke nol sebagai gantinya. Jika Anda menjalankan kode ini, Anda akan mendapatkan:

```
Carol
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Calculating Pay...Pay = 0
```

7.8 KATA KUNCI STATIS

Kita telah membahas beberapa konsep yang cukup rumit dalam bab ini. Saya sangat menyarankan Anda mengunduh program lengkap untuk bab ini dari http://www.learncodingfast.com/csharp dan bermain-main dengannya. Pelajari kodenya dan pastikan Anda sepenuhnya memahami topik yang dibahas dalam bab ini sebelum melanjutkan. Di bagian ini, kita akan melihat kata kunci lain yang terkadang digunakan ketika kita mendeklarasikan kelas atau anggota kelas (yaitu metode, bidang, properti, konstruktor, dll). Sebelumnya, kita melihat bagaimana kita dapat menggunakan kelas Staff untuk membuat objek staff1, staff2 dan staff3. Namun, ada beberapa kelas atau anggota kelas yang dapat diakses tanpa perlu membuat objek apa pun. Ini dikenal sebagai kelas statis atau anggota kelas dan dideklarasikan menggunakan kata kunci statis.

Pertimbangkan kelas berikut:

```
1 class MyClass
2 {
3 //Non static members
4 public string message = "Hello World";
5 public string Name { get; set; }
6 public void DisplayName()
7 {
8 Console.WriteLine(“Name = {0}”, Name);
9 }
10
11 //Static members
12 public static string greetings = "Good morning";
13 public static int Age { get; set; }
14 public static void DisplayAge()
15 {
16 Console.WriteLine("Age = {0}", Age);
17 }
18 }
```

Kelasku berisi satu pesan bidang non-statis, satu Nama properti non-statis dan satu metode non-statis DisplayName() (baris 4 hingga 9). Ini juga berisi satu salam bidang statis, satu properti statis Age dan satu metode statis DisplayAge() (baris 12 hingga 17). Untuk mengakses anggota non-statis MyClass dari kelas lain, kita perlu membuat instance objek seperti sebelumnya:

```
MyClass classA = new MyClass();
Console.WriteLine(classA.message);
classA.Name = “Jamie”;
```

```
classA.DisplayName();
```

Namun, untuk mengakses anggota statis, kita tidak perlu membuat objek apa pun. Kita cukup menggunakan nama kelas untuk mengaksesnya seperti yang ditunjukkan di bawah ini.

```
Console.WriteLine(MyClass.greetings);
MyClass.Age = 39;
MyClass.DisplayAge();
```

Jika Anda menjalankan kode di atas, Anda akan mendapatkan output berikut:

```
Hello World
Name = Jamie
Good Morning
Age = 39
```

Selain memiliki metode, bidang, properti, dan konstruktor statis, kita juga dapat memiliki kelas statis. Kelas statis hanya dapat berisi anggota statis. Sebuah contoh ditunjukkan di bawah ini:

```
static class MyStaticClass
{
        public static int a = 0; public static
        int B{get; set;}
}
```

Beberapa kelas pra-tertulis dalam C# dideklarasikan sebagai kelas statis. Contohnya adalah kelas Console. Kita tidak perlu membuat objek Console saat menggunakan metode dari kelas Console. Kita cukup menulis Console.WriteLine("HelloWorld");.

Konsep Metode Tingkat Lanjut

Sekarang Anda sudah familiar dengan kelas, mari kita beralih ke beberapa konsep lanjutan mengenai deklarasi dan penggunaan metode di kelas. Konsep-konsep ini lebih kompleks dan mungkin memerlukan lebih dari satu bacaan untuk memahaminya sepenuhnya.

7.9 MENGGUNAKAN ARRAY DAN DAFTAR

Sebelumnya, kita telah mempelajari cara menggunakan tipe data dasar seperti int dan float sebagai parameter pada suatu metode. Selain menggunakan tipe data dasar, kita juga bisa menggunakan array dan list. Untuk menggunakan array sebagai parameter, kita menambahkan tanda kurung siku [] setelah tipe data parameter dalam deklarasi metode. Sebuah contoh ditunjukkan di bawah ini:

```
public void PrintFirstElement(int[] a)
{
        Console.WriteLine("The first element is {0}.\n", a[0]);
}
```

Untuk memanggil metode ini, kita perlu mendeklarasikan array dan meneruskannya sebagai argumen ke metode:

```
int[] myArray = {1, 2, 3, 4, 5};
PrintFirstElement(myArray);
```

Contoh berikutnya menunjukkan bagaimana kita dapat menggunakan daftar sebagai parameter.

```
public void PrintFirstListElement(List<int> a)
{
Console.WriteLine("The first list element is {0}.\n", a[0]);
}
```

Untuk memanggil metode, kita perlu mendeklarasikan daftar dan meneruskannya sebagai argumen ke metode.

```
List<int> myList = new List<int> {1, 2, 3};
PrintFirstListElement(myList);
```

Selain menggunakan larik atau daftar sebagai parameter ke suatu metode, kita juga dapat mengembalikan larik atau daftar dari suatu metode. Untuk mengembalikan array dari suatu metode, kita menambahkan tanda kurung siku [] setelah tipe kembalian dalam deklarasi metode.

```
public int[] ReturnUserInput()
{
        int[] a = new int[3];

        for (int i = 0; i < a.Length; i++)
        {
                Console.Write("Enter an integer: ");
                a[i] = Convert.ToInt32(Console.ReadLine()); Console.WriteLine("Integer added
                to array.\n");
}
         return a;
  }
```

Untuk menggunakan metode ini, kita perlu mendeklarasikan array dan menetapkan hasil metode ke dalamnya.

```
int[] myArray2 = ReturnUserInput ();
```

Untuk mengembalikan daftar dari suatu metode, kita menggunakan kata kunci List<> sebagai tipe pengembalian dalam deklarasi metode. Contohnya adalah:

```
public List<int> ReturnUserInputList()
{
```

```
        List<int> a = new List<int>();
        int input;

        for (int i = 0; i < 3; i++)
        {
                Console.Write("Enter an integer: ");
                input = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine("Integer added to list.\n");
                a.Add(input);
        }
        return a;
}
```

Untuk menggunakan metode ini, kita perlu mendeklarasikan daftar dan menetapkan hasil metode ke dalamnya.

```
List<int> myList2 = ReturnUserInputList();
```

7.10 MENGGUNAKAN KATA KUNCI PARAMS

Selanjutnya, mari kita jelajahi kata kunci params. Kata kunci params berguna ketika kita tidak mengetahui jumlah argumen yang dimiliki suatu metode. Misalnya, kita mungkin memiliki metode yang mencetak serangkaian nama, tetapi kita tidak tahu berapa banyak nama yang ada sebelumnya. Dalam kasus seperti ini, kita dapat menggunakan array sebagai parameter dan menambahkan kata kunci params di depannya.

Contohnya adalah:

```
public void PrintNames(params string[] names)
{
        for (int i = 0; i < names.Length; i++)
{
                Console.Write(names[i] + " ");
        }
        Console.WriteLine();
}
```

Untuk menggunakan metode ini, kita dapat memasukkan sejumlah string sebagai argumen. Contoh:

```
PrintNames("Peter"); PrintNames("Yvonne",
"Jamie");
PrintNames("Abigail", "Betty", "Carol", "David");
```

Output:

Peter
Yvonne
Jamie
Abigail Betty Carol David

Perhatikan bahwa tidak ada parameter tambahan yang diizinkan setelah kata kunci params dalam deklarasi metode, dan hanya satu kata kunci params yang diizinkan dalam deklarasi metode. Oleh karena itu, deklarasi metode berikut baik-baik saja:
public void PrintNames2(int a, double b, params int[] ages)

tetapi pernyataan berikut tidak:
public void PrintNames3(int a, params string[] names, double b)
public void PrintNames4(params string[] names, params int[] ages)

PrintNames3tidak diperbolehkan karena double b muncul setelah params string[] names
PrintNames4 tidak diperbolehkan karena ada dua kata kunci params.

7.11 MELEWATI JENIS NILAI VS PARAMETER JENIS REFERENSI

Saya harap Anda sekarang memiliki pemahaman yang baik tentang cara kerja kelas dan metode. Sebelum kita mengakhiri bab ini, saya ingin meninjau kembali konsep tipe data nilai dan tipe data referensi. Dalam Bab 4, kita mempelajari bahwa ada dua kategori utama tipe data dalam C# - tipe nilai dan tipe referensi. Ada perbedaan ketika Anda meneruskan variabel tipe nilai ke metode vs variabel tipe referensi.

Saat Anda memasukkan variabel tipe nilai, setiap perubahan yang dibuat pada nilai variabel itu hanya valid di dalam metode itu sendiri. Setelah program keluar dari metode, perubahan tidak lagi valid. Di sisi lain, jika Anda memasukkan variabel tipe referensi, perubahan apa pun yang dilakukan pada variabel akan valid bahkan setelah metode berakhir. Perhatikan kelas di bawah ini:

```
class MethodDemo
{
        public void PassByValue(int a)
        {
                a = 10;
                Console.WriteLine("a inside method = {0}", a);
        }
        public void PassByReference(int[] b)
        {
                b[0] = 5;
                Console.WriteLine("b[0] inside method = {0}", b[0]);
        }
}
```

Di dalam kelas, kita memiliki dua metode. Metode pertama menerima variabel tipe nilai dan mencoba mengubah nilai variabel itu. Ini kemudian mencetak nilai variabel. Metode kedua menerima array (tipe referensi) dan mencoba mengubah nilai elemen pertama dalam array. Kemudian mencetak nilai elemen itu. Dalam program Main() kita, misalkan kita memiliki baris kode berikut:

```
int a = 2;
int[] b = { 1, 2, 3 };
MethodDemo obj = new MethodDemo();
Console.WriteLine("a before = {0}", a);
obj.PassByValue(a);
Console.WriteLine("a after = {0}", a);
Console.WriteLine("\n\n");
Console.WriteLine("b[0] before = {0}", b[0]);
obj.PassByReference(b);
Console.WriteLine("b[0] after = {0}", b[0]);
```

Jika Anda menjalankan program, Anda akan mendapatkan:

```
a before = 2
a inside method = 10
a after = 2
b[0] before = 1
b[0] inside method = 5
b[0] after = 5
```

Nilai a tetap sama sebelum dan sesudah pemanggilan metode; perubahan hanya valid di dalam metode itu sendiri. Di sisi lain, nilai b[0] berubah setelah pemanggilan metode. Waspadai perbedaan ini saat Anda meneruskan variabel tipe nilai ke metode (mis. int, float, dll.) vs variabel tipe referensi (seperti array atau daftar).

BAB 8
PEMROGRAMAN BERORIENTASI OBJEK BAGIAN 2

Sekarang, mari kita beralih ke beberapa topik lanjutan dalam pemrograman berorientasi objek. Dalam bab ini, kita akan belajar tentang pewarisan, polimorfisme, kelas abstrak, dan antarmuka.

8.1 WARISAN

Warisan adalah salah satu konsep kunci dari pemrograman berorientasi objek. Secara sederhana, pewarisan memungkinkan kita untuk membuat kelas baru dari kelas yang ada sehingga kita dapat secara efektif menggunakan kembali kode yang ada.

Menulis Kelas Orang Tua

Misalkan kita sedang menulis program untuk klub kebugaran yang memiliki dua jenis keanggotaan – VIP dan Normal. Untuk melakukan itu, mari buat kelas bernama Member terlebih dahulu.

```
class Member
{
        protected int annualFee;
        private string name; private
        int memberID; private int
        memberSince;
}
```

Anggota berisi satu bidang yang dilindungi dan tiga bidang pribadi. Bidang yang dilindungi adalah bidang yang hanya dapat diakses di dalam kelas di mana ia dideklarasikan dan setiap kelas yang diturunkan darinya. Kita akan segera berbicara tentang kelas turunan. Selanjutnya, mari kita tulis metode ToString() untuk menampilkan nilai dari empat bidang.

```
  public override string ToString()
  {
        return "\nName: " + name + "\nMember ID: " + memberID +"\nMember
  Since: " + memberSince + "\nTotal Annual Fee: " +annualFee;
}
```

Terakhir, mari tambahkan dua konstruktor ke kelas Anggota.

```
public Member()
{
        Console.WriteLine("Parent Constructor with no parameter");
}

  public Member(string pName, int pMemberID, int pMemberSince)
  {
```

```
        Console.WriteLine("Parent Constructor with 3 parameters");

        name = pName;
        memberID =
        pMemberID;
        memberSince = pMemberSince;
}
```

Konstruktor pertama hanya mencetak baris "Konstruktor Induk tanpa parameter". Konstruktor kedua lebih menarik. Ini mencetak baris "Konstruktor Induk dengan 3 parameter" dan menetapkan parameternya ke tiga bidang pribadi di kelas Anggota.

Menulis Kelas Anak

Sekarang, mari kita pelajari cara menurunkan kelas dari kelas Anggota. Kelas turunan dikenal sebagai kelas anak, sedangkan kelas dari mana mereka diturunkan dikenal sebagai kelas induk atau kelas dasar. Kelas turunan mewarisi semua anggota publik dan yang dilindungi dari kelas induk. Dengan kata lain, ia dapat menggunakan bidang, properti, dan metode tersebut seolah-olah itu adalah bagian dari kodenya sendiri.

Kelas induk kita (Anggota) memiliki konten berikut:

Fields

```
protected int annualFee
private string name
private int memberID
private int memberSince
```

Metode

```
public override string ToString()
```

Konstruktor

```
public Member()
public Member(string pName, int pMemberID, int pMemberSince)
```

Kita akan menurunkan dua kelas – NormalMember dan VIPMember – dari Anggota kelas. Pertama, mari kita deklarasikan kelas anak NormalMember. Kita menunjukkan bahwa itu berasal dari kelas Anggota menggunakan titik dua (:) seperti ini:

```
class NormalMember : Member
  {
}
```

Sekarang, kita perlu menulis konstruktor untuk kelas anak. Konstruktor kelas anak dibangun di atas konstruktor induk. Setiap kali kita membuat objek anak, konstruktor kelas induk selalu dipanggil terlebih dahulu. Ada dua cara untuk membuat konstruktor anak. Cara pertama adalah dengan mendeklarasikannya seperti konstruktor lainnya.

```
public NormalMember()
  {
```

```
        Console.WriteLine("Child constructor with no parameter");
}
```

Ketika kita mendeklarasikan konstruktor kita seperti di atas, C# mencari konstruktor tanpa parameter (yaitu konstruktor tanpa parameter) di kelas induk dan memanggilnya terlebih dahulu sebelum mengeksekusi kode di konstruktor anak. Jika Anda menggunakan konstruktor ini untuk membuat objek anak, dua baris berikut akan ditampilkan di layar:

Parent Constructor with no parameter
Child constructor with no parameter

Baris pertama dari konstruktor induk sedangkan baris kedua dari konstruktor anak.

Cara kedua untuk mendeklarasikan konstruktor anak adalah dengan menggunakan tanda titik dua (:) dan kata kunci dasar untuk memanggil konstruktor non-parameter di kelas induk. Sebuah contoh ditunjukkan di bawah ini:

```
public NormalMember(string remarks) : base (“Jamie”, 1, 2015)
  {
        Console.WriteLine("Remarks = {0}", remarks);
}
```

Saat kita memanggil konstruktor tanpa parameter di kelas induk, kita perlu meneruskan nilai ke parameternya. Pada contoh di atas, kita meneruskan nilai “Jamie”, 1 dan 2015 ke konstruktor induk. Nilai-nilai ini kemudian ditetapkan ke nama bidang, ID anggota dan anggota Sejak di kelas dasar masing-masing. Dalam contoh ini, kita meneruskan nilai tetap sebagai argumen ke konstruktor dasar. Namun, cara yang lebih baik adalah dengan meneruskan argumen melalui konstruktor anak. Contoh di bawah ini menunjukkan bagaimana hal ini dapat dilakukan. Ganti konstruktor sebelumnya dengan konstruktor di bawah ini:

```
  public NormalMember(string remarks, string name, int memberID, intmemberSince) :
  base (name, memberID, memberSince)
  {
        Console.WriteLine("Child Constructor with 4 parameters");
        Console.WriteLine("Remarks = {0}", remarks);
                                                    }
```

Konstruktor anak baru ini memiliki empat parameter. Parameter pertama adalah string parameter yang disebut komentar. Parameter ini digunakan di dalam konstruktor anak. Parameter kedua, ketiga dan keempat tidak digunakan dalam konstruktor anak. Sebaliknya, mereka diteruskan sebagai argumen ke konstruktor induk berdasarkan nama mereka. Misalnya, parameter kedua dalam konstruktor anak (nama string) diteruskan sebagai argumen pertama ke konstruktor induk (nama). Saat kita membuat objek anak dengan konstruktor ini, kita menulis sesuatu seperti:

```
NormalMember myChildMember = new NormalMember(“Special Rate”,"James", 1, 2010);
```

Konstruktor dasar dengan 3 parameter dipanggil dan dieksekusi terlebih dahulu. Nilai "James", 1 dan 2010 diteruskan ke konstruktor dasar. Di belakang layar, nilai-nilai ini ditetapkan ke nama bidang, ID anggota dan anggota Sejak di kelas dasar masing-masing.

Setelah menjalankan konstruktor dasar, konstruktor anak akan dieksekusi. String "Tarif Khusus" ditetapkan ke komentar dan ditampilkan di layar. Saat Anda menjalankan kode, Anda akan mendapatkan output berikut:

```
Parent Constructor with 3 parameters
Child Constructor with 4 parameters
Remarks = Special Rate
```

Sekarang kita telah membuat konstruktor untuk kelas anak kita, mari kita lanjutkan untuk membuat metode untuk menghitung biaya tahunan anggota normal. Kode sederhananya:

```
public void CalculateAnnualFee()
{
        annualFee = 100 + 12*30;
}
```

Ketika kita menulis “annualFee” pada kode di atas, kita mengakses bidang yang dilindungi annualFee di kelas induk. Ingat bahwa kelas anak memiliki akses ke semua bidang publik dan yang dilindungi di kelas induknya? Oleh karena itu, kelas anak dapat menggunakan bidang ini seolah-olah itu adalah bidangnya sendiri. Kelas anak tidak perlu membuat turunan dari kelas induk untuk mengakses bidang yang dilindungi.

Itu saja untuk kelas anak kita NormalMember. Kelas memiliki konten berikut:

Fields

Diwarisi dari kelas induk:

```
protected int annualFee
```

Metode

Diwarisi dari kelas induk:

```
public override string ToString()
```

Dideklarasikan di kelas anak:

```
public void CalculateAnnualFee()
```

Konstruktor

```
public NormalMember()
public NormalMember(string remarks, string name, int memberID, int
```

Selanjutnya, mari kita tulis kelas lain yang mewarisi dari Anggota. Kali ini, kelas turunan disebut VIPMember. Kode ditunjukkan di bawah ini:

```
class VIPMember : Member
{
        public VIPMember(string name, int memberID, int memberSince) :base (name,
memberID, memberSince)
```

```
        {
            Console.WriteLine("Child Constructor with 3 parameters");
        }

        public void CalculateAnnualFee()
        {
            annualFee = 1200;
        }
}
```

Kelas ini memiliki satu konstruktor (dengan 3 parameter) dan satu metode HitungAnnualFee(). Metode HitungAnnualFee() di sini menggunakan rumus yang berbeda untuk menghitung biaya tahunan dari metode HitungAnnualFee() di kelas NormalMember. Tidak apa-apa untuk dua metode untuk berbagi nama yang sama (dan tanda tangan) karena mereka berada di kelas yang berbeda.

Kelas VIPMember memiliki konten sebagai berikut:

Fields

Diwarisi dari kelas induk:

protected int annualFee

Metode

Diwarisi dari kelas induk:

public override string ToString()

Dideklarasikan di kelas anak:

public void CalculateAnnualFee()

Konstruktor

public VIPMember(string name, int memberID, int memberSince)

8.2 METODE UTAMA()

Sekarang kita telah menulis tiga kelas yang kita butuhkan, mari kita tulis kode untuk metode Main(). Pertama, kita akan membuat dua objek dari dua kelas turunan.

```
NormalMember mem1 = new NormalMember("Special Rate", "James", 1,2010);
VIPMember mem2 = new VIPMember("Andy", 2, 2011);
```

mem1 dibuat menggunakan konstruktor 4 parameter dari kelas NormalMember.

Mem2 dibuat menggunakan konstruktor 3 parameter dari kelas VIPMember. Selanjutnya, kita akan menggunakan metode CalculateAnnualFee() di masing-masing kelas.

```
mem1.CalculateAnnualFee();
mem2.CalculateAnnualFee();
```

Karena mem1 adalah turunan dari kelas NormalMember CalculateAnnualFee(), metode dari kelas itu dieksekusi. Biaya tahunan untuk mem1 adalah 100 + 12*30 = 460. Untuk mem2, biaya tahunannya adalah 1200 karena menggunakan metode dari VIPMember kelas.

Terakhir, mari kita gunakan metode ToString() dari kelas induk (Anggota) untuk menampilkan informasi di layar kita. Kita menulis:

```
Console.WriteLine(mem1.ToString()); Console.WriteLine(mem2.ToString());
```

Karena metode ToString() milik kelas induk dan bersifat publik, baik mem1 dan mem2 mewarisi metode tersebut dan dengan demikian dapat menggunakannya dalam metode Main(). Ini memfasilitasi penggunaan kembali kode karena kita tidak perlu menulis ulang metode ToString() untuk kedua kelas anak.

Anda akan mendapatkan output berikut saat menjalankan program:

```
Parent Constructor with 3 parameters
Child Constructor with 4 parameters
Message = Special Rate
Parent Constructor with 3 parameters
Child Constructor with 3 parameters

Name:James
Member ID:
1

Member Since: 2010
Total Annual Fee: 460
Name: Andy
Member ID:
2
Member Since: 2011
Total Annual Fee: 1200
```

8.3 POLIMORFISME

Sekarang setelah kita melihat contoh bagaimana pewarisan bekerja, mari kita lanjutkan untuk membahas topik lain yang terkait erat dengan pewarisan - konsep polimorfisme. Polimorfisme mengacu pada kemampuan program untuk menggunakan metode yang benar untuk suatu objek berdasarkan jenis runtime-nya. Cara terbaik untuk menjelaskan polimorfisme adalah melalui sebuah contoh. Mari kita kembangkan contoh klub kebugaran kita di atas.

Pertama, hapus semua kode di metode Main() sebelumnya dan tambahkan baris berikut:

```
Member[] clubMembers = new Member[5];

clubMembers[0] = new NormalMember("Special Rate", "James", 1,2010);
clubMembers[1] = new NormalMember("Normal Rate", "Andy", 2, 2011);
```

```
clubMembers[2] = new NormalMember("Normal Rate", "Bill", 3, 2011);
clubMembers[3] = new VIPMember("Carol", 4, 2012);
clubMembers[4] = new VIPMember("Evelyn", 5, 2012);
```

Di sini, kita mendeklarasikan array tipe Anggota dan menambahkan 5 anggota ke dalamnya. Tiga anggota pertama adalah instance dari kelas NormalMember sedangkan dua yang terakhir adalah instance dari kelas VIPMember. Meskipun clubMembers dideklarasikan sebagai sebuah array dari tipe Member, kita dapat menetapkan instance dari NormalMember dan VIPMember karena mereka adalah kelas anak dari kelas Member. Kita tidak perlu mendeklarasikan array terpisah untuk objek NormalMember dan VIPMember. Selanjutnya, kita akan menggunakan loop foreach untuk menghitung biaya tahunan setiap anggota dan menampilkan informasi.
Untuk melakukan itu, kita menulis:

```
foreach (Member m in clubMembers)
  {
        m.CalculateAnnualFee();
        Console.WriteLine(m.ToString());
}
```

Jika Anda mencoba menjalankan program pada tahap ini, Anda akan mendapatkan pesan kesalahan yang mengatakan Anggota tidak mengandung definisi untuk 'CalculateAnnualFee'. Ini karena clubMembers dideklarasikan sebagai array dari tipe Member. Oleh karena itu, kompiler mencoba mengeksekusi metode CalculateAnnualFee(di kelas Anggota saat kita menulis m.CalculateAnnualFee(). Terjadi kesalahan karena kita tidak memiliki metode seperti itu di kelas induk Anggota kita; kita hanya memilikinya di dua kelas anak. Untuk memperbaiki kesalahan ini, kita harus menambahkan metode berikut ke kelas induk kita.

```
public void CalculateAnnualFee()
{
annualFee = 0;
}
```

Sekarang jalankan program dan perhatikan “Total Biaya Tahunan” untuk setiap anggota. Apa yang Anda perhatikan? Itu semua harus menunjukkan $0. Ini berarti metode HitungAnnualFee() yang dipanggil adalah yang ada di kelas induk. Ini tidak mengherankan karena clubMembers dinyatakan sebagai tipe Member. Jika Anda ingin metode anak dipanggil, Anda harus membuat dua perubahan. Pertama, Anda perlu mendeklarasikan metode induk sebagai virtual, seperti ini:

```
public virtual void CalculateAnnualFee()
  {
        annualFee = 0;
}
```

Kata kunci virtual memberi tahu kompiler bahwa metode ini dapat diganti di kelas turunan. Ketika kompiler menemukan kata kunci ini, ia akan mencari metode yang sama di kelas turunan dan menjalankan metode itu sebagai gantinya. Selanjutnya, di kelas turunan, Anda harus mendeklarasikan bahwa metode Anda menimpa metode di kelas induk menggunakan kata kunci override, seperti ini:

```
  //In VIPMember child class
  public override void CalculateAnnualFee()
  {
        annualFee = 1200;
  }

  //In NormalMember child class
  public override void CalculateAnnualFee()
  {
        annualFee = 100 + 12 * 30;
}
```

Sekarang jika Anda menjalankan program lagi, biaya tahunan untuk tiga anggota pertama (NormalMember) dan dua anggota terakhir (VIPMember) masing-masing adalah $460 dan $1200.

Ini adalah hasil dari polimorfisme. Pada saat run time (yaitu ketika program berjalan), program menentukan bahwa tiga anggota pertama dari clubMembers bertipe NormalMember dan mengeksekusi metode HitungAnnualFee() dari kelas tersebut. Itu juga menentukan bahwa dua anggota terakhir adalah tipe VIPMember dan mengeksekusi metode dari kelas itu.

Polimorfisme secara sederhana berarti bahwa pada waktu berjalan, program cukup pintar untuk menggunakan metode HitungAnnualFee() dari kelas anak yang benar bahkan ketika objek tersebut dinyatakan sebagai tipe Anggota. Kita mengatakan bahwa tipe runtime dari tiga elemen pertama clubMembers adalah NormalMember sedangkan tipe runtime dari dua elemen terakhir adalah VIPMember. Jenis yang dideklarasikan dari semua 5 elemen adalah Anggota.

8.4 GETTYPE() DAN TYPEOF()

Pada contoh sebelumnya, kita membiarkan program menentukan tipe waktu proses dari setiap anggota larik clubMembers dan memanggil metode HitungAnnualFee() yang benar. Namun, terkadang, mungkin kita perlu menentukan sendiri jenis runtime dari setiap anggota individu saat kita membuat kode. Kita akan melihat contohnya nanti di proyek kita. Pernyataan if di bawah ini menunjukkan bagaimana Anda dapat menentukan apakah elemen pertama dari larik clubMember bertipe VIPMember saat dijalankan:

```
  if (clubMembers[0].GetType() == typeof(VIPMember))
        Console.WriteLine("Yes");
  else
```

```
Console.WriteLine(“No”);
```

Metode GetType() mengembalikan tipe runtime dari suatu objek.

Metode typeof() mengambil nama tipe data (misalnya int, float, atau nama kelas) dan mengembalikan tipe nama tersebut, yang kemudian dapat kita bandingkan dengan hasil metode GetType() di sebelah kiri . Jika Anda menjalankan kode di atas, Anda akan mendapatkan "Tidak" sebagai output karena clubMembers[0] bukan tipe VIPMember.

8.5 KELAS DAN METODE ABSTRAK

Sekarang kita sudah familiar dengan pewarisan (dan polimorfisme), mari kita lanjutkan untuk membahas dua tipe khusus "kelas induk" di C# - kelas abstrak dan antarmuka. Pertama, mari kita lihat kelas abstrak.

Kelas abstrak adalah jenis kelas khusus yang dibuat secara ketat untuk menjadi kelas dasar bagi kelas lain untuk diturunkan. Mereka tidak dapat diinstansiasi. Dengan kata lain, jika FourWheelVehicles adalah kelas abstrak, pernyataan FourWheelVehicle myVeh = new FourWheelVehicle(); akan memberi Anda kesalahan karena Anda tidak dapat membuat objek dari kelas abstrak.

Kelas abstrak mungkin memiliki bidang, properti, dan metode seperti kelas lainnya. Namun, mereka tidak dapat memiliki anggota statis. Selain itu, kelas abstrak dapat memiliki tipe metode khusus yang dikenal sebagai metode abstrak. Metode abstrak adalah metode yang tidak memiliki tubuh dan HARUS diimplementasikan di kelas turunan. Mereka hanya bisa ada di kelas abstrak. Di satu sisi, metode abstrak seperti kontrak. Jika Anda ingin memastikan bahwa setiap kelas yang mewarisi kelas Anda mengimplementasikan metode tertentu, Anda dapat mendeklarasikan kelas tersebut sebagai kelas abstrak dan metode tersebut sebagai metode abstrak.

Untuk mendeklarasikan kelas abstrak, cukup tambahkan kata kunci abstract sebelum kelas kata kunci seperti ini:

```
abstract class MyClass
{
}
```

Untuk mendeklarasikan metode abstrak di dalam kelas abstrak, tambahkan kata kunci abstract sebelum tipe kembalian, seperti ini:

```
public abstract void MyAbstractMethod();
```

Karena metode abstrak tidak memiliki badan, kita mengakhiri deklarasi dengan titik koma (;). Untuk mengimplementasikan metode abstrak di kelas turunan, kita menggunakan kata kunci override, seperti ini.

```
public override void MyAbstractMethod()
{
}
```

Kode di bawah ini menunjukkan contoh kelas abstrak.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace AbstractClassDemo
8 {
9 class Program
10 {
11 static void Main(string[] args)
12 {
13 //MyAbstractClass abClass = new MyAbstractClass();
14 ClassA a = new ClassA();
15 a.PrintMessage();
16 a.PrintMessageAbstract();
17 Console.Read();
18 }
19 }
20
21 abstract class MyAbstractClass
22 {
23 private string message = "Hello C#";
24 public void PrintMessage()
25 {
26 Console.WriteLine(message);
27 }
28 public abstract void PrintMessageAbstract();
29 }
30
31 class ClassA : MyAbstractClass
32 {
33 public override void PrintMessageAbstract()
34 {
35 Console.WriteLine("C# is fun!");
36 }
37 }
38 }
```

Kelas abstrak adalah dari Baris 21 hingga 29. Ini berisi pesan bidang pribadi dan metode publik PrintMessage(). Ini juga berisi metode abstrak PrintMessageAbstract() pada baris 28. Baris 31 hingga 37 menunjukkan kelas turunan yang mengimplementasikan metode abstrak (baris 33 hingga 36). Jika Anda menjalankan program di atas, Anda akan mendapatkan:
Hello C#
C# is fun!

Perhatikan bahwa Baris 13 dikomentari dengan tanda //? Jika Anda menghapus dua garis miring, Anda akan mendapatkan kesalahan karena kelas abstrak tidak dapat dipakai.

8.6 ANTARMUKA

Selanjutnya, mari kita lihat antarmuka. Antarmuka sangat mirip dengan kelas abstrak karena tidak dapat dipakai dan harus diwariskan. Namun, antarmuka lebih konseptual daripada kelas abstrak. Mereka hanya dapat berisi metode tanpa badan. Selain itu, mereka tidak dapat berisi bidang tetapi dapat berisi properti. Antarmuka juga tidak dapat memiliki anggota statis. Ketika kelas anak mewarisi antarmuka, kita mengatakan bahwa itu mengimplementasikan antarmuka. Salah satu perbedaan utama antara kelas abstrak dan

antarmuka adalah bahwa kelas hanya dapat mewarisi satu kelas abstrak tetapi dapat mengimplementasikan beberapa antarmuka. Kita tidak akan menunjukkan contoh implementasi beberapa antarmuka karena itu adalah topik lanjutan di luar cakupan buku ini. Kode di bawah ini menunjukkan contoh bagaimana sebuah kelas dapat mengimplementasikan satu antarmuka. Nama antarmuka biasanya diawali dengan huruf I. Semua properti dan metode dalam antarmuka bersifat publik, jadi tidak perlu menambahkan pengubah akses apa pun ke dalamnya.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace InterfaceDemo
8 {
9 class Program
10 {
11 static void Main(string[] args)
12 {
13 ClassA a = new ClassA();
14 a.MyNumber = 5;
15 a.InterfaceMethod();
16 Console.Read();
17 }
18 }
19
20 interface IShape
21 {
22 int MyNumber
23 {
24 get;
25 set;
26 }
27 void InterfaceMethod();
28 }
29
30 class ClassA : IShape
31 {
32
33 private int myNumber;
34 public int MyNumber
35 {
36 get
37 {
38 return myNumber;
39 }
40 set
41 {
42 if (value < 0)
43 myNumber = 0;
```

```
44 else
45 myNumber = value;
46 }
47 }
48
49 public void InterfaceMethod()
50 {
51 Console.WriteLine("The number is {0}.", MyNumber);
52 }
53 }
54 }
```

Antarmuka dideklarasikan pada baris 20 hingga 28. Pada baris 22 hingga 26, kita mendeklarasikan properti dan pada baris 27, kita mendeklarasikan metode. ClassA mengimplementasikan antarmuka IShape. Properti diimplementasikan pada baris 33 hingga 47 di mana kita mendeklarasikan bidang pendukung pribadi (myNumber) untuk properti dan menerapkan beberapa aturan kontrol.

Metode ini diimplementasikan pada baris 49 hingga 52. Kita tidak perlu menggunakan kata kunci override saat mengimplementasikan metode yang dimiliki oleh sebuah antarmuka. Jika Anda menjalankan program ini, Anda akan mendapatkan:
The number is 5.

8.7 PENGUBAH AKSES DITINJAU KEMBALI

Sekarang kita telah membahas berbagai topik yang berkaitan dengan pewarisan, mari kita lihat lagi konsep pengubah akses dalam pemrograman berorientasi objek. Sebelumnya, kita telah mempelajari bahwa pengubah akses seperti penjaga gerbang. Ini mengontrol siapa yang memiliki akses ke bidang, properti, atau metode tertentu. C# hadir dengan 4 pengubah akses: private, public, protected dan internal. Apa pun yang dideklarasikan sebagai internal hanya dapat diakses di dalam Majelis saat ini. Karena kita tidak akan membahas rakitan dalam buku ini, kita tidak akan mendemonstrasikan cara kerja internal. Untuk memahami cara kerja pribadi, publik, dan dilindungi, mari pertimbangkan contoh di bawah ini. Kita akan menggunakan bidang untuk mendemonstrasikan konsep tersebut. Hal yang sama berlaku untuk metode dan properti.
Misalkan kita memiliki kelas dengan tiga bidang:

```
class ClassA
{
        private int privateNum = 1; public int
        publicNum = 2; protected int
        protectedNum = 3;
}
```

Jika KelasB diturunkan dari ClassA,

```
class ClassB:ClassA
  {
```

```
        public void PrintMessages()
        {
                //This is ok
                Console.WriteLine(publicNum);

            //This is ok Console.WriteLine(protectedNum);

            //This is NOT ok
                Console.WriteLine(privateNum);
        }
}
```

dua pernyataan WriteLine() pertama tidak akan memberi kita kesalahan apa pun karena kelas turunan dapat mengakses bidang publik dan yang dilindungi di kelas induk.

Namun, pernyataan ketiga memberi kita kesalahan karena privateNum adalah bidang pribadi dan dengan demikian hanya dapat diakses di ClassA itu sendiri.

Jika sebuah kelas tidak diturunkan dari ClassA, kita perlu membuat instance objek ClassA untuk mengakses bidang publik ClassA. Namun, bahkan dengan objek ClassA, kita tidak dapat mengakses bidang pribadi dan terproteksi dari ClassA. Pada contoh di bawah, ClassC tidak diturunkan dari ClassA. Oleh karena itu, pernyataan WriteLine() pertama tidak akan memberi kita kesalahan apa pun, tetapi pernyataan kedua dan ketiga akan memberikan kesalahan.

```
class ClassC
{
ClassA a = new ClassA();
public void PrintMessages()
        {
                //This is ok
                Console.WriteLine(a.publicNum);

            //This is NOT ok Console.WriteLine(a.protectedNum);

                //This is NOT ok Console.WriteLine(a.privateNum);
        }
  }
```

Singkatnya, apa pun yang dinyatakan sebagai publik dapat diakses di mana-mana; tidak ada batasan untuk mengakses anggota publik. Di sisi lain, apa pun yang dideklarasikan sebagai pribadi hanya dapat diakses di dalam kelas di mana ia dideklarasikan. Apa pun yang dideklarasikan sebagai dilindungi dapat diakses di dalam kelas di mana ia dideklarasikan dan setiap kelas yang diturunkan darinya.

BAB 9
E-NUM DAN STRUKTUR

Dalam Bab 3 dan 4, kita melihat beberapa tipe data bawaan yang disediakan oleh C#. Ini termasuk tipe nilai seperti int, float dan double dan tipe data referensi seperti array, string, dan daftar. Selain itu, kita juga melihat bagaimana Anda dapat menulis kelas Anda sendiri di Bab 7 dan 8. Kelas dapat dianggap sebagai tipe data yang ditentukan pengguna tingkat lanjut yang mengelompokkan sekumpulan bidang, properti, dan metode terkait ke dalam unit logis. Dalam bab ini, kita akan melihat dua tipe data yang ditentukan pengguna di C# – enum dan struct.

9.1 E-NUM

E-num (yang merupakan singkatan dari tipe enumerated) adalah tipe data khusus yang memungkinkan pemrogram untuk memberikan nama yang bermakna untuk satu set konstanta integral. Untuk mendeklarasikan enum, kita menggunakan kata kunci enum diikuti dengan nama enum. Anggota enum diapit oleh kurung kurawal dan dipisahkan dengan koma. Contoh ditunjukkan di bawah ini:

```
enum DaysOfWeek
{
    Sun, Mon, Tues, Wed, Thurs, Fri, Sat
}
```

Perhatikan bahwa kita tidak menempatkan titik koma di akhir anggota terakhir.

Setelah mendeklarasikan enum DaysOfWeek, kita dapat mendeklarasikan dan menginisialisasi variabel DaysOfWeek seperti ini:

```
DaysOfWeek myDays = DaysOfWeek.Mon;
```

Nama variabelnya adalah myDays. Jika kita menulis:

```
Console.WriteLine(myDays);
```

kita akan mendapatkan:

```
Mon
```

Secara default, setiap anggota dalam enum diberi nilai integer, mulai dari nol. Artinya, dalam contoh kita, Matahari diberi nilai 0, Sen adalah 1, Sel adalah 2 dan seterusnya. Karena anggota enum pada dasarnya adalah bilangan bulat, kita dapat memasukkan variabel DaysOfWeek ke dalam int dan sebaliknya. Misalnya, Console.WriteLine((int)myDays); memberi kita bilangan bulat 1 while Console.WriteLine((DaysOfWeek)1);memberi kita Mon.

Jika Anda ingin menetapkan kumpulan bilangan bulat yang berbeda ke anggota enum Anda, Anda dapat lakukan hal berikut:

```
enum DaysOfWeekTwo
  {
        Sun = 5, Mon = 10, Tues, Wed, Thurs, Fri, Sat
}
```

Sekarang, Matahari diberi nilai 5 dan Sen diberi nilai 10. Karena kita tidak menetapkan nilai untuk Sel hingga Sab, angka berurutan setelah 10 akan diberikan padanya.
Yaitu Sel = 11, Rab = 12 dan seterusnya.

Semua enum disimpan secara internal sebagai bilangan bulat (int). Jika Anda ingin mengubah tipe data yang mendasari dari int ke tipe data lain, Anda menambahkan titik dua setelah nama enum, diikuti dengan tipe data yang diinginkan. Semua tipe data integer diperbolehkan kecuali char. Contohnya adalah:

```
  enum DaysOfWeekThree : byte
  {
        Sun, Mon, Tues, Wed, Thurs, Fri, Sat
  }
```

Tentu saja, jika Anda menggunakan tipe data byte, Anda tidak dapat melakukan sesuatu seperti

```
  enum DaysOfWeekFour : byte
  {
        Sun = 300, Mon, Tues, Wed, Thurs, Fri, Sat
}
```

karena rentang byte adalah dari 0 hingga 255.

Ada dua alasan utama untuk menggunakan enum. Yang pertama adalah meningkatkan keterbacaan kode Anda. Pernyataan:

```
myDays = DaysOfWeek.Mon;
```

lebih jelas daripada pernyataan:

```
myDays = 1;
```

Alasan kedua adalah untuk membatasi nilai yang dapat diambil oleh variabel. Jika kita memiliki variabel yang menyimpan hari dalam seminggu, kita mungkin secara tidak sengaja menetapkan nilai 10 padanya. Ini dapat dicegah ketika kita menggunakan enum karena kita hanya dapat menetapkan anggota enum yang telah ditentukan sebelumnya ke variabel.

9.2 STRUKTUR

Sebuah struct mirip dengan kelas dalam banyak aspek. Seperti kelas, mereka mengandung elemen seperti properti, konstruktor, metode dan bidang dan memungkinkan

Anda untuk mengelompokkan anggota terkait ke dalam satu paket sehingga Anda dapat memanipulasi mereka sebagai sebuah grup. Untuk mendeklarasikan struct, Anda menggunakan kata kunci struct. Contohnya adalah:

```
1 struct MyStruct
2 {
3 //Fields
4 private int x, y;
5 private AnotherClass myClass;
6 private Days myDays;
7
8 //Constructor
9 public MyStruct(int a, int b, int c)
10 {
11 myClass = new AnotherClass();
12 myClass.number = a;
13 x = b;
14 y = c;
15 myDays = Days.Mon;
16 }
17
18 //Method
19 public void PrintStatement()
20 {
21 Console.WriteLine("x = {0}, y = {1}, myDays = {2}", x, y,
myDays);
22 }
23 }
24
25 class AnotherClass
26 {
27 public int number;
28 }
29
30 enum Days { Mon, Tues, Wed }
```

struct dideklarasikan dari baris 1 hingga 23. Pada baris 4, kita mendeklarasikan dua bidang int pribadi untuk struct. Pada baris 5, kita mendeklarasikan bidang pribadi lain yang disebut myClass. Bidang ini adalah turunan dari kelas AnotherClass. Pada baris 6, kita mendeklarasikan variabel enum myDays. Kedua bidang (myClass dan myDays) secara khusus disertakan dalam contoh ini untuk menunjukkan bagaimana kita dapat menyertakan instance kelas dan variabel enum sebagai bidang struct. Struct (dan kelas) dapat berisi variabel enum dan instance dari struct dan kelas lain sebagai bidang.

Setelah mendeklarasikan bidang, kita mendeklarasikan konstruktor untuk struct (baris 9 hingga 16), diikuti dengan metode untuk mencetak nilai x, y, dan myDays. (baris 19 sampai 22). Setelah mendeklarasikan struct, kita mendeklarasikan kelas AnotherClass pada baris 25 hingga 28 dan enum Days pada baris 30. Dalam contoh ini, kita mendeklarasikan kelas dan enum di luar struct myStruct. Namun, dimungkinkan bagi kita untuk mendeklarasikan enum atau kelas di dalam struct itu sendiri. Enum, struct atau kelas dapat bersarang di dalam struct atau kelas lain. Kita akan melihat contoh enum yang dideklarasikan di dalam kelas saat kita

mengerjakan proyek di akhir buku. Untuk menggunakan struct di atas, kita dapat menambahkan kode berikut ke metode Main() kita:

```
MyStruct example = new MyStruct(2, 3, 5);
example.PrintStatement();
```

Jika kita menjalankan kodenya, kita akan mendapatkan

```
x = 3, y = 5, myDays = Mon
```

Ada dua perbedaan utama antara struct dan kelas. Pertama, tipe data struct tidak mendukung pewarisan. Karenanya Anda tidak dapat menurunkan satu struct dari yang lain. Namun, sebuah struct dapat mengimplementasikan sebuah antarmuka. Cara melakukannya adalah identik dengan bagaimana hal itu dilakukan dengan kelas. Lihat Bab 8 untuk informasi lebih lanjut. Perbedaan kedua antara struct dan kelas adalah bahwa struct adalah tipe nilai sedangkan kelas adalah tipe referensi. Untuk daftar lengkap perbedaan antara struct dan kelas, lihat halaman berikut: https://msdn.microsoft.com/en-us/library/saxz13w4.aspx

BAB 10
LINQ

LINQ adalah singkatan dari Language-Integrated Query dan merupakan fitur menarik dari C# yang memungkinkan Anda untuk meminta data dalam program Anda. Dalam bab ini, kita akan membahas pengenalan singkat tentang LINQ diikuti dengan dua contoh bagaimana LINQ dapat digunakan. Pertama-tama mari kita pelajari cara menulis kueri LINQ. Sintaks khas untuk kueri LINQ adalah:
from… where… orderby… select

Misalkan kita memiliki array angka dan kita ingin memilih semua angka genap dari array. Kita dapat melakukannya dengan mudah dengan LINQ. Pertama, mari kita deklarasikan array.
int[] numbers = { 0, 1, 2, 3, 4, 5, 6 };

Selanjutnya, kita menulis kueri LINQ sebagai berikut:
var evenNumQuery =
from num in numbers
where (num % 2) == 0
select num;

Pertanyaannya adalah dari baris kedua hingga keempat. Pembaca yang memiliki pengalaman dengan SQL mungkin akan menemukan kueri ini cukup familier. Kueri terdiri dari tiga bagian. Bagian pertama:
from num in numbers

menyatakan bahwa kita melakukan kueri pada larik angka. num adalah nama yang kita gunakan untuk mewakili item individual dalam array. Baris berikutnya:
where (num % 2) == 0

menguji masing-masing item untuk menentukan apakah sisa angka dibagi 2 adalah nol. Jika ya, num adalah bilangan genap. Baris ketiga:
select num;

memilih semua elemen yang memenuhi kriteria ini.

Hasil ini kemudian diberikan ke variabel evenNumQuery,, yang dinyatakan sebagai tipe var. var adalah tipe data khusus yang kita gunakan setiap kali kita ingin kompiler menentukan tipe data itu sendiri. Ini diperlukan karena dalam contoh kita, tipe data evenNumQuery cukup kompleks; kita lebih baik membiarkan C# mencari tahu tipe datanya untuk kita. Setelah kita membuat pernyataan kueri, kita dapat menjalankan kueri dengan menulis:
foreach (int i in evenNumQuery)
{

```
        Console.WriteLine("{0} is an even number", i);
}
```

Jika Anda menjalankan kode ini, Anda akan mendapatkan:

```
0 is an even number
2 is an even number
4 is an even number
6 is an even number
```

Itu dia. Begitulah cara mudah menggunakan LINQ. Mari kita beralih ke contoh LINQ yang lebih kompleks. Misalkan Anda memiliki kelas Customer dengan Name, Phone, Address dan Balance sebagai propertinya dan konstruktor untuk menginisialisasi setiap properti ini.

Kita dapat membuat daftar objek Customer dalam metode Main() kita menggunakan kode di bawah ini:

```
List<Customer> customers = new List<Customer>();

customers.Add(new Customer("Alan", "80911291", "ABC Street", 25.60m));
customers.Add(new Customer("Bill", "19872131", "DEF Street",
-32.1m));
customers.Add(new Customer("Carl", "29812371", "GHI Street",
-12.2m));
customers.Add(new Customer("David", "78612312", "JKL Street", 12.6m));
```

Sekarang anggaplah kita ingin mencari semua pelanggan dengan saldo akun negatif, kita dapat menggunakan kueri LINQ berikut:

```
var overdue =

from cust in customers where
cust.Balance < 0
orderby cust.Balance ascending
select new { cust.Name, cust.Balance };
```

Kueri ini mirip dengan kueri pertama, dengan dua perbedaan utama. Di sini, kita menggunakan dua kata kunci tambahan, orderby dan ascending, untuk mengatur hasil dalam urutan menaik.

Selain itu, kita menggunakan kata kunci baru dalam pernyataan pilih. Kata kunci baru diperlukan setiap kali kita ingin memilih lebih dari satu bidang dari objek.

Untuk mengeksekusi dan mencetak hasilnya, kita dapat menggunakan loop foreach di bawah ini:

```
foreach (var cust in overdue)
Console.WriteLine("Name = {0}, Balance = {1}", cust.Name,cust.Balance);
```

Kita akan mendapatkan:
Name = Bill, Balance = -32.1
Name = Carl, Balance = -12.2

BAB 11
PENANGANAN FILE

Kita telah sampai pada bab terakhir buku ini sebelum proyek dimulai. Dalam bab ini, kita akan mempelajari cara membaca dan menulis ke file eksternal. Pada Bab 5 sebelumnya, kita telah mempelajari cara mendapatkan input dari pengguna menggunakan metode ReadLine(). Namun, dalam beberapa kasus, membuat pengguna memasukkan data ke dalam program kita mungkin tidak praktis, terutama jika program kita perlu bekerja dengan data dalam jumlah besar. Dalam kasus seperti ini, cara yang lebih mudah adalah menyiapkan informasi yang diperlukan sebagai file eksternal dan membuat program kita membaca informasi dari file tersebut.

C# memberi kita sejumlah kelas untuk bekerja dengan file. Kelas yang akan kita lihat dalam bab ini adalah kelas File, StreamWriter dan StreamReader. Ketiga kelas tersedia di namespace System.IO. Untuk menggunakan metode dalam bab ini, Anda harus menambahkan direktif using System.IO; ke awal kode Anda.

11.1 MEMBACA FILE TEKS

Untuk membaca data dari file teks, kita menggunakan kelas StreamReader. Misalkan kita ingin membaca data dari file “myFile.txt” yang terletak di drive C. Contoh di bawah ini menunjukkan bagaimana melakukannya.

```
1 string path = “c:\\myFile.txt”;
2 using (StreamReader sr = new StreamReader(path))
3 {
4 while (sr.EndOfStream != true)
5 {
6 Console.WriteLine(sr.ReadLine());
7 }
8
9 sr.Close();
10 }
```

Pada baris 1, pertama-tama kita mendeklarasikan jalur variabel string dan menetapkan jalur file ke variabel.

string path = “c:\\myFile.txt”;

Perhatikan bahwa kita harus menggunakan garis miring ganda \\ saat menulis jalur. Ini karena jika kita hanya menggunakan satu garis miring, kompilator akan menganggap garis miring tunggal adalah awal dari urutan escape dan menafsirkan \m sebagai urutan escape. Ini akan menghasilkan kesalahan.

Pada baris 2, kita membuat instance StreamReader. Konstruktor StreamReader mengambil satu argumen – jalur file yang akan dibaca.

StreamReader sr = new StreamReader(path)

Perhatikan bahwa kita membuat instance StreamReader ini di dalam sepasang tanda kurung yang mengikuti kata using pada baris 2?

Kata kunci using di sini berbeda dengan kata kunci yang kita gunakan saat menulis direktif. Kata kunci using di sini memastikan bahwa metode Dispose() selalu dipanggil. Metode Dispose() adalah metode yang telah ditulis sebelumnya di ruang nama Sistem yang menutup atau melepaskan semua sumber daya yang tidak dikelola seperti file dan aliran setelah tidak lagi diperlukan. Saat kita menggunakan kata kunci using, kita memastikan bahwa metode Dispose() dipanggil bahkan jika pengecualian terjadi dan mencegah kode kita mencapai Baris 9 tempat kita menutup file secara manual. Ini adalah praktik yang baik untuk selalu menggunakan kata kunci using setiap kali Anda berurusan dengan file. Kode untuk membaca dan menutup file diapit oleh kurung kurawal {} setelah pernyataan using.

Dari baris 4 sampai 7, kita menggunakan while loop untuk membaca file teks baris demi baris.

```
while (sr.EndOfStream != true)
{
        Console.WriteLine(sr.ReadLine());
}
```

EndOfStream adalah properti dari kelas StreamReader yang mengembalikan nilai true ketika akhir file tercapai. Selama akhir file tidak tercapai, loop while akan terus berjalan. Di dalam while loop, kita memiliki pernyataan:

```
Console.WriteLine(sr.ReadLine());
```

sr.ReadLine() membaca baris dari file teks dan mengembalikannya sebagai string. String ini kemudian dicetak ke layar menggunakan metode Console.WriteLine() Terakhir, setelah kita selesai membaca file, kita menutup file tersebut agar program lain dapat menggunakannya. Anda harus selalu menutup file Anda setelah Anda tidak lagi membutuhkannya.

```
sr.Close();
```

Itu dia. Begitulah cara Anda membaca file teks di C#. Cukup langsung bukan? Namun, ada satu masalah dengan kode di atas. Kode ini akan menghasilkan kesalahan jika file "myFile.txt" tidak dapat ditemukan. Kita memiliki dua pilihan di sini.

Opsi 1: coba…tangkap

Opsi pertama adalah menggunakan pernyataan try…catch seperti yang ditunjukkan di bawah ini:

```
1 try
2 {
3 using (StreamReader sr = new StreamReader(path))
4 {
5 while (!sr.EndOfStream)
6 {
7 Console.WriteLine(sr.ReadLine());
8 }
9 sr.Close();
10 }
11 }catch (FileNotFoundException e)
12 {
13 Console.WriteLine(e.Message);
14 }
```

Dari baris 1 hingga 11, kita mencoba membuka, membaca, dan menutup file di blok coba. Dari baris 11 hingga 14, kita menggunakan blok tangkap untuk menangkap FileNotFoundException pengecualian jika file tidak ditemukan. Di dalam blok tangkap, kita mencetak pernyataan kesalahan untuk memberi tahu pengguna bahwa file tidak ditemukan.

Opsi 2: File.Exists()

Metode kedua untuk menangani skenario "file tidak ditemukan" adalah dengan menggunakan metode Exists() di kelas File. Seperti namanya, metode Exists() memeriksa apakah ada file. Kelas File adalah kelas yang telah ditulis sebelumnya dalam namespace System.IO yang menyediakan metode statis untuk pembuatan, penyalinan, penghapusan, pemindahan, dan pembukaan satu file.

Untuk menggunakan metode Exists(), kita menggunakan pernyataan if untuk memeriksa apakah file ada sebelum menggunakan StreamReader untuk membuka dan membaca file.

```
if (File.Exists(path))
{
    using (StreamReader sr = new StreamReader(path))
    {
        while (!sr.EndOfStream)
        {
        Console.WriteLine(sr.ReadLine());
        }
        sr.Close();
    }
}else
{
    //Do something else
}
```

Di blok else, kita dapat menulis kode untuk membuat file jika tidak ditemukan. Seperti yang Anda lihat, dua metode untuk menangani kasus di mana file hilang cukup mirip. Namun, metode File.Exists() adalah metode yang disukai karena lebih cepat daripada pernyataan try…catch.

11.2 MENULIS KE FILE TEKS

Selanjutnya, mari kita lihat cara menulis ke file teks. Untuk menulis ke file teks, kita menggunakan kelas StreamWriter.

Jika Anda ingin menambahkan data ke file yang sudah ada, buat StreamWriter, contohnya seperti ini:
StreamWriter sw = new StreamWriter(path, true);

di mana path adalah path file dan true menunjukkan bahwa kita ingin menambahkan data.

Jika Anda ingin menimpa data yang ada dalam file, Anda membuat StreamWriter, contohnya seperti ini:
StreamWriter sw = new StreamWriter(path);

Saat kita membuat instance StreamWriter, konstruktor mencari file di jalur yang diberikan. Jika file tidak ditemukan, itu membuat file.

Setelah kita membuat instance objek StreamWriter, kita dapat mulai menulis ke file kita menggunakan metode WriteLine() seperti yang ditunjukkan di bawah ini:
sw.WriteLine("It is easy to write to a file.");

Setelah kita selesai menulis ke file, kita harus menutup file dengan menulis:
sw.Close();

Perhatikan bahwa ketika Anda menulis ke file teks, juga merupakan praktik yang baik untuk menyertakan kode Anda dalam pernyataan using. Kode di bawah ini menunjukkan contoh lengkap bagaimana semua ini bersatu.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace FileDemo
{
    class Program
    {
        static void Main(string[] args)
        {
        //declaring the path to the file
        string path = "myfile.txt";

        //Writing to the file
        using(StreamWriter sw=new StreamWriter(path, true))
        {
            sw.WriteLine("ABC");
            sw.WriteLine("DEF");
            sw.Close();
        }
```

```
        //Reading from the file
        if (File.Exists(path))
        {
            using(StreamReader sr=new StreamReader(path))
            {
                while (!sr.EndOfStream)
                {
                Console.WriteLine(sr.ReadLine());
                }
                sr.Close();
            }
        }
        Console.Read();
        }
    }
}
```

Dalam contoh ini, kita memilih untuk menambahkan data ke file kita saat kita menulisnya. Saat Anda menjalankan program ini untuk pertama kalinya, Anda akan mendapatkan:

ABC

EFG

sebagai file output dan tampilan layar. Jika Anda menjalankannya untuk kedua kalinya, Anda akan mendapatkan:

ABC

EFG

ABC

EFG

Karena path lengkap "myfile.txt" tidak diberikan dalam contoh ini, file teks akan dibuat di folder yang sama dengan file .exe, yang ada di FileDemo > FileDemo> Debug > folder Bin.

PROYEK – PERANGKAT LUNAK PENGGAJIAN SEDERHANA

Selamat!

Kita sekarang telah menyelesaikan konsep inti dalam C#. Dalam bab terakhir ini, kita akan membuat kaki kita basah dengan mengkodekan aplikasi konsol lengkap yang menghasilkan slip gaji sebuah perusahaan kecil. Siap?

Ringkasan

Pertama, mari kita buat aplikasi konsol baru dan beri nama CSProject. Aplikasi ini terdiri dari enam kelas seperti gambar di bawah ini.

Staff
Manager : Staff
Admin : Staff
File Reader
PaySlip
Program

Kelas Staf berisi informasi tentang setiap staf di perusahaan. Ini juga berisi metode virtual yang disebut CalculatePay()yang menghitung gaji setiap staf. Kelas Manajer dan Admin mewarisi kelas Staf dan menimpa Metode CalculatePay(). Kelas FileReader berisi metode sederhana yang membaca dari file .txt dan membuat daftar objek Staff berdasarkan konten dalam file .txt. Kelas PaySlip menghasilkan slip gaji setiap karyawan di perusahaan. Selain itu, juga menghasilkan ringkasan detail staf yang bekerja kurang dari 10 jam dalam sebulan. Terakhir, kelas Program berisi metode Main() yang bertindak sebagai titik masuk utama aplikasi kita.

Kelas Staf

Pertama, mari kita mulai dengan kelas Staff. Kelas Staff berisi informasi dasar tentang seorang karyawan dan menyediakan metode untuk menghitung gaji pokok. Ini berfungsi sebagai kelas induk dari mana dua kelas lain akan diturunkan.

Fields

Kelas ini memiliki satu bidang privatefloat yang disebut hourlyRate dan satu private int bidang yang disebut hWorked. Coba nyatakan sendiri bidang ini.

Properti

Selanjutnya, deklarasikan tiga properti publik yang diimplementasikan secara otomatis untuk kelas. Propertinya adalah TotalPay, BasicPay dan NameOfStaff.

TotalPay adalah properti float dan memiliki setter yang dilindungi. BasicPay adalah properti float dan memiliki setter pribadi. NameOfStaff adalah properti string dan memiliki setter pribadi. Getter dari ketiga properti bersifat publik. Oleh karena itu, Anda tidak perlu mendeklarasikan pengubah akses dari getter ini karena mereka memiliki tingkat akses yang sama dengan properti.

Selain ketiga metode yang diterapkan secara otomatis ini, kelas Staf juga memiliki properti publik yang disebut HoursWorked. Bidang pendukung untuk properti ini adalah bidang hWorked. Properti ini memiliki pengambil yang hanya mengembalikan nilai hWorked.

Penyetel memeriksa apakah nilai yang disetel untuk HoursWorked lebih besar dari 0. Jika ya, ia memberikan nilai ke hWorked. Jika tidak, itu memberikan 0 ke hWorked. Coba nyatakan sendiri properti ini. Anda dapat merujuk ke Bab 7 untuk bantuan.

Konstruktor

Kelas Staff memiliki konstruktor publik dengan dua parameter, nama (string) dan rate (float). Di dalam konstruktor, kita menetapkan dua parameter masing-masing ke properti NameOfStaff dan field hourlyRate. Coba coding konstruktor ini sendiri.

Metode

Sekarang, mari kita tulis metode untuk kelas. Pertama, kita akan membuat kode metode virtual yang disebut CalculatePay().

CalculatePay()bersifat publik, tidak memiliki parameter dan tidak mengembalikan nilai. Metode ini melakukan tiga hal:

Pertama, ia mencetak baris "Menghitung Pembayaran ..." di layar. Selanjutnya, ia menetapkan nilai hWorked*hourlyRate ke properti BasicPay. Terakhir, ia memberikan nilai BasicPay ke properti TotalPay. Dengan kata lain, BasicPay dan TotalPay akan memiliki nilai yang sama. Coba coding metode ini sendiri. Terakhir, tulis metode ToString()untuk menampilkan nilai bidang dan properti kelas Staf. Itu saja yang ada untuk kelas Staf. Tabel di bawah ini menunjukkan ringkasan kelas Staf:

Fields

private float hourlyRate
private int hWorked (backing field for HoursWorked)

Properti

public float TotalPay
public float BasicPay
public string NameOfStaff
public int HoursWorked

Konstruktor

public Staff(string name, float rate)

Metode

public virtual void CalculatePay()
public override string ToString()

Manajer: Kelas Staf

Selanjutnya, mari kita beralih ke kode kelas Manajer.

Fields

Kelas Manajer adalah kelas anak dari kelas Staf. Ini memiliki satu bidang private const yang disebut managerHourlyRate yang bertipe float. Coba deklarasikan bidang ini dan inisialisasi dengan nilai 50.

Properti

Manager juga memiliki properti publik yang diterapkan secara otomatis yang disebut Allowance. Tunjangan bertipe int dan memiliki setter pribadi. Coba coding properti ini.

Konstruktor

Sekarang, mari kita deklarasikan konstruktor untuk Manager. Kelas Manager memiliki public konstruktor dengan parameter string, nama.

Tugas konstruktor adalah memanggil konstruktor dasar dan meneruskan nama parameter dan field managerHourlyRate ke konstruktor dasar. Selain itu, konstruktor anak tidak melakukan apa pun. Oleh karena itu, tidak ada apa pun di dalam kurung kurawal konstruktor anak. Coba coding konstruktor ini sendiri. Anda dapat merujuk ke ringkasan kelas Manajer di bawah ini untuk bantuan jika Anda memiliki masalah dalam mengkodekan konstruktor.

Metode

Selanjutnya, mari buat kode metode untuk menimpa metode CalculatePay()di kelas Staf. Karena Manajer berasal dari Staf, ia memiliki akses ke properti BasicPay, TotalPaydan HoursWorked yang dideklarasikan di kelas Staf. Selain itu, Manajer juga memiliki properti sendiri – Tunjangan. Kita akan menggunakan keempat properti ini dalam metode ini.

Pertama, mari kita deklarasikan metodenya. CalculatePay() bersifat publik dan tidak mengembalikan nilai apa pun. Kita harus menggunakan kata kunci override saat mendeklarasikan metode ini karena akan menimpa metode CalculatePay() di kelas Staf. Dalam metode CalculatePay() di kelas Manajer, pertama-tama kita akan memanggil CalculatePay() metode di kelas induk dan menggunakannya untuk mengatur nilai-nilai BasicPay dan TotalPay. Untuk memanggil metode virtual di kelas induk, Anda harus menggunakan kata kunci dasar. Tambahkan baris berikut ke metode CalculatePay() Anda.

```
base.CalculatePay();
```

Ini memanggil metode CalculatePay()di kelas dasar (induk), yang menetapkan nilai BasicPay dan TotalPay. Setelah mengatur nilai dari kedua properti ini, mari kita lanjutkan untuk mengatur nilai Allowance. Kita akan menetapkan nilainya menjadi 1000. Selanjutnya, kita ingin mengubah nilai TotalPay. Berdasarkan metode CalculatePay() di kelas dasar, TotalPay sama dengan BasicPay, keduanya sama dengan produk hWorked dan hourlyRate.

Namun, di kelas anak Manajer, kita ingin memperbarui nilai TotalPay dengan menambahkan tunjangan ke dalamnya. Misalkan seorang manajer dibayar tunjangan sebesar Rp 15.000.000 jika dia bekerja lebih dari 160 jam dalam bulan itu. Coba gunakan pernyataan if untuk memperbarui nilai TotalPay berdasarkan nilai HoursWorked. Setelah memperbarui nilai TotalPay, metode CalculatePay()selesai. Terakhir, kita perlu mengkodekan metode ToString() untuk kelas Manager. Coba coding metode ini. Setelah Anda selesai, kelas Manajer selesai. Tabel di bawah ini menunjukkan ringkasan kelas Manajer:

Fields

private const float managerHourlyRat

Properti

public int Allowance

Konstruktor

public Manager(string name) : base(name, managerHourlyRate)

Metode

public override void CalculatePay()
public override string ToString()

Admin : Kelas Staf

Kelas selanjutnya adalah kelas Admin yang juga merupakan turunan dari kelas Staff.

Fields

Kelas Admin memiliki dua bidang const pribadi: overtimeRate dan adminHourlyRate. Kedua bidang bertipe float. Coba deklarasikan kedua bidang ini dan inisialisasi dengan nilai masing-masing 15,5 dan 30.

Properti

Selanjutnya, coba deklarasikan properti publik yang diterapkan secara otomatis, Lembur. Lembur adalah tipe float dan memiliki setter pribadi.

Konstruktor

Sekarang, mari kita deklarasikan konstruktornya. Mirip dengan konstruktor kelas Manajer, konstruktor kelas Admin bersifat publik dan memiliki satu parameter string, nama. Tugasnya hanyalah memanggil konstruktor dasar dan meneruskan nama parameter dan bidang adminHourlyRate ke konstruktor dasar.

Metode

Terakhir, kita siap untuk mengkodekan metode CalculatePay()untuk kelas Admin. Metode CalculatePay()di kelas Admin sangat mirip dengan metode di kelas Manajer. Mari kita deklarasikan metodenya terlebih dahulu. Selanjutnya, di dalam kurung kurawal, kita menggunakan metode CalculatePay() dari kelas dasar untuk mengatur properti BasicPay dan TotalPay dari staf admin. Setelah menetapkan nilai dari kedua properti ini, kita memeriksa apakah HoursWorked lebih besar dari 160. Jika ya, kita akan memperbarui nilai properti TotalPay. Misalkan seorang staf admin dibayar upah lembur di atas gaji pokok jika dia bekerja lebih dari 160 jam. Coba gunakan pernyataan if untuk memperbarui TotalPay milik staf admin. Uang lembur dihitung dengan rumus berikut:

Overtime = overtimeRate * (HoursWorked - 160);

di mana overtimeRate adalah bidang pribadi di kelas Admin dan Lembur adalah properti di kelas yang sama. HoursWorked adalah properti yang diwarisi dari kelas Staff. Selesai? Bagus! Sekarang, lanjutkan ke kode metode ToString(). Dengan itu, kelas Admin selesai. Tabel di bawah ini menunjukkan ringkasan kelas:

Fields

private const float overtimeRate

private const float adminHourlyRate

Properti

public float Overtime

Konstruktor

public Admin(string name) : base(name, adminHourlyRate)

Metode

public override void CalculatePay()
public override string ToString()

Kelas FileReader

Sekarang, kita siap untuk mengkode kelas FileReader. Kelas FileReader relatif mudah. Ini terdiri dari satu metode publik yang disebut ReadFile()yang tidak memiliki parameter. Metode mengembalikan daftar objek Staf. Deklarasi metode adalah sebagai berikut: public List<Staff> ReadFile()

```
{
}
```

Metode ReadFile() membaca dari file .txt yang terdiri dari nama dan posisi staf. Formatnya adalah:
Name of Staff, Position of Staff

Contohnya adalah:
Yvonne, Manager
Peter, Manager
John, Admin
Carol, Admin

Nama file teks adalah “staff.txt” dan disimpan dalam folder yang sama dengan file .exe. Buat sendiri file ini menggunakan Notepad dan simpan di folder CSProject > CSProject > Bin > Debug tempat file .exe berada. Sekarang, kita dapat mulai mengkodekan metode ReadFile(). Pertama-tama kita mendeklarasikan empat variabel lokal bernama myStaff, result, path dan separator seperti yang ditunjukkan di bawah ini.

```
List<Staff> myStaff = new List<Staff>();
string[] result = new string[2];
string path = "staff.txt";
string[] separator = {“, ”};
```

Selanjutnya, kita memeriksa apakah file “staff.txt” ada menggunakan pernyataan if dan

File.Exists()metode. Anda perlu menambahkan arahan using System.IO;

untuk menggunakan metode File.Exists()

Jika file ada, kita menggunakan objek StreamReader untuk membaca file teks baris demi baris. (Lihat Bab 11 jika Anda memerlukan bantuan dengan ini.) Setiap kali kita membaca sebuah baris, kita menggunakan metode Split() (lihat Bab 4) untuk membagi baris menjadi dua bagian dan menyimpan hasilnya dalam larik hasil. Misalnya, ketika kita membaca baris pertama, metode Split() membaginya menjadi dua string “Yvonne” dan “Manager”. Oleh karena itu, result[0] = “Yvonne” result[1] = “Manager”. Berdasarkan nilai result[1], kita menggunakan pernyataan if untuk membuat objek Manager jika nilai result[1]adalah "Manager" atau objek Admin jika nilainya adalah "Admin". Kita menambahkan objek ini ke daftar myStaff.

Setelah selesai membaca file, kita tutup file tersebut menggunakan metode Close(). Jika file tidak ada, kita menampilkan pesan untuk memberi tahu pengguna tentang kesalahan tersebut. Terakhir, kita mengembalikan daftar myStaff ke pemanggil setelah pernyataan if-else. Itu saja yang ada di kelas FileReader. Kita tidak perlu mendeklarasikan konstruktor untuk kelas ini. Kita hanya akan menggunakan konstruktor default yang dibuat C# untuk kita secara otomatis. Ringkasan untuk kelas FileReader ditunjukkan di bawah ini:

Metode

public List<Staff> ReadFile()

Kelas Slip Gaji

Sekarang, mari buat kode kelas PaySlip. Kelas ini sedikit berbeda dari kelas lain yang telah kita lihat sejauh ini. Selain memiliki bidang, properti, metode, dan konstruktor, kelas PaySlip juga memiliki enum yang disebut MonthsOfYear.

Fields

Pertama, mari kita mendeklarasikan bidang. Kelas memiliki dua bidang int pribadi bernama bulan
dan tahun. Coba nyatakan.

enum

Selanjutnya, kita akan mendeklarasikan enum bernama MonthsOfYear di dalam kelas PaySlip. MonthsOfYear mewakili dua belas bulan dalam setahun, di mana JAN = 1, FEB = 2 dll. Coba deklarasikan sendiri enum ini. Anda tidak perlu menentukan pengubah akses apa pun untuk enum ini. Enum yang dideklarasikan di dalam kelas adalah pribadi secara default.

Konstruktor

Sekarang, coba tambahkan konstruktor ke kelas PaySlip. Konstruktor bersifat publik dan memiliki dua parameter int payMonth dan payYear. Di dalam konstruktor, kita menetapkan dua parameter ke bidang pribadi bulan dan tahun masing-masing.

Metode

Selanjutnya, mari kita mengkodekan metode GeneratePaySlip()Metode ini mengambil daftar dari Staf menolak dan tidak mengembalikan apa pun. Deklarasi metode adalah:

public void GeneratePaySlip(List<Staff> myStaff)

```
{
}
```

Di dalam metode, kita mendeklarasikan variabel string yang disebut path. Selanjutnya, masih dalam metode GeneratePaySlip(), kita menggunakan loop foreach untuk mengulang elemen-elemen di myStaff. Hal ini dapat dilakukan sebagai berikut:

```
foreach (Staff f in myStaff)
{
}
```

Segala sesuatu yang mengikuti dari sini untuk metode GeneratePaySlip() harus dikodekan di dalam kurung kurawal dari foreach loop. Pertama, kita menetapkan nilai ke variabel jalur berdasarkan nama staf. Ingat bahwa kelas Staf memiliki properti bernama NameOfStaff? Misalkan NameOfStaff = "Yvonne", kita ingin menetapkan string "Yvonne.txt" ke variabel path.

Bagaimana Anda melakukannya? Coba coding sendiri. (Petunjuk: Anda dapat menggunakan f.NameOfStaff untuk mengakses nama staf dan menggunakan operator + untuk menggabungkan ekstensi ".txt") Setelah menetapkan nilai ke jalur, kita ingin membuat instance objek StreamWriter untuk menulis ke file di jalur yang ditentukan oleh variabel jalur, menimpa konten yang ada di file sehingga setiap slip gaji yang dihasilkan tidak berisi konten dari bulan sebelumnya. Lihat Bab 11 jika Anda lupa cara menggunakan kelas StreamWriter. Sebut saja objek StreamWriter sw. Kita kemudian dapat melanjutkan untuk menggunakan serangkaian pernyataan sw.WriteLine() untuk menghasilkan slip gaji setiap karyawan.

Slip gaji khas untuk seorang manajer terlihat seperti ini:

```
1 PAYSLIP FOR DEC 2010
2 ==========================
3 Name of Staff: Yvonne
4 Hours Worked: 1231
5
6 Basic Pay: $61,550.00
7 Allowance: $1,000.00
8
9 ==========================
10 Total Pay: $62,550.00
11 ==========================
```

Angka-angka di sebelah kiri ditambahkan untuk referensi dan bukan bagian dari slip gaji yang sebenarnya. Slip gaji biasa untuk staf admin terlihat serupa kecuali untuk baris 7. Untuk staf admin, baris 7 akan membaca sesuatu seperti:

Overtime Pay: $1,286.50

Sekarang mari kita lihat cara membuat slip gaji ini. Untuk menulis baris 1, kita perlu mengakses bidang bulan dan tahun di kelas. Karena bulan adalah bilangan bulat, kita perlu

memasukkannya ke dalam nilai enum MonthsOfYear sehingga akan ditulis sebagai DEC, bukan 12. Pernyataan di bawah ini menunjukkan bagaimana baris 1 dapat ditulis.

```
sw.WriteLine("PAYSLIP FOR {0} {1}", (MonthsOfYear)month, year);
```

Baris 2 mudah ditulis. Itu hanya terdiri dari serangkaian tanda sama dengan (=). Coba coding sendiri. Untuk menulis baris 3 dan 4, kita perlu mengakses properti NameOfStaff dan HoursWorked di kelas Staff. Pernyataan di bawah ini menunjukkan bagaimana hal itu dapat dilakukan untuk baris 3.

```
sw.WriteLine("Name of Staff: {0}", f.NameOfStaff);
```

Coba coding baris 4 sendiri. Selanjutnya, kita menggunakan sw.WriteLine(“”); pernyataan untuk mencetak baris kosong. Untuk menulis baris 6, kita perlu mengakses properti BasicPay di kelas Staff. Selain itu, kita juga perlu menggunakan specifier C untuk menampilkan properti BasicPay dalam notasi mata uang (lihat Bab 5). Cobalah sendiri.

Baris 7 lebih sulit karena kita perlu menentukan jenis runtime dari objek saat ini di loop foreach. Kita belajar bagaimana melakukannya di Bab 8. Jika contoh saat ini adalah objek Manajer, kita mengakses dan mencetak properti Allowance di kelas Manajer. Untuk mengakses properti Allowance di kelas Manager, kita perlu memasukkan f ke objek Manager dengan menulis

```
((Manager)f).Allowance
```

Jika instance saat ini adalah objek Admin, kita mengakses dan mencetak Lembur properti di kelas Admin. Coba coding baris 7 sendiri. Baris 8 adalah baris kosong lain dan baris 9 terdiri dari serangkaian tanda sama dengan. Baris 10 menunjukkan total gaji staf saat ini, yang bisa kita dapatkan dari properti TotalPay dari kelas Staf. Akhirnya, baris 11 adalah baris lain yang terdiri dari tanda sama dengan. Coba coding baris-baris ini sendiri.

Terakhir yang tidak kalah penting, setelah membuat slip gaji untuk setiap staf, kita perlu menutup file menggunakan metode sw.Close(). Itu membawa kita ke akhir metode GeneratePaySlip(). Setelah Anda selesai mengkodekan metode ini, kita dapat melanjutkan ke metode berikutnya di kelas Slip Gaji.

Metode selanjutnya menghasilkan ringkasan karyawan yang bekerja kurang dari 10 jam di bulan itu. Sebut saja metode ini GenerateSummary(). Seperti metode GeneratePaySlip(), metode GenerateSummary() bersifat publik, mengambil daftar objek Staf dan tidak mengembalikan nilai apa pun. Coba nyatakan sendiri metode ini.

Di dalam metode GenerateSummary(), kita menggunakan LINQ untuk memilih semua karyawan yang bekerja kurang dari 10 jam di bulan itu. Kita ingin mengetahui properti NameOfStaff dan HoursWorked untuk karyawan ini. Selain itu, kita ingin mengatur hasilnya dalam urutan menaik berdasarkan NameOfStaff. Coba coding pernyataan LINQ ini sendiri dan tetapkan hasilnya ke variabel var yang disebut result. Anda dapat merujuk ke Bab 10 untuk bantuan. Selesai? Bagus. Selanjutnya, mari kita mendeklarasikan jalur variabel string dan menetapkan string "summary.txt"

Sekarang kita siap untuk menulis ke “summary.txt”. Deklarasikan instance StreamWriter untuk menulis ke file ini. File "summary.txt" khas terlihat seperti ini (angka di sebelah kiri ditambahkan untuk referensi):

```
1 Staff with less than 10 working hours
2
3 Name of Staff: Carol, Hours Worked: 2
4 Name of Staff: Peter, Hours Worked: 6
```

Baris 1 dan 2 seharusnya cukup mudah untuk dikodekan. Coba coding sendiri. Untuk mencetak baris 3 dan 4, kita perlu menggunakan loop foreach untuk mengulang setiap elemen dalam variabel hasil yang diperoleh dari pernyataan LINQ. Coba coding ini sendiri. Setelah menampilkan hasilnya, Anda dapat menutup file “summary.txt” menggunakan metode Close().

Itu saja untuk metode GenerateSummary() kita. Setelah mengkode metode GenerateSummary()kita hanya perlu mengkodekan metode ToString() dan class PaySlip kita selesai. Tabel di bawah ini menunjukkan ringkasan kelas PaySlip:

Fields

private int month
private int year

Enum

enum MonthsOfYear

Konstruktor

public PaySlip(int payMonth, int payYear)

Metode

public void GeneratePaySlip(List<Staff> myStaff)
public void GenerateSummary(List<Staff> myStaff)
public override string ToString()

Kelas Program

Sekarang kita telah sampai pada bagian terpenting dari proyek – kelas Program. Kelas Program hanya memiliki satu metode – metode Main().

Metode Utama()

Pertama, mari kita mendeklarasikan empat variabel lokal untuk metode Main(). Yang pertama adalah daftar objek Staf. Kita akan menyebut daftar ini myStaff. Berikutnya adalah objek FileReader bernama fr. Dua sisanya adalah variabel int. Mari kita sebut mereka bulan dan tahun dan inisialisasi ke nol. Coba deklarasikan sendiri variabel lokal ini. Sekarang, kita akan menggunakan while loop dan pernyataan try catch untuk meminta pengguna memasukkan tahun untuk slip gaji. Loop akan berulang kali meminta pengguna untuk memasukkan tahun hingga mendapatkan nilai yang valid.

Untuk melakukan itu, kita menggunakan loop sementara di bawah ini:

```
1 while (year == 0)
2 {
3 Console.Write("\nPlease enter the year: ");
4
5 try
6 {
7 //Code to convert the input to an integer
8 }
9 catch (FormatException)
10 {
11 //Code to handle the exception
12 }
13 }
```

Di dalam blok try (Baris 7), kita membaca nilai yang dimasukkan pengguna dan mencoba mengubahnya menjadi integer. Kita kemudian menetapkannya ke tahun variabel. Jika berhasil, tahun tidak lagi menjadi nol dan loop while akan keluar. Coba coding blok try sendiri.

Jika konversi tidak berhasil, kita menangkap kesalahan di blok tangkap untuk mencegah program mogok. Coba coding pesan kesalahan di blok catch (Baris 11). Ketika konversi tidak berhasil, tahun tetap nol dan sementara loop berlanjut. Pengguna akan berulang kali diminta untuk memasukkan tahun sampai mereka memasukkan nilai yang valid. Setelah Anda selesai dengan blok while ini, Anda dapat melanjutkan ke kode blok while untuk meminta pengguna memasukkan bulan. Blok while untuk variabel bulan sangat mirip dengan blok variabel tahun. Namun, kita ingin melakukan lebih banyak pemeriksaan untuk variabel bulan.

Di blok coba, pertama-tama kita mencoba mengonversi input ke bilangan bulat dan menetapkannya ke variabel bulan. Jika berhasil, kita menggunakan pernyataan if untuk memeriksa apakah bulan kurang dari 1 atau lebih besar dari 12. Jika ya, input tidak valid. Kita akan menampilkan pesan kesalahan untuk memberi tahu pengguna bahwa mereka telah memasukkan nilai yang tidak valid. Selain itu, kita juga akan mengatur ulang bulan ke nol sehingga loop sementara akan berulang. Coba coding ini coba blokir sendiri. Setelah mengkode blok coba, Anda dapat melanjutkan ke kode blok tangkap yang hanya memberi tahu pengguna tentang kesalahan tersebut. Selesai? Bagus.

Selanjutnya, kita akan menambahkan item ke daftar myStaff kita. Kita melakukannya dengan menggunakan objek fr untuk memanggil metode ReadFile() di kelas FileReader dan menetapkan hasilnya ke myStaff. Kita kemudian dapat mulai menghitung gaji untuk setiap staf. Kita akan menggunakan yang berikut untuk loop seperti ini:

```
for (int i = 0; i < myStaff.Count; i++)
{
    try
    {
    }
    catch (Exception e)
    {
    }
```

}

Dalam perulangan for, kita menggunakan pernyataan try catch. Di blok try, kita melakukan hal berikut:
Pertama, minta pengguna untuk memasukkan jumlah jam kerja untuk setiap staf. Contoh dari prompt adalah:
Enter hours worked for Yvonne:

di mana "Yvonne" adalah nama staf. Anda perlu mengakses NameOfStaff properti untuk setiap staf dengan menulis myStaff[i].NameOfStaff. Selanjutnya, baca inputnya, coba ubah menjadi bilangan bulat dan tetapkan ke Properti HoursWorked dari objek Staff.

Setelah itu, kita memanggil metode HitungPay() pada objek Staf untuk menghitung gaji staf tersebut. Terakhir, kita menggunakan metode ToString() untuk mendapatkan informasi tentang objek Staff dan menampilkan informasi ini di layar menggunakan metode Console.WriteLine(). Coba coding ini coba blokir sendiri.

Selanjutnya di blok catch, kita mencoba menangkap kesalahan yang mungkin terjadi. Di dalam blok tangkap ini, kita cukup menampilkan pesan kesalahan dan mengurangi nilai i per satu (i--;) sehingga perulangan for akan berulang lagi untuk objek Staff saat ini alih-alih berpindah ke elemen berikutnya di myStaff.

Coba koding sendiri blok tangkapan ini. Dengan itu, kita sampai pada akhir perulangan for. Kita sekarang siap untuk membuat slip gaji untuk setiap staf. Untuk melakukan itu, pertama-tama kita harus mendeklarasikan dan membuat instance objek PaySlip. Sebut saja objek itu ps. Kita meneruskan variabel bulan dan tahun ke konstruktor saat membuat instance objek. Selanjutnya, kita menggunakan objek ps untuk memanggil metode GeneratePaySlip() dan GenerateSummary() dan meneruskan myStaff sebagai argumen. Terakhir, kita tambahkan Console.Read(); pernyataan untuk mencegah konsol menutup segera setelah program berakhir.

Selesai?

Jika Anda telah berhasil mengkodekan program Main(), beri tepukan pada diri Anda sendiri. Anda baru saja mengkodekan program lengkap dalam C#! Sudah selesai dilakukan dengan baik! Jika Anda memiliki masalah dalam mengkodekannya, teruslah mencoba. Anda dapat merujuk ke yang disarankan solusi dalam Lampiran A untuk referensi. Setelah Anda selesai mengkode metode Main(), Anda siap untuk menjalankan program Anda. Bersemangat? Ayo lakukan!

Klik tombol “Start” untuk menjalankan program dan masukkan nilai yang diminta. Slip gaji yang dihasilkan dapat ditemukan di folder yang sama dengan file .exe, yaitu di folder CSProject > CSProject > Bin > Debug. Cobalah membuat kesalahan dan memasukkan huruf alfabet alih-alih angka. Bermain-main dengan program untuk melihat cara kerjanya. Apakah semuanya berjalan seperti yang diharapkan? Jika berhasil, bagus! Anda telah melakukan pekerjaan yang sangat baik! Cobalah untuk memikirkan cara untuk meningkatkan perangkat lunak. Misalnya, Anda dapat menyertakan lebih banyak pemeriksaan untuk memastikan

bahwa pengguna memasukkan nilai yang benar untuk tahun dan Jam Kerja. Jika kode Anda tidak berfungsi, bandingkan dengan contoh jawaban dan coba cari tahu apa yang salah. Anda akan belajar banyak dengan menganalisis kesalahan Anda. Pemecahan masalah adalah di mana letak kesenangannya dan di mana imbalannya adalah yang terbesar. Bersenang-senanglah dan jangan pernah menyerah! Contoh jawaban dapat ditemukan di Lampiran.

DAFTAR PUSTAKA

Heriyanto, Abdul Kadir. 2006. Algoritma Pemrograman Menggunakan C++. Yogyakarta: Andi.

Raharjo, Budi. 2006. Pemrograman C++. Bandung: Informatika.

Heri Sismoro, 2005. Pengantar Logika Informatika, Algoritma dan Pemrograman Komputer, Yogyakarta: ANDI.

Jogiyanto Hartono, 2000. Konsep Dasar Pemrograman Bahasa C, Yogyakarta: ANDI.

S, R. A. (2018). Logika Algoritma dan Pemrograman Dasar. Bandung: Modula.

Sjukani, M. (2014). Algoritma dan Struktur Data 1 dengan C, C++ dan Java (Edisi 9 ed.). Jakarta: MItra Wacana Media.

Miles, Rob. 2016. C# Programming Yellow Book. Cheese Edition.

Munir, Rinaldi, 2016. Algoritma dan Pemrograman Dalam Bahasa Pascal, C dan C++, Edisi Keenam, Bandung: Informatika.

Hanif al fatta (2006). *Dasar Pemrograman C++ ditemani dengan Pengenalan Pemrograman Berpandangan Objek*. ISBN 979-763-582-1.

Kadir, Abdul. 2012. Buku Pintar C++ untuk Pemula. Yogyakarta: Penerbit Andi.

Sitorus, Lamhot & David Sembiring. 2012. Konsep dan Implementasi Struktur Data dengan C++. Yogyakarta: Penerbit Andi.

Agung, Gragorius. 2014. Step by Step Visual C#. Yogyakarta: PT Elex Media Komputindo.

LAMPIRAN

LEMBAR JAWABAN PROYEK C#

Kode sumber untuk program ini dapat diunduh di http://www.learncodingfast.com/csharp.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace CSProject
{
class Program
{
static void Main(string[] args)
{
List<Staff> myStaff = new List<Staff>();
FileReader fr = new FileReader();
int month = 0, year = 0;

while (year == 0)
{
Console.Write("\nPlease enter the year: ");

try
{
year = Convert.ToInt32(Console.ReadLine());
}
catch (Exception e)
{
Console.WriteLine(e.Message + " Please try again.");
}
}

while (month == 0)
{
Console.Write("\nPlease enter the month: ");

try
{
month = Convert.ToInt32(Console.ReadLine());
```

```
if (month < 1 || month > 12)
{
Console.WriteLine("Month must be from 1 to 12. Please try again.");
month = 0;
}
}
catch (Exception e)
{
Console.WriteLine(e.Message + " Please try again.");
}
}

myStaff = fr.ReadFile();

for (int i = 0; i< myStaff.Count; i++)
{
try
{
Console.Write("\nEnter hours worked for {0}: ",
myStaff[i].NameOfStaff);
myStaff[i].HoursWorked = Convert.ToInt32(Console.ReadLine());
myStaff[i].CalculatePay();
Console.WriteLine(myStaff[i].ToString());
}
catch (Exception e)
{
Console.WriteLine(e.Message);
i--;
}
}

PaySlip ps = new PaySlip(month, year);
ps.GeneratePaySlip(myStaff);
ps.GenerateSummary(myStaff);

Console.Read();
}
}

class Staff
{
private float hourlyRate;
private int hWorked;

public float TotalPay { get; protected set; }
public float BasicPay { get; private set; }
public string NameOfStaff { get; private set; }
```

```
public int HoursWorked
{
get
{
return hWorked;
}
set
{
if (value > 0)
hWorked = value;
else
hWorked = 0;
}
}

public Staff(string name, float rate)
{
NameOfStaff = name;
hourlyRate = rate;
}

public virtual void CalculatePay()
{
Console.WriteLine("Calculating Pay...");

BasicPay = hWorked * hourlyRate;
TotalPay = BasicPay;
}

public override string ToString()
{
return "\nNameOfStaff = " + NameOfStaff
+ "\nhourlyRate = " + hourlyRate + "\nhWorked = " + hWorked
+ "\nBasicPay = " + BasicPay + "\n\nTotalPay = " + TotalPay;
}
}

class Manager : Staff
{
private const float managerHourlyRate = 50;

public int Allowance { get; private set; }

public Manager(string name) : base(name, managerHourlyRate) { }
```

```
public override void CalculatePay()
{
base.CalculatePay();

Allowance = 1000;

if (HoursWorked > 160)
TotalPay = BasicPay + Allowance;
}

public override string ToString()
{
return "\nNameOfStaff = " + NameOfStaff + "\nmanagerHourlyRate = "
+ managerHourlyRate + "\nHoursWorked = " + HoursWorked +
"\nBasicPay = "
+ BasicPay + "\nAllowance = " + Allowance + "\n\nTotalPay = " +
TotalPay;
}
}

class Admin : Staff
{
private const float overtimeRate = 15.5f;
private const float adminHourlyRate = 30f;

public float Overtime { get; private set; }

public Admin(string name) : base(name, adminHourlyRate) { }

public override void CalculatePay()
{
base.CalculatePay();

if (HoursWorked > 160)
Overtime = overtimeRate * (HoursWorked - 160);
}

public override string ToString()
{
return "\nNameOfStaff = " + NameOfStaff
+ "\nadminHourlyRate = " + adminHourlyRate + "\nHoursWorked = " +
HoursWorked
+ "\nBasicPay = " + BasicPay + "\nOvertime = " + Overtime
+ "\n\nTotalPay = " + TotalPay;
}
}
```

```
class FileReader
{
public List<Staff> ReadFile()
{
List<Staff> myStaff = new List<Staff>();
string[] result = new string[2];
string path = "staff.txt";
string[] separator = { ", " };

if (File.Exists(path))
{
using (StreamReader sr = new StreamReader(path))
{
while (!sr.EndOfStream)
{
result = sr.ReadLine().Split(separator,
StringSplitOptions.RemoveEmptyEntries);

if (result[1] == "Manager")
myStaff.Add(new Manager(result[0]));
else if (result[1] == "Admin")
myStaff.Add(new Admin(result[0]));
}
sr.Close();
}
}else
{
Console.WriteLine("Error: File does not exist");
}

return myStaff;
}
}

class PaySlip
{
private int month;
private int year;

enum MonthsOfYear { JAN = 1, FEB = 2, MAR, APR, MAY, JUN, JUL, AUG,
SEP, OCT, NOV, DEC }

public PaySlip(int payMonth, int payYear)
{
month = payMonth;
year = payYear;
```

```
}

public void GeneratePaySlip(List<Staff> myStaff)
{
string path;

foreach (Staff f in myStaff)
{
path = f.NameOfStaff + ".txt";

using (StreamWriter sw = new StreamWriter(path))
{
sw.WriteLine("PAYSLIP FOR {0} {1}", (MonthsOfYear)month, year);
sw.WriteLine("====================");
sw.WriteLine("Name of Staff: {0}", f.NameOfStaff);
sw.WriteLine("Hours Worked: {0}", f.HoursWorked);
sw.WriteLine("");
sw.WriteLine("Basic Pay: {0:C}", f.BasicPay);

if (f.GetType() == typeof(Manager))
sw.WriteLine("Allowance: {0:C}", ((Manager)f).Allowance);
else if (f.GetType() == typeof(Admin))
sw.WriteLine("Overtime: {0:C}", ((Admin)f).Overtime);

sw.WriteLine("");
sw.WriteLine("====================");
sw.WriteLine("Total Pay: {0:C}", f.TotalPay);
sw.WriteLine("====================");

sw.Close();
}
}

}

public void GenerateSummary(List<Staff> myStaff)
{
var result
= from f in myStaff
where f.HoursWorked < 10
orderby f.NameOfStaff ascending
select new { f.NameOfStaff, f.HoursWorked };

string path = "summary.txt";

using (StreamWriter sw = new StreamWriter(path))
```

```
{
sw.WriteLine("Staff with less than 10 working hours");
sw.WriteLine("");

foreach (var f in result)
sw.WriteLine("Name of Staff: {0}, Hours Worked: {1}",
f.NameOfStaff, f.HoursWorked);

sw.Close();
}
}

public override string ToString()
{
return "month = " + month + "year = " + year;
}
}
}
```

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

Pemrograman Bahasa C#

BIODATA PENULIS

Dr. Budi Raharjo, S.Kom, M.Kom, MM lahir di Semarang, tanggal 22 Februari 1985. Beliau adalah Alumni dari Universitas Bina Nusantara (BINUS University) Jakarta dan juga alumni Universitas Kristen Satya wacana (UKSW) Salatiga. Dr. Budi Raharjo telah menjadi Dosen pada Universitas STEKOM pada mata kuliah Kepemimpinan (Leadership), mata kuliah Pengantar Akuntansi, Manajemen Proses, Manajemen Akuntansi dan Manajemen Resiko Bisnis. Selain sebagai dosen Universitas STEKOM, Dr. Budi Raharjo, M.Kom, MM juga mempunyai bisnis sendiri dalam bidang perhotelan dan juga sebagai wirausaha dalam bidang pemasok unggas (ayam) beku, ke berbagai kota besar, khususnya Jakarta dan sekitarnya.

Pengalaman beliau berwirausaha menjadi bekal utama dalam penulisan buku ajar yang diterbitkan oleh Yayasan Prima Agus Teknik (YPAT) Semarang. Oleh sebab itu bukunya berisi langkah langkah praktis yang mudah diikuti oleh para mahasiswa, saat mahasiswa mengikuti proses perkuliahan pada Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Jabatan struktural yang di embannya saat ini adalah Wakil Rektor 1 (Akademik) Universitas STEKOM Semarang.

YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-5734-68-2 (PDF)

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

Pemrograman Bahasa C#

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
JL. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id