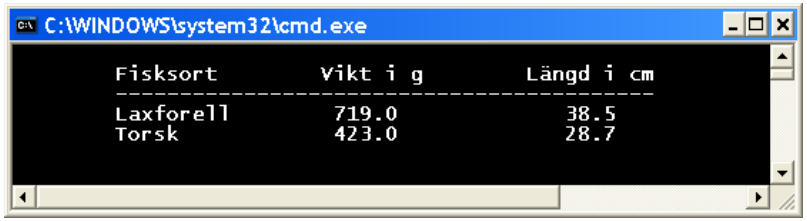
Övningar till kapitel 2

2.1 Skriv ett program som består endast av klassen **`All_in_Main`** som i sin tur innehåller endast **`Main()`**-metoden. Läs in radien **`r`** till en cirkel och beräkna samt skriv ut cirkelns area π **`r`**^{2} och dess omkrets 2π **`r`**, där $\pi = 3.14159$. Du kan använda konstanten **`Math.PI`** från C#:s klassbibliotek för π. Programmet ska inte vara objektorienterat eftersom du inte skapar några objekt, utan endast lokala variabler (radie, area, omkrets). Programmet ska inte heller vara modulariserat eller proceduralt eftersom all kod (inmatning-bearbetning- utmatning) finns i en enda metod **`Main()`** som definieras i en klass. Dessa steg ska tas i de efterföljande två övningarna. Deklarera alla variabler till **`double`**.

2.2 Modularisera programmet **`All_in_Main`** från övn 2.1 på metodnivå, dvs: Flytta bearbetningsdelen dvs beräkningen av area och omkrets ur **`Main()`** till separata metoder **`Area()`** och **`Circumference()`**, men stanna i samma klass. Döp om klassnamnet till **`Procedural`**. I **`Main()`** ska finnas kvar variabeln för radien, inmatning, utmatning och anropen av **`Area()`** och **`Circumference()`**. Förse de nya metoderna med en parameter som överför radiens värde från **`Main()`** till dem. Välj olika namn för den aktuella än för den formella parametern. Dessutom ska **`Area()`** och **`Circumference()`** returnera ett **`double`**-värde och vara statiska. För att testa, mata in **`1`** för radien. Då ska arean bli π pga π **`r`** $^2 = \pi$ och omkretsen bli 2π pga 2π **`r`** $= 2\pi$.

2.3 Modularisera programmet **`All_in_Main`** från övn 2.1 på klassnivå, dvs: Dela upp programmet i två klasser, lagrade i två separata filer. Kalla den ena klassen för **`Circle`**, den andra för **`CircleTest`**. Samla all information om *begreppet cirkel* i klassen **`Circle`**, dvs: Deklarera radien **`r`** som datamedlem samt **`Area()`** och **`Circumference()`** som metoder. Ta bort från metoderna både **`static`** och parametern för radien. Den andra klassen **`CircleTest`** ska endast innehålla metoden **`Main()`**. Skapa i den ett objekt av klassen **`Circle`**. Läs in ett värde till objektets datamedlem **`r`** och anropa samt skriv ut returvärdena till objektets metoder **`Area()`** och **`Circumference()`**. Båda klassfiler borde ligga i samma projekt.

2.4 Skriv en klass **`Fish`** som beskriver en fisk med datamedlemmarna **`sort`**, **`weight`** och **`size`**. Testa din klass i en annan klass **`FishTest`** i en separat fil som endast innehåller metoden **`Main()`** där två objekt av klassen **`Fish`** skapas. Tilldela det första objektets datamedlemmar värdena *Laxforell*, 719 (gram) och 38,5 (cm). Enheterna gram och cm behöver inte anges. Välj själv andra värden till det andra objektets datamedlemmar. Skriv ut dessa värden till konsolen i en tabell av typ:



2.5 Ta klassen **`Fish`** från övn 2.4. Förse den med en metod som beräknar priset på fisken oberoende av sort, t.ex. 7,25 kr per hekto. Lägg till även en metod som beräknar och returnerar frakten utifrån fiskens vikt och längd genom att t.ex. multiplicera en viss kostnadsfaktor, säg 0,02, med vikten, en annan, säg 0,1, med längden och addera dem. Metoderna ska returnera priset och frakten i hela kronor utan ören. Anropa metoderna från klassen **`FishTest`**:s **`Main()`**-metod för de två **`Fish`**-objekten. Lägg till nya rubriker *Pris* och *Frakt* i tabellen ovan och skriv ut deras värden till tabellens två rader.

2.6 Modifiera programmet från övn 2.5 så att datamedlemmarnas värden inte hårdkodas utan läses in. Utskriften ska skickas till konsolen och läggas till tabellen ovan. Skriv din kod så att den lätt kan generaliseras så att man kan mata in flera fisksorter med hjälp av en loop och en array av referenser till **`Fish`**-objekt som vi kommer att lära oss senare. Dessutom ska programmet kunna modifieras till att skriva ut till en tabell i en databas istället för att skriva till konsolen.

2.7 Deklarera en klass **`Triangle`** med datamedlemmarna **`side_a`**, **`side_b`**, **`side_c`**, **`height_b`** av typ **`int`** och metoderna **`Area()`**, **`Circumference()`**. Skapa i en annan klass som innehåller **`Main()`**, ett objekt av klassen **`Triangle`** och tilldela datamedlemmarna värden. Anropa metoderna och skriv ut denna triangels area och omkrets. Skapa en andra referens som pekar på samma objekt och anropa metoderna samt skriv ut deras returvärden med denna referens. Du borde få samma resultat som med den första referensen. Anropa sedan metoderna **`Area()`** och **`Circumference()`** med två anonyma objekt (utan referenser). Kolla om du får de förväntade resultaten som är baserade på objektens default-initiering. Sist, peka om **`Triangle`**-objektets första referens till **`null`** och försök att anropa metoderna med denna referens. Vad händer?

2.8 ***Kaffeautomaten (projekt)*** Du får i uppdrag att programmera en kaffeautomat. Uppdragsgivaren förväntar sig ett professionellt program som lätt kan uppdateras, om man skulle byta till en nyare automatmodell om något år. Därför anlitar man en objektorienterad programmerare. Skriv koden så generellt som möjligt så att programmet även kan modifieras för vilken varuautomat som helst, dessutom enkelt kan översättas till vilket programmeringsspråk som helst.

Programmet ska inte simulera själva automaten utan en *aktion* i automaten, dvs snarare det man *gör* med den. I händelsernas centrum ska finnas en *klass* som beskriver det som pågår i automaten *efter* att användaren stopppat in pengar i den och valt en dryck. Deklarationen till denna klass kan – i stora drag – se ut så här:

```
class Automat_action
{
   string productName;
   double price;
   double payment;
   double change;

   public Automat_action(double money,
                         char product)
   {
      switch(product)
      {
         . . .
      }
      payment = money;
      change = payment - price;
   }

   public void Change_in_coins()
   {
         . . .
   }
}
```

Konstruktorn **`Automat_action()`** ska tilldela de by default privata datamedlemmarna **`productName`** och **`price`** värden beroende på valet av dryck och skriva ut ett meddelande om inlagt belopp samt drycken som ska levereras. Detta kan kodas med hjälp av en **`switch`**-sats (ovan). Men istället för **`switch`** kan man lika bra använda nästlade **`if-else`**-satser. Skapa objekt av klassen **`Automat_action`** i en annan klass i en separat fil som endast innehåller **`Main()`**.

Börja i **`Main()`** med att skriva ut en meny över alla varor samt priserna, t.ex.:

K(affe)	8.00 kr
E(spresso)	9.50 kr
C(hoklad)	7.50 kr
L(Kaffe Latte)	9.00 kr
P(Cappuccino)	9.50 kr

Låt sedan användaren lägga in pengar. Läs in beloppet till en **`double`**-variabel. Låt användaren även välja en dryck genom att läsa in begynnelsebokstaven till

varorna ovan med en **char**-variabel. Sedan kan ett objekt av den ovan deklarerade klassen **Automat_action** skapas in inkl. anrop av konstruktorn **Automat_action()**. Vid detta anrop skickas de inlästa värdena till det inlagda beloppet och den valda varan som aktuella parametrar till **Automat_action()**. Efter att objektet skapats och datamedlemmarna initierats kan metoden **Change_in_coins()** anropas.

Komplettera programmet med att ta hand om en eventuellt felaktig eller otillräcklig betalning från användarens sida.

Metoden **Change_in_coins()** * är till för att dela upp växeln i automatens ”tillåtna” myntslag (endast 10-kr, 5-kr, 1-kr och 50-öringar) och skriva ut hur många av varje ”tillåtet” myntslag som ska ges tillbaka. Växelbeloppet måste omvandlas till detta mynt”system”. För att åstadkomma det, använd följande algoritm:

Algoritm för omvandling av ett belopp till olika myntslag

Eftersom denna algoritm endast fungerar för heltal, måste **change** som är ett belopp i kronor och ören av typ **double**, först räknas om till ett rent örebelopp av typ **int**, vilket kan göras genom att multiplicera det först med **100** och sedan avrunda resultatet till heltal:

```
int total = (int) Math.Round(change * 100);
```

I fortsättningen kommer alltså den givna växeln att stå som ett *öre*belopp i **int**-variabeln **total**. Anledningen till konverteringen till **int** i satsen ovan är att den fördefinierade metoden **Round()** som avrundar till närmaste heltal, ändå returnerar ett värde av typ **double**.

1. För att få antalet 10-kronor heltalsdivideras **total** med **1000** eftersom 10-kronor är **1000** ören:

```
int ten = total / 1000;
```

Hur många gånger ryms **1000** – eller 10-kronor – i **total**? Det antalet tilldelas till **ten**. Eller med andra ord: **1000** dras av från **total** så många gånger tills resten blivit mindre än **total**. Det antalet som tilldelas till **ten** blir antalet 10-kronor. Divisionen ovan är inte vanlig division utan heltalsdivision eftersom både **total** och **1000** är heltal. Dvs **total** divideras med **1000**, resultatet tas, resten ignoreras, t.ex. **6975/1000** ger **6**. Resten **975** ignoreras här, men används i fortsättningen. Om heltalsdivision läs på nästa sida: Modulooperatorn **%** .

---

* Myntbetalningen inkl. behandlingen av 50-öringen beror inte på nostalgi utan på internationalisering. Vi vill hålla möjligheten öppen för en överföring av programmet till andra länder där automater med myntbetalning fortfarande finns. Även ett ev. byte till Euro eller andra valutor där den halva valutaenheten finns kvar, ska vara möjligt. Omvandlingen av växelbeloppet till automatens myntsystem inkluderar en programmeringsteknisk finess som kan vara värd att lära sig. Logiken inkl. användningen av modulooperatorn ligger till grund även för en generell omvandling av det decimala talsystemet till andra system. Läs mer om det på nästa sida: Modulooperatorn % .

2. För att få antalet 5-kronor divideras just *resten* som blev kvar från punkt **1** med **500** eftersom 5-kronor är **500** ören:

```
int five = (total % 1000) / 500;
```

Här används modulooperatorn **%**. Läs om den nedan. ”Resten som blev kvar från punkt **1**” är just **(total % 1000)**. T.ex. **6975 % 1000** ger **975**. Efter att ha dragit av alla 10-kronor från **total** divideras resten med **500** för att få reda på hur många 5-kronor som finns i **total**. T.ex. **975/500** ger **1**. Resultatet av denna division ges till **five**, resten ignoreras och används i fortsättningen.

I ytterligare tre steg kan de övriga formlerna för beräkning av antalet 1-kronor (**one**), 50-öringar (**half**) och resten i öre (**rest**) skrivas, när mönstret i algoritmen (förhoppningsvis) har trätt fram:

```
int one  =  ((total % 1000) % 500) / 100;
int half = (((total % 1000) % 500) % 100) / 50;
int rest = (((total % 1000) % 500) % 100) % 50;
```

Man tar förra stegets formel, ersätter **/** med **%** och lägger till en heltalsdivision med den nya enhetens örebelopp. I det allra sista steget däremot, där man är ute efter allra sista resten i öre, måste **%** användas hela vägen. Självklart är restöreberloppet inte av praktiskt intresse när automaten inte kan spotta ut det. Mer om modulooperatorn och heltalsdivision kan du läsa här:

Modulooperatorn %

% har i C# ingenting med procenträkning att göra utan är symbolen för ett räknesätt som kallas *modulo* och innebär *resten vid heltalsdivision*. Man dividerar två heltal utan att gå vidare till decimaler, tar resten och ignorerar resultatet. T.ex. **16 % 5** ger **1**, därför att **16** heltalsdividerat med **5** ger **3**, och en rest på **1** blir kvar. Modulooperatorn **%** ignorerar **3** och returnerar resten **1**. Resten vid heltalsdivision kallas *modulo*: **9** *modulo* **2** ger **1**. Man kan uppfatta räknesättet *modulo* även som en upprepad subtraktion: Man drar av **2** från **9** så många gånger det bara går och tar det som blir kvar. Fyra gånger går det att ta bort **2** från **9**, kvar blir **1**. Därför är **9 % 2 = 1**. Generellt innebär att *räkna modulo* **a** helt enkelt att man bortser från alla multipler av heltalet **a** och behåller resten. Räknesättet modulo har många tillämpningar, speciellt vid övergång mellan två system, t.ex. mellan talsystem med olika baser som det decimala talsystemet med basen 10 och det binära med basen 2. Man kan användsa modulo för att omvandla ett antal sekunder till antal timmar, minuter och sekunder.

En rolig användning av modulo är följande litet vardagsexempel:

> Idag är det fredag, och du vill träffa din kompis om 11 dagar.
> Vilken veckodag blir det?

Om vi numrerar veckodagarna stigande från 1 med början på måndag så att fredag blir den 5:e veckodagen, får du svaret på frågan ovan genom att räkna modulo **7**:

```
(5 + 11) % 7 = 2
```

Dvs veckodagen i frågan är tisdag. Man lägger till aktuell veckodag **5**, antalet dagar **11** vilket ger **16**, men räknar modulo **7** dvs **16 % 7 = 2**, som är veckodag nr. **2**: tisdag.

I själva verket handlar det om en omvandling av det decimala talsystemet med basen 10 och siffrorna 0-9 – det talsystem vi är vana vid att räkna med – till veckodagarnas system dvs till talsystemet med basen **7** och siffrorna 0-6.

2.9 ***Master Mind (projekt)*** är ett litet spel som låter användaren gissa ett slumpmässigt genererat fyrsiffrigt heltal genom att leda spelaren med en inbyggd hjälpprocedur vars regler är beskrivna nedan. Även här gäller det att försöka hitta egna lösningar. Följande ska anses som ett förslag till lösning:

Börja med att behandla *fyr*siffriga heltal som en serie av *fyra en*siffriga tal dvs som en array av heltal med fyra element.

Skriv först en metod med huvudet **`void Create(int[] secretNo)`** som ska generera det hemliga fyrsiffriga talet och lagra det i en **`int`**-array, säg **`secretNo`**, med 4 element. Varje element i arrayen **`secretNo`** kan genereras som ett slumptal mellan 0 och 9. Dessutom ska metoden **`Create()`** kontrollera spelets regel enligt vilken alla fyra siffror måste vara olika.

Skriv sedan en metod med huvudet **`void Help(int[] guessedNo, int[] secretNo)`** som ska bearbeta spelarens gissning enligt följande regler:

För varje rätt siffra på rätt plats från vänster till höger skrivs ut ett	**`R`**
För varje rätt siffra på fel plats från vänster till höger skrivs ut ett	**`S`**
För varje fel siffra från vänster till höger skrivs ut ett mellanslag	**`?`**

Om t.ex. det hemliga talet är 4693 och spelaren gissar 7498, så erhålls hjälpen:

```
? S R ?
```

När hjälpen skriver ut **`RRRR`** har spelaren lyckats och programmet avslutas med att skriva ut ett lämpligt meddelande. Skriv ett program som tillåter flera spelomgångar.