Instuderingsuppgifter läsvecka 6 - LÖSNINGAR

1.

Om vi använder interfacet List behöver vi inte bry oss om den konkret implementation som används, därför kan implementationen bytas ut genom att endast ändra på ett enda ställe i koden. *Depend upon abstractions, not upon concrete implementations.*

2.

Deklarationen

```
Set<String> stringSet;
```

är att föredra. Det finns två anledningar till detta:

1. vi erhåller ett kompileringsfel istället för få ett exekveringsfel om försöker lägga in något annat än en String i mängden stringSet
2. vi behöver inte göra explicita typomvandlingar när vi hämtar element ur mängden stringSet.

3.

Om man vill använda Collection-ramverkets sorteringshjälpmedel så måste man se till att alla objekt som ska sorteras implementerar gränssnittet Comparable vilket betyder att metoden compareTo () som jämför två objekt av klassen måste finnas.

```
public class Point implements Comparable{
   . . .
   public int compareTo(Point p) { . . . }
}
```

4.

```
TreeSet<Double>  mySet = new TreeSet<Double>( );
mySet.add( new Double(1.0) );
. . .
mySet.add( new Double(0.5) );
Double d = mySet.first();
```

5.

Utskrift av alla elementen i listan med användning av en iterator:

```
Iterator<String> iter = list.iterator();
while (iter.hasNext()) {
      String str = iter.next();
      System.out.println(str);
}
```

Utskrift av alla elementen i listan med användning av den förenklade **for**-satsen:

```
for (String str : list) {
   System.out.println(str);
}
```

6.

Felen inträffar vi satserna:

```
c = cigs.remove(0);   respektive   Cigarette c = cigs.get(i);
```

Metoderna remove och get returnerar typen Object medan variabeln c i båda fallen är av typen Cigarette.

Istället för att göra explicit typomvandling skall följande åtgärder göras:

```
private ArrayList< Cigarette> cigs;
public CigarettePack() {
   cigs = new ArrayList<Cigarette>();
}
```

7.

i. Ja - String implementerar Comparable

ii. Nej - Set är inte en subtyp till List

iii. Ja - ArrayList är en subtyp till List och String implementerar Comparable

iv. Nej - Dog implementerar inte Comparable

8.

a) Kompileringsfel. lep kan t.ex vara av typen List<ColorPoint> och ett element av typen Point är ingen subtyp till ColorPoint.

b) Kompileringsfel. lep kan t.ex vara av typen List<Point3D> och ett element av typen ColorPoint är ingen subtyp till Point3D.

c) Kompileringsfel. lecp kan t.ex vara av typen List<SomeType>, där SomeType **extends** ColorPoint, och element av typen ColorPoint3D är ingen subtyp till SomeType.

d) Ok. Listan lscp kan vara av typen List<ColorPoint>, List<Point> eller List<Object>. Typen ColorPoint är subtyp både till Point2D och Object.

e) Ok! **null** har ingen specifik typ och kan läggas in oberoende av vilken typ som kan lagras i listan.

f) Ok! Alla typer är sybtyp till typen Object.

g) Ok! Listan lep kan vara av typen List<Point> eller List<SomeType>, där SomeType är en subtyp till Point. Således är Point supertyp till SomeType.

h) Kompileringsfel. Listan lscp kan vara av typen List<ColorPoint>, List<Point> eller List<Object>. Typen ColorPoint är inte supertyp till Point2D och Object.

i) Ok! lecp kan t.ex vara av typen List<SomeType>, där SomeType **extends** ColorPoint, och typen ColorPoint är supertyp till SomeType.

j) Kompileringsfel. Listan lecp kan t.ex vara av typen List<SomeType>, där SomeType **extends** ColorPoint, och typen ColorPoint3D är inte supertyp till SomeType.

9.

a)

```
public static ArrayList<String> bornThisYear(Person[] people, int year) {
    ArrayList<String> result = new ArrayList<String>();
    for (int i = 0; i < people.length; i = i + 1) {
        if (people[i].getBirthYear() ==  year)
            result.add(people[i].getName());
    }
    return result;
}
```

b)

```
public static ArrayList<String> bornThisYear(ArrayList<Person> people, int year) {
    ArrayList<String> result = new ArrayList<String>();
    for (Person p : people) {
        if (p.getBirthYear() ==  year)
            result.add(p.getName());
    }
    return result;
}
```

```
public static ArrayList<String> bornThisYear(ArrayList<Person> people, int year) {
    ArrayList<String> result = new ArrayList<String>();
    Iterator<Person> p =  people.iterator();
    while (p.hasNext()) {
        Person obj = p.next();
        if (obj.getBirthYear() ==  year)
            result.add(obj.getName());
    }
    return result;
}
```

c)

```
public static void removeNames(ArrayList<Person> people,  ArrayList<String>  names) {
   Iterator<String> n =  names.iterator();
   while (n.hasNext()) {
      String name = n.next();
      Iterator<Person> p =  people.iterator();
      while (p.hasNext()) {
         if (p.next().getName() ==  name)
         p.remove();
      }
   }
}
```

10.

Om ordningen som låtarna spelas i, är av betydelse, så är en lista det rätta valet. Om däremot ordningen inte har någon betydelse, men att samma låt inte får spelas flera gånger, så är en mängd rätta valet.

11.

Inre klasser använd då en klass behöver en "hjälpklass" som inte används utanför klassen eller då klassen behöver tillgång till den omslutande klassens attribut (representation) t.ex. en iterator.

12.

En statisk inre klass har inte access till instansvariabler i den yttre klassen. Lösningen är antingen att göra klassen Inner till en icke-statisk klass, eller att göra instansvariabeln str till en klassvariabel.

```
public class Problem {
   private String str;
   private class Inner {
      private void testMethod() {
         str = "Set from Inner";
      }//testMethod
   }//Inner
}//Problem
```

eller att göra instansvariabeln str till en klassvariabel

```
public class Problem {
   private static String str;
   private static class Inner {
      private void testMethod() {
         str = "Set from Inner";
      }//testMethod
   }//Inner
}//Problem
```

13.

a) Ström som används för att lagra data på ett skivminne: FileOutputStream

b) Superklass för alla klasser som representerar utgående strömmar: OutputStream

c) Superklass för flera av de klasser som utökar funktionaliteten hos en existerande ström: FilterOutputStream

d) Innehåller metoder för att skicka olika typer av primitiva datatyper, t.ex. heltal, flyttal: DataOutputStream

14.

a)

```
import java.io.*;
public class Faulty {
   public static void main(String[] arg) throws IOException {
      BufferedReader infil = new BufferedReader(new FileReader("data.txt"));
      int totally = 0;
      int faultyBefore = 0;
      int faultyAfter = 0;
      while(true) {
         String rad = infil.readLine();
         if (rad == null)
            break;
         totally++;
         double diff = Double.parseDouble(rad);
         if (Math.abs(diff) >= 0.0001)
            faultyBefore++;
         if (Math.abs(diff) >= 0.00001)
            faultyAfter++;
      }
      infil.close();
      System.out.println("Ej accepterade axlar ökar från " + 100* (double) faultyBefore/ totally
                                    + " % till " + 100 * (double) faultyAfter/ totally + " %.");
   }//main
}//Faulty
```

b)
```
import java.io.*;
public class Faulty {
   public static void main(String[] arg) throws IOException {
      DataInputStream infil = new DataInputStream(new FileInputStream("data.bin"));
      int totally = 0;
      int faultyBefore = 0;
      int faultyAfter = 0;
      while(true) {
         try {
            double diff = infil.readDouble();
            totally++;
            if (Math.abs(diff) >= 0.0001)
               faultyBefore++;
            if (Math.abs(diff) >= 0.00001)
               faultyAfter++;
         } catch (EOFException e) { //reached end of file
               break;
         }
      }
      infil.close();
      System.out.println("Ej accepterade axlar ökar från " + 100* (double) faultyBefore/ totally
                              + " % till " + 100 * (double) faultyAfter/ totally + " %.");
   }//main
}//Faulty
```

c) Därför att binärfiler kräver mindre minnesutrymme, samt är enklare och effektivare att bearbeta.

15.

a) Klassen måste implementera interfacet Serializable.

```
import  java.io.Serializable;
public class  Person implements  Serializable {
```

b)
```
import java.io.*;
public class Splitt {
   public static void main(String[] arg) throws IOException, ClassNotFoundException {
      try {
         ObjectInputStream infile = new ObjectInputStream(new FileInputStream(arg[0]));
         ObjectOutputStream femalefile  = new ObjectOutputStream(new FileOutputStream(arg[1]));
         ObjectOutputStream malefile = new ObjectOutputStream(new FileOutputStream(arg[2]));
         Person p;
         while (true) {
            try {
               p = (Person) infile.readObject();
            }  catch (EOFException e) {
               break;
            }
            if (p.isFemale())
               femalefile.writeObject(p);
            else
               malefile.writeObject(p);
         }
         infile.close();
         femalefile.close();
         malefile.close();
      }
      catch(FileNotFoundException e) {
         System.out.println("Öppningen av filerna misslyckades!");
         System.exit(0);
      }
   }
}//Splitt
```