impera

Objektorienterad programmering med C#

Kursträff 2

Grunderna i C# + Använda debuggern

Vanliga datatyper

- Vanliga inbyggda värdetyper (s. 101):
  - `byte, short, int, long, float, double, decimal, char, bool`
- Vanliga inbyggda referenstyper (s. 119):
  - `string, object`
- Två sätt att deklarera variabler:
  - `int x; // explicit typdeklaration, behöver ingen tilldelning`
  - `var x = 42; // implicit typdeklaration, måste ha tilldelning`
  - ```
    var d = 0d; // typsuffix: double
    var d = 0m; // typsuffix: decimal
    var f = 0f; // typsuffix: float
    ```
- Diskutera:
  - Varför behövs en tilldelning vid implicit typ (“var”)?
    ```

Typomvandling

- Implicit typomvandling (behöver ingen “cast”):
  - `byte → short → int → long → decimal`
  - `int → double`
  - `short → float → double`
  - `char → int`
- T.ex:

```
double d = 32; // d == 32.0
```

- Explicit typomvandling (med cast):

```
int n = (int)42.75; // n == 42
char c = (char)65; // c == 'A'
var result = (decimal)c / (decimal)n; // result == 1.5476…
```

Strängar - grunder

- Deklarera en sträng:
  - ```
    var greeting = "Hello";
    ```
  - ```
    string name;
    name = "Sailor"
    ```
- Sammanfoga strängar:
  - ```
    var output = greeting + ", " + name + "!";
    ```
- Escapesekvenser (s. 117):
  - ```
    output += "\n";
    ```
- Verbatimsträngar:
  - ```
    var result = @"Hello,
    ""World""!";
    ```
- Strängjämförelser:
  - ```
    var b = "abc" == "abc"; // b == true
    ```

Strängar - konvertera till sträng

- `ToString()` (s. 123):
  - `var s = 13.ToString(); // s == "13"`
  - `var s = (13.25).ToString("c"); // s == "13,25 kr"`
  - `var s = (13.25).ToString("N3"); // s == "13,250"`
  - `var s = (0.25).ToString("P1"); // s == "25,0 %"`

- `string.Format()`
  - `var s = string.Format("{0:c} är {1:p}", 13.5, .24);`
    `// s == "13,50 kr är 24,00 %"`

Strängar - konvertera från sträng

- `Parse` (s. 121):
  - `var n = int.Parse("20"); // n == 20`
- `TryParse` (s. 121):
  - ```
    int n;
    if (int.TryParse("20", out n)) {
       // lyckades!
    }
    ```
- `Convert`-klassen
  - `var n = Convert.ToInt32("20");`
  - `var d = Convert.ToDecimal("20.0");`
    ```

enum

```
enum AppState {
    PreInit = 0,
    Started, // 1
    Stopped  // 2
}
enum StopReason {
    IllegalAction = 10,
    UserCancelled = 20
}

var appState = AppState.Stopped;
var n = (int)appState; // n == 2
var stopReason = StopReason.UserCancelled;

if (appState == AppState.Stopped) {
   Console.WriteLine(stopReason.ToString()); // "UserCancelled"
}
```

enum - flags

```
[Flags]
enum Color {
    Red   =  1,
    Green =  2,
    Blue  =  4,
    White =  8,
    Black = 16
}
// …

app.AllowedColors = Color.Red | Color.Black;

if (app.AllowedColors.HasFlag(ValidColor.Red)) {
    // …
}
```

enum - flags

```
[Flags]
enum Color {
    Red   =  1, // 00000001
    Green =  2, // 00000010
    Blue  =  4, // 00000100
    White =  8, // 00001000
    Black = 16, // 00010000
    Grey  = ??
}
```

Nullable

- Göra så en värdetyp får innehålla null (inget värde):
  - ```
    int n = null; // Går inte!
    int? n = null;
    ```

- “Null-coalescing”:
  - ```
    int x = n ?? 10; // värdet av n, om n inte är null, annars 10
    ```
  - ```
    int x = n.Value; // För att bara hämta värdet.
                     // Kastar undantag om det är null
    ```
    ```

Kontrollstrukturer

- Alla vi känner igen från Java och JavaScript:

```
if (x >= 10) {
    // …
} else if (x < 2) {
} else {
}

switch (x) {
    case 2:
        // …
        break;
}

var x = n >= 12 ? n : 12;

for, while, do while, osv…
```

Debugger

- Livekodning:
  a. Sätta breakpoint
  b. Bevaka värde
  c. Stega fram
  d. Backa
  e. Inspektera värde

Parprogrammeringsuppgifter

1. Skapa ett nytt “Console App”-projekt
   a. Programmet ska:
      - Fråga efter två värden från användaren, X och Y
        - (Console.ReadLine)
      - Konvertera värdena till decimal
        - (decimal.Parse)
      - Multiplicera värdena med varandra
      - Skriva ut resultatet av multiplikationen med tre decimaler
      - Vänta på input innan programmet avslutas

2. Kör programmet
   a. Vad händer om man fyller i något som inte är siffror?
      - Använd decimal.TryParse istället, och skriv ut ett felmeddelande om man inte anger giltiga tal
   b. Sätt en breakpoint någonstans i ert program och kör det. Titta på vad olika variabler har för värden, och prova att stega er fram i koden.

3. Skapa en ny enum i ditt program, som heter Mode och har posterna Multiply och Percentage, samt en ny variabel av typen Mode. (Mode mode;)
   a. Ändra så att man får fylla i mode efter att man fyllt i siffrorna
      - Om man anger “percent” ska mode sättas till Mode.Percentage
      - Om man anger något annat ska mode sättas till Mode.Multiply
   b. Om mode == Mode.Percentage ska programmet istället skriva ut “Resultat: NN %” där NN är X delat på Y formaterat som procenttal.
      - Använd string.Format eller .ToString för att formatera resultatet
   c. Lägg till ytterligare ett Mode, Divide, som fungerar som Percentage, men skriver ut resultatet som ett decimaltal istället för en procentsats

```
X: 12
Y: 13
Resultat: 156,000
```

```
X: 12,5
Y: 21
Mode: percent
Resultat: 59,52 %
```

```
X: 12,5
Y: 21
Mode: divide
Resultat: 0,595
```

impera