Spelprogrammering I
Övningsuppgifter 2

Följande uppgifter är till för att öva på klasser, metoder och rekursion, samt att kunna söka i dokumentationen.

Uppgifterna ska inte lämnas in. Arbeta i par, så att ni kan diskutera era ansatser och kritisera varandras kod.

1. Fibonaccis talserie definieras som

$$F(n) = \begin{cases} 1 & \text{om } n = 0 \\ 1 & \text{om } n = 1 \\ F(n-1) + F(n-2) & \text{annars} \end{cases}$$

   Skriv funktionen `Fibonacci()`.

2. Gå tillbaka till den fakultetsfunktion du skrev tidigare. Som du såg då, blev talen snabbt för stora för att kunna hanteras av en `int`, eftersom den är begränsad till de tal som kan beskrivas med 32 binära siffror. Ett alternativ är att använda klassen `System.Numerics.BigInteger`, som kan räkna med heltal så stora som datorns hela minne tillåter. Skriv om fakultetsfunktionen till att använda `BigInteger` istället för `int`.

   Teknisk not: Du måste lägga till en *assembly reference* för att få tillgång till `System.Numerics`. I Visual Studio, välj **Project→Edit References...**. I listan med assemblies, kryssa i rutan för **System.Numerics**. Tryck på **OK**. Klart.

3. Använd din nya fakultetsfunktion för att definiera `Binom(int n, int k)`, som beräknar binomialkoefficientsfunktionen

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

4. Skriv ett program som använder sig av `Console.Write()` och `Console.ReadLine()` för att du ska kunna ha följande konversation med datorn:

```
Hej, vad heter du?
Kris
Hej Kris!
Hur gammal är du?
205
Nä, det tror jag inte på!
Hur gammal är du?
19
Det låter rimligt.
```

Tanken är alltså att programmet inte ska godkänna åldrar över 112 och under 4 och fortsätta fråga tills användaren uppger en rimlig ålder.

5. Betrakta nedanstående klassdefinition:

```
public class Demo
{
  int a, b;

  public void Method1(int a, int b)
  {
    a = 17;  b = 69;
    Method2(a, b);
  }
  public void Method2(int a, int b)
  {
    a = 42; b = 105;
  }
  public void Method3()
  {
    a = 1; b = 0;
  }
}
```

Skriv ett program som innehåller klassdefinitionen, skapar ett objekt av klassen och sedan i tur och ordning anropar `Method1()`, `Method2()` och `Method3()`. Följ med i avlusaren steg för steg och se hur (och om) värdena för `a` och `b` förändras under exekveringens gång. Vad händer?

6. Klassen `String` har inte mindre än åtta olika konstruktorer. Skapa `String`-objekt med var och en av konstruktorerna. Gör de samma sak, bara lite olika, eller har de stora skillnader?

7. Alla klasser är subklasser av `Object` och har därför en `ToString()`-metod. När du skriver ut ett värde med `Console.WriteLine()` så anropar den `ToString`-metoden för aktuell klass för att förvandla värdet till en sträng (alltså en följd tecken) som kan skrivas ut. Kontrollera detta genom att

skriva ut variabler av olika typer. Prova nu med en variabel av typen `int[]`. Vad händer?

8. Det går inte att direkt göra en underklass till `Array`, men det går att göra en *wrapper*-klass som har en vektor som instansvariabel. Skapa en dylik klass `PrintableArray` som innehåller en `int[]` och som har en `ToString()`-metod som återlämnar en strängrepresentation av elementen i vektorn.

9. `PrintableArray` behöver konstruktorer. Tänk ut några rimliga konstruktorer och implementera dem.

10. *Binärsökning* är ett effektivt sätt att hitta ett objekt i en sorterad lista `l`. Det fungerar så här: Vi har två index i listan: `low` och `high`. I första steget sätts de till första respektive sista elementet i listan. Därefter sätter vi `probe` till (det avrundade) medelvärdet av `low` och `high` och kollar värdet av `l[probe]`. Om värdet är det vi söker är vi klara; om det är mindre än det vi söker sätter vi `low` till `probe + 1`, annars sätter vi `high` till `probe - 1`. Därefter tar vi fram ett nytt värde på `probe` och fortsätter tills vi antingen hittat det sökta värdet eller `low > high`, varvid vi inser att det sökta värdet inte finns i listan.

    Utvidga nu `PrintableArray` med en metod `BinarySearch()` som söker efter ett givet element i vektorn. Vad bör metoden återlämna?

11. `System.Collections` är en abstrakt klass med många nyttiga underklasser för lagring av linjärt ordnade data. Slå upp beskrivningen av `System.Collections` i *.NET API*. Jämför `Queue` med `Stack`. Vilka metoder har de gemensamma, vilka skiljer sig? Vad betyder detta för de respektive klassernas användning?

12. Studera definitionen av `ArrayList`. Skapa sedan en underklass `SortedArrayList`, som garanterar att alla element i listan är sorterade. M a o, om man lägger till ett element med `Add()`, så ska listan fortfarande vara sorterad. Det finns några olika sätt att uppnå detta, t ex: (a) Varje gång efter att vi lagt till ett element, använder vi oss av `Sort()`, (b) vi går igenom listan element för element tills vi hittar rätt plats att placera det nya elementet med `Insert()`, (c) vi använder oss av binärsökning för att hitta rätt plats och om det sökta värdet inte finns lägger vi till det efter det närmast mindre värdet i listan.Vilken tycker du är den bästa lösningen och varför?

13. Skapa en metod `IsSorted()` som är `true` om en `SortedArrayList` faktiskt är sorterad.

14. Du kan alltid tilldela ett objekt av en underklass till en variabel som är typad som dess superklass. I det här fallet kan du göra tilldelningen `ArrayList al = new SortedArrayList();`. Men vad händer om du nu gör `al.IsSorted()`?

För att göra en typkonvertering kan du skriva om anropet som `((SortedArrayList)al).IsSorted()`, men detta kräver att `al` ursprungligen skapades som en `SortedArrayList` (eller någon underklass därav). Vad händer om du istället gör

```
ArrayList al = new ArrayList();
Console.WriteLine(((SortedArrayList)al).IsSorted());
```

?

15. Skriv ett program som skapar ett `Stack`-objekt. Lägg fyra element på stacken med `Push()` Ta bort fem med `Pop()`. Vad händer?

16. Notera att `Stack` är definierad att lagra objekt av typen `Object`. Prova att lagra i tur och ordning en `bool`, en `int` och en `string` i samma `Stack`. För att kunna hämta tillbaka objekt av okänd typ kan man använda nyckelordet `is` för att kontrollera om ett objekt är av en viss typ. Använd detta för att läsa tillbaka objekten från stacken.

17. Kan man göra en vektor som också tillåter att man lagrar vad som helst i den? Hur ser den deklarationen ut?

18. I rollspel använder man ofta notationen $m\mathrm{D}n$ för att ange kast med flera tärningar av en viss typ, m a o innebär ”3D6” kast med tre sex-sidiga tärningar. Skriv en metod som tar en sträng på det givna formatet och återlämnar ett värde som motsvarar resultatet av det angivna kastet. Tips: `String.Split()` kan vara till hjälp.

19. Utvidga ovanstående metod till att kunna hantera notationen $m_1\mathrm{D}n_1 + m_2\mathrm{D}n_2 + \ldots$.

20. Skapa en abstrakt klass `Polygon` som innehåller metoderna `Area()` och `Circumference()`. Skapa de härledda klasserna `Rectangle`, `Triangle`, `Square` och implementera metoderna för dem.

21. Lägg till en klassmetod `Area(Polygon p)` som återlämnar arean för `p`.

22. I *Monopol* består spelplanen av rutor av någon av följande typ: Tomt, järnvägsstation, statligt verk, chans/allmänning, skatt, fängelse, fri parkering, besök i fängelset. Utgå från en abstrakt klass `Tile` och skapa sedan subklasser till denna så att du kan beskriva alla de olika typerna av rutor med deras olika egenskaper.

23. Skapa en klass som dels definierar en *Monopol*-spelplan, dels håller reda på var spelarna befinner sig på denna spelplan, dels innehåller metoder för att låta spelarna förflytta sig på spelplanen.

24. Välj ut någon av ruttyperna ovan och implementera metoder för att hantera att en spelare hamnat på denna sorts ruta. Utgå från en metod `Tile.Touch()`.

25. En hel klass kan deklareras `static`. Då kan inga instanser av klassen skapas. Föreslå ett användningsområde för en sådan klass.