Tentamen i Programmeringsteknik I 2015-10-16

Skrivtid: 8:00 – 13:00
Hjälpmedel: Java-bok

Tänk på följande

- Det finns referensbok (Java) hos vakten som du får gå fram och läsa men inte ta tillbaka till bänken.
- Skriv läsligt! Använd *inte* rödpenna!
- Skriv bara på framsidan av varje papper.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer och tentamens-kod på alla papper. Skriv inte längst upp i vänstra hörnet - det går inte att läsa där efter sammanhäftning.
- Uppgift A1 skall rivas ur från tentamen och lämnas in tillsammans med övriga svar.
- Fyll i försättssidan fullständigt.
- Alla uppgifter gäller programmeringsspråket Java och programkod skall skrivas i Java. Koden skall vara läslig dvs den skall vara vettigt strukturerad och indenterad. Namn på variabler, metoder, klasser etc skall vara beskrivande men kan ändå hållas ganska korta.

  Observera att ”avdrag” i bedömningen bland annat kan göras för

  - icke-privata eller onödiga instansvariabler,
  - dålig läslighet,
  - upprepning av identisk kod och
  - underlåtenhet att utnyttja given eller egen tidigare skriven metod.
- För att bli godkänd krävs att del A är väsentligen korrekt. Om del A inte är godkänd rättas inte del B. För betyg 4 och 5 bedöms hela tentamen. För betyg 5 skall alla uppgifter vara i stort sett korrekta.

Lycka till!

Del A (obligatorisk för alla)

A1. Ringa in rätt svar och lämna in denna sida tillsammans med dina övriga svar.

a) Koden

```
int [] z = new int[5];
z[0]=14; z[1]=-1; z[2]=8;
int[] y = new int[z.length];
System.out.println(y[2]);
```

1. Ger utskriften 0
X. Ger utskriften 8
2. Medför ArrayOutOfBounds

b) Koden

```
String [] s = new String[2];
s[0]="12"; s[1]="34";
System.out.println(s[1]+s[2]);
```

1. Ger utskriften 46
X. Medför ArrayOutOfBounds
2. Ger utskriften 1234

c) Koden

```
public class Pollax {
   private int x;
   public Pollax() {
     this.x = 2;
   }
   public int doX(int a) {
      this.x = this.x+a;
      return this.x;
   }
  public static void main(String[] arg) {
      int x = -7;
      Pollax p = new Pollax();
      System.out.println(p.doX(14));
   }
}
```

1. Skriver ut 16
X. Ger kompileringsfel
2. Skriver ut 7

d) Koden

```
ArrayList <Integer> a;
a = new ArrayList<Integer>();
a.add(9); a.add(6);
a.add( a.get(0)+a.get(1) );
System.out.println(a);
```

1. Skriver ut [9, 6, 15]
X. Skriver ut [15]
2. Ger kompileringsfel

e) Koden:

```
this.z = -99;
```

1. skapar en klass med värdet -99
X. tilldelar instansvariabeln z värdet -99
2. tilldelar den lokala variabeln z värdet -99

f) Koden:

```
World w; // 1
Turtle t = new World(w); // 2
t.moveTo(20,40);  // 3
```

1. Ger garanterat fel för rad 1
X. Ger garanterat fel för rad 2
2. Ger garanterat fel för rad 3

g) Givet följande kod:

```
Measurements m1,m2;
m1 = new Measurements(20);
double m3 = Math.random();
```

1. Det skapas tre objekt
X. Det skapas två objekt
2. Det skapas ett objekt

h) Givet följande kod

```
MyClass m = new MyClass();
int h=14;
System.out.println(m.doH(h));
```

Vilket metodhuvud ger *kompileringsfel* för den sista satsen i koden?

1. `int doH(double h)`
X. `double doH(double g)`
2. `void doH(double h)`

A2. En klass `Beetle` skall representera en fiktiv skalbagge som skall ha ett namn (`name`) och en position som består av en x-koordinat (`x`) och en y-koordinat (`y`), båda är heltal. Klassen skall ha två konstruktorer och följande metoder:

- `getX` returnerar x-koordinaten
- `getY` returnerar y-koordinaten
- `getName` returnerar namnet
- `move` flyttar skalbaggen relativt den position den är i.
- `collision` testar huruvida två skalbaggar krockar och med krock menas att deras positioner är exakt lika.
- `toString` som returnerar information på lämpligt sätt (se programkörningen)

Skriv klassen enligt ovan och så den fungerar med denna mainmetod:

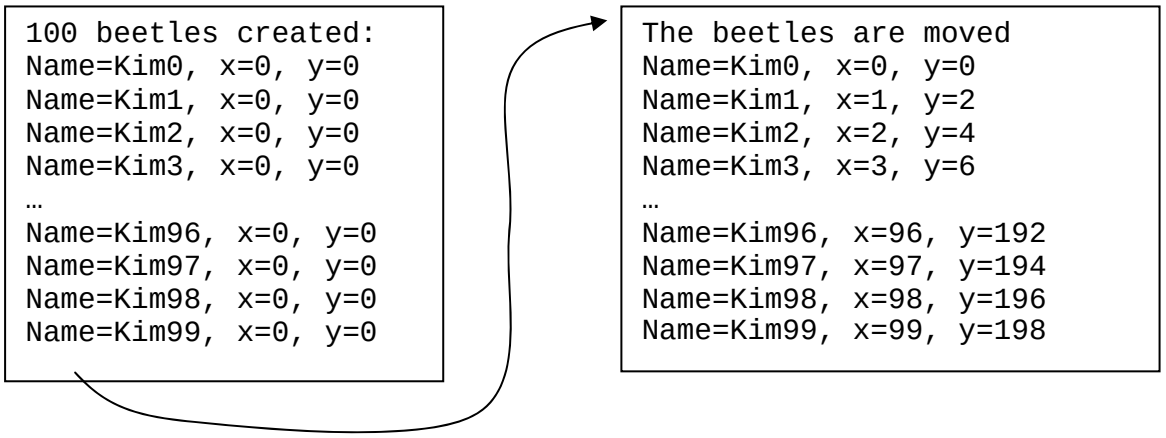
```
public static void main (String[] arg) {
    Beetle t1 = new Beetle();
    Beetle t2 = new Beetle("Torstina",2,1);
    int n=2;
    String name="Ann-Bengt"+n;
    Beetle t3 = new Beetle(name,5,6);
    System.out.println(t1);
    System.out.println(t2);
    System.out.println(t3);
    t2.move(3,5);
    System.out.println(t2.getName() + " moved to x="
                       + t2.getX() + ",y=" + t2.getY());
    System.out.println(t1.collision(t2));
    System.out.println(t1.collision(t3));
    System.out.println(t2.collision(t3));
}
```

Körning av ovan mainmetod:

```
Name=NoName, x=0, y=0
Name=Torstina, x=2, y=1
Name=Ann-Bengt2, x=5, y=6
Torstina moved to x=5,y=6
false
false
true
```

A3. Skriv en klass `TestBeetle` som enbart skall bestå av en mainmetod som skapar 100 st skalbaggar som alla placeras i origo och därefter flyttar alla en gång enligt nedanstående körning av mainmetoden (notera att vissa utskrifter är utelämnade):

```
100 beetles created:
Name=Kim0, x=0, y=0
Name=Kim1, x=0, y=0
Name=Kim2, x=0, y=0
Name=Kim3, x=0, y=0
…
Name=Kim96, x=0, y=0
Name=Kim97, x=0, y=0
Name=Kim98, x=0, y=0
Name=Kim99, x=0, y=0
```



```
The beetles are moved
Name=Kim0, x=0, y=0
Name=Kim1, x=1, y=2
Name=Kim2, x=2, y=4
Name=Kim3, x=3, y=6
…
Name=Kim96, x=96, y=192
Name=Kim97, x=97, y=194
Name=Kim98, x=98, y=196
Name=Kim99, x=99, y=198
```

B (Endast för dem som siktar på betyget 4 eller 5)

B1.
Ett lok har en identitet och en maximal dragvikt som innebär en begränsning i antalet vagnar loket orkar dra. En vagn har en vikt i antal ton. Givet är klasser som representerar ett lok (Lok) och en vagn (Vagn).

```
public class Lok {

  private String id;    // lokets identitet
  private int dragvikt; // max vikt i ton som loket klarar av att dra

  public Lok(String id, int dragvikt) {
    this.id=id;
    this.dragvikt=dragvikt;
  }

  public int getDragvikt() {
    return this.dragvikt;
  }

  public String toString() {
    return "<Lokets id="+this.id + ",dragvikt="+this.dragvikt+">";
  }

}  // Slut klassen Lok

public class Vagn {

  private int vikt;       // vagnens vikt i ton

  public Vagn(int vikt) {
    this.vikt=vikt;
  }

  public int getVikt() {
    return this.vikt;
  }

  public String toString() {
    return "Vagnens vikt " + this.vikt +" ";
  }

} // Slut klassen Vagn
```

Ett tåg består av ett lok och ett antal vagnar. Givet är ett skal till en klass Tag som representerar ett tåg.

```
import java.util.*;

public class Tag {

  private Lok loket;
  private ArrayList<Vagn> vagnar;

  // Konstruktor saknas...

  // Metoden totalvikt saknas ...

  // Metoden kopplaIn saknas ...

  // Metoden kopplaBort saknas ...

  // Metoden toString saknas...

}  // Slut klassen Tag
```

Det saknas följande i klassen `Tag`:

a) Skriv konstruktorn
b) Skriv metoden `totalvikt` som skall beräkna totalvikten (i ton) för vagnarna för tåget
c) Skriv metoden `kopplaIn` som kopplar in en vagn sist i tåget. Metoden skall returnera `true` om det gick att koppla in en vagn (om loket orkar dra alla vagnar) annars skall metoden returnera `false`.
d) Skriv metoden `kopplaBort` som kopplar bort en vagn från tåget. Antag att den vagn som skall tas bort anges med ett ordningsnummer. Den första vagnen har ordningsnumret 1. Metoden skall returnera ett lämpligt värde som anger ifall det gick bra eller inte att koppla bort vagnen.
e) Skriv metoden `toString`

De skall fungera ihop med följande mainmetod:

```
public static void main ( String[] arg ) {
    Lok loket = new Lok("RS1001",62);
    Tag X4000 = new Tag(loket);
    Vagn v;
    int antalVagnar=10;
    for (int i=1;i<=antalVagnar;i++) {
       int vikt=16+i;
       v = new Vagn(vikt);
       System.out.print("Vagn nummer " + i );
       if ( X4000.kopplaIn(v) ) {
         System.out.println(" är inkopplad");
       }
       else {
         System.out.println(" gick ej att koppla in");
         break;
       }
    } // for

    System.out.println(X4000);
    System.out.println("Vagnarnas totalvikt = " + X4000.totalvikt());
    int nr=14;
    if (X4000.kopplaBort(nr)) {
       System.out.println("Vagn nummer " + nr + " är bortkopplad")
    }
    else {
       System.out.println("Vagn nummer " + nr + " existerar ej");
    }
    nr=2;
    if (X4000.kopplaBort(nr)) {
       System.out.println("Vagn nummer " + nr + " är bortkopplad");
    }
    else {
       System.out.println("Vagn nummer " + nr + " existerar ej");
    }
    System.out.println(X4000);
}  // main
```

Som ger följande körresultat:

```
Vagn nummer 1 är inkopplad
Vagn nummer 2 är inkopplad
Vagn nummer 3 är inkopplad
Vagn nummer 4 gick ej att koppla in
[<Lokets id=RS1001,dragvikt=62>][Vagnens vikt 17 , Vagnens vikt 18 , Vagnens vikt 19 ]
Vagnarnas totalvikt = 54
Vagn nummer 14 existerar ej
Vagn nummer 2 är bortkopplad
[<Lokets id=RS1001,dragvikt=62>][Vagnens vikt 17 , Vagnens vikt 19 ]
```

B2

Givet är klassen PP

```
public class PP {
  private int[] x;
  private int stored;

  public PP (int [] q) {
    this.x = q;
    this.stored = q.length;
  }

  public void add(int a) {
    this.x[this.stored]=a;
    this.stored++;
  }

  public String toString() {
    if (this.stored==0) {
      return "[]";
    }
    String s = "[";
    for (int i=0; i<this.stored-1; i++) {
      s = s + this.x[i] + ",";
    }
    s = s + this.x[this.stored-1] + "]";
    return s;
  }

  public static void main (String[] arg) {
    int [] z = new int[3];
    z[0]=14; z[1]=-2; z[2]=5;
    PP p = new PP(z);
    System.out.println(p);
    z[1]=99;
    System.out.println(p);
    p.add(8);
    System.out.println(p);
  }
}
```

När man kör mainmetoden fås resultatet:

```
[14,-2,5]
[14,99,5]
java.lang.ArrayIndexOutOfBoundsException: 3
             at PP.add
```

Men man hade förväntat sig att resultatet skulle bli:

```
[14,-2,5]
[14,-2,5]
[14,-2,5,8]
```

Ändra lämpliga delar av klassen så att det förväntade resultatet erhålls. Mainmetoden får inte ändras.