Tentamen Programmeringsteknik I 2015-06-11

Skrivtid: 08:00 – 13:00
Hjälpmedel: Java-bok

Tänk på följande

- Det finns en 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 pin-kod (eller namn om du saknar sådan) på alla papper. Skriv inte längst upp i vänstra hörnet - det går inte att läsa där efter sammanhäftning.
- 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.

  En korrekt lösning
  - har ej publika eller onödiga instansvariabler,
  - har god läslighet,
  - upprepar ej identisk kod och
  - utnyttjar givna och tidigare skrivna metoder och klasser.
- 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 tentan. För betyg 5 skall alla uppgifter vara i stort sett korrekta.

Lycka till!

1 Del A (obligatorisk för alla)

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

a) Givet följande metodhuvud

```
double doThings(double arg);
```

Vilken av vidstående satser ger ej kompileringsfel?

[1]	`double x = doThings(5);`
x	`double y = doThings();`
2	`int z = doThings(4.7);`

b) Koden

```
String[] arr = new String[3];
arr[3] = "Hej";
System.out.println(arr[3]);
```

1	ger utskriften `Hej`
[x]	medför `ArrayOutOfBoundsException`
2	ger utskriften `null`

c) `ArrayList` är en praktisk datastruktur för att

[1]	den kan växa när man vill lägga till fler element
x	man kan använda indexoperatorn `[ ]`
2	den kan lagra primitiva datatyper utan omslagsklasser

d) Koden

```
ArrayList<String> a =
    new ArrayList<String>();
a.add("Hej");
a.add(Tjena");
System.out.println(a.get(0));
```

[1]	ger utskriften `Hej`
x	ger utskriften `Tjena`
2	ger utskriften `null`

e) En konstruktor

1	skapar en ny klass
[x]	anropas alltid när ett objekt skapas
2	har returtypen `void`

f) Koden

```
public class Scope {
  private int x = 10;
  public Scope(int x) {
       x = x + 5;
  }
  public int getX() {
       return x;
  }
  public static void main(String[] a) {
       int x = 1;
       Scope s = new Scope(x);
       System.out.println(s.getX());
  }
}
```

1	skriver ut `1`
[x]	skriver ut `10`
2	skriver ut `15`

g) Uttrycket `"" + 4 + "."` har typen

[1]	`String`
x	`double`
2	`int`

h) Givet en arraylist `a` med ett antal heltal. Vi vill iterera över alla element. Vilket av alternativen är *inte* korrekt?

1	`for (int i=0; i<a.size(); i++)`
[x]	`for (int i=0; i<=size(); i++)`
2	`for (int i=0; i!=a.size(); i++)`

A2. Skriv en klass `MoneyMachine` (en förenklad bankomat) med följande skelett

```
public class MoneyMachine {
  // Instansvariabler
    private String id;    // namn
    private int balance;  // saldo
    ...
  // Konstruktorer
    public MoneyMachine(String id, int saldo) {...}
    public MoneyMachine() {...}
  // Metoder
    public String toString() {...}
    public boolean withdraw(int amount) {...}
    ...
    ...
}
```

Alla belopp uttrycks i kronor (heltal).

Skriv

a) En instansvariabel som anger hur många uttag som gjorts från maskinen.

b) Konstruktorn med två parametrar som anger namn (`String`) och en summa pengar (`int`) maskinen initialt skall innehålla.

c) Den parameterlösa konstruktorn som ger maskinen ett standardnamn och ett nollsaldo.

d) En `toString`-metod.

e) Metoden `public boolean withdraw(int amount)` där `amount` anger önskat belopp. Om saldot är större än eller lika med önskad belopp ska metoden minska saldot med beloppet och returnera `true`. Om saldot är mindre än det önskade beloppet skall metoden lämna saldot oförändrat och returnera `false`.

f) En metod som returnerar antalet uttag som gjorts.

g) En metod som som fyller på maskinen med ett belopp som anges som parameter. Metoden skall inte returnera eller skriva ut något.

Lösning även till B2

```
import java.util.ArrayList; // För uppgift B3

public class MoneyMachine {
  private String id;
  private int balance;
  private int numberOfTransactions;
  private ArrayList<Transaction> transactions; // B3

  public MoneyMachine(String id, int amount) {
    this.id = id;
    this.balance = amount;
    this.transactions = new ArrayList<Transaction>();  // B3
    this.transactions.add(new Transaction(amount));    // B3

  }

  public MoneyMachine() {
    this("No ID", 0);
  }

  public boolean withdraw(int amount) {
     if (amount > balance)
       return false;
     else {
        transactions.add(new Transaction(-amount)); // B3
        balance -= amount;
       return true;
     }
  }

  public int getBalance() {
     return balance;
  }

  public int getNumberOfTransactions() {
    return numberOfTransactions;
  }

  public void deposit(int amount) {
    balance += amount;
    transactions.add(new Transaction(amount));  // B3
  }

  public String toString() {
     return id + ": " + balance;
  }

  public void listHistory() {   // B3
     System.out.println("Transaktionshistoria för " + id);
     for (Transaction t : transactions) {
        System.out.println(t);
     }
     System.out.println("Aktuellt saldo: " + getBalance());
  }
}
```

A3. Skriv klassen `TestMoneyMachine`. Klassen ska innehålla en `main`-metod som gör följande:

- Skapar 100 `MoneyMachine`-objekt och spara dem med hjälp av en array.
- Fyller alla med `(int)(Math.random()*1000)` kr.
- Skriver ut informationen om den första maskinen med hjälp av dess `toString`-metod.
- Försöker ta ut 500:- från varje maskin. Räknar och skriver ut hur många det lyckades för.

Exempel på utskrift:

```
Machine 0: 16
Antal lyckade uttag: 44
```

Lösning:

```
public class TestMoneyMachine {
   public static void main(String[] args)
      throws InterruptedException
   {
      MoneyMachine[] machines  = new MoneyMachine[100];
      for (int i=0; i<machines.length; i++) {
         machines[i] = new MoneyMachine("Machine " + i, (int)(Math.random()*1000));
      }
      System.out.println(machines[0].toString());
      int number = 0;
      for(MoneyMachine mm: machines) {
         if (mm.withdraw(500)) {
            number++;
         }
      }
      System.out.println("Antal lyckade uttag: " + number);
   }
}
```

Del B (Endast för dem som vill ha betyget 4 eller 5)

Observera: Denna del rättas endast om del A är godkänd!

B1. Nedanstående klass ska kunna användas av `MoneyMachine` för att hålla reda på vilka transaktioner som gjorts. Varje transaktion skall representeras med ett belopp (positivt för insättning, negativt för uttag) och en tidpunkt.

I `java.util` finns klassen `Date`. Genom att anropa den parameterlösa konstruktorn till denna erhålles ett objekt som beskriver datum och klockslag när objektet skapas.

```
import java.util.Date;

public class Transaction {

   // Instansvariabler
   ...


   // Konstruktorer
   ...

   public int getAmount() {
      return this.amount;
   }

   public Date getTime() {
      return this.time;
   }

   public String toString() {
      String result = "[";

      result += this.time.toString();
      result += " : " + this.amount + "kr]";

      return result;
   }

   public static void main(String[] args)
      throws InterruptedException {
      Transaction t1 = new Transaction();
      Thread.sleep(2000);  // För att få annan tidpunkt
      Transaction t2 = new Transaction(100);
      Thread.sleep(5000);
      Transaction t3 = new Transaction(200);

      System.out.println(t1);
      System.out.println(t2);
      System.out.println(t3);
   }
}

/* Möjlig output när main-metoden körs:

[Mon Jun 01 17:08:29 CEST 2015 : 0kr]
[Mon Jun 01 17:08:31 CEST 2015 : 100kr]
[Mon Jun 01 17:08:36 CEST 2015 : 200kr]
*/
```

a) Skriv de instansvariabler som behövs

Lösning:

```
private Date time;
private int amount;
```

b) Skriv de konstruktorer som behövs för att `main`-metoden ska fungera.

Lösning:

```
public Transaction() {
  this(0);
}

public Transaction(int amount) {
   this.time = new Date();
   this.amount = amount;
}
```

B2. Modifiera klassen `MoneyMachine` så att den med hjälp klassen `Transaction` lagrar alla transaktioner som gjorts.

Följande kod

```
MoneyMachine mm = new MoneyMachine("TestMachine", 500);
mm.listHistory();
Thread.sleep(2000);
mm.deposit(100); Thread.sleep(3000);
mm.deposit(100); Thread.sleep(1000);
mm.withdraw(150);
mm.withdraw(1000);    // Beviljas ej - ska inte med i historiken
mm.listHistory();
```

ska producera nedanstående utskrift

```
Transaktionshistoria för TestMachine
[Thu Jun 04 18:39:12 CEST 2015 : 500kr]
Aktuellt saldo: 500
Transaktionshistoria för TestMachine
[Thu Jun 04 18:39:12 CEST 2015 : 500kr]
[Thu Jun 04 18:39:14 CEST 2015 : 100kr]
[Thu Jun 04 18:39:17 CEST 2015 : 100kr]
[Thu Jun 04 18:39:18 CEST 2015 : -150kr]
Aktuellt saldo: 550
```

Ange vilka kompletteringar som behöver göras i klassen `Bankapparat`!

Lösning: Se A2

B3. Skapa en egen klass för komplexa tal.

Klassen ska lagra realdel och imaginärdel som separata flyttal. Det ska finnas tre konstruktorer, en parameterlös med nollvärden, en för att sätta enbart realdelen och en för att sätta både real- och imaginärdelen. Det ska också finnas metoder för att addera och multiplicera instanser av klassen på lämpliga sätt. Du ska dessutom skriva metoderna `toString` och `equals`, samt en `main`-metod som testar alla konstruktorer och metoder i klassen. Det är bara din `main`-metod som direkt får skriva något på skärmen eller ta någon inmatning.

Ledning: Addition av komplexa tal kan uttryckas som att addera realdelen och imaginärdelen separat. Multiplikation av komplexa tal kan uttryckas som:

Re(a * b) = Re(a) * Re(b) - Im(a) * Im(b)

Im(a * b) = Re(a) * Im(b) + Im(a) * Re(b)

```
public class Complex {
   private double re;
   private double im;

   public Complex(double re, double im) {
      this.re = re;
      this.im = im;
   }

   public Complex() {
     this(0., 0.);
   }

   public Complex(double re) {
      this(re, 0);
   }

   public String toString() {
      return "(" + this.re + ", " + this.im + ")";
   }

   public boolean equals(Complex c) {
      return this.re==c.re && this.im==c.im;
   }

   public Complex add(Complex c) {
      return new Complex(this.re + c.re,
                         this.im + c.im);
   }

   public Complex multiply(Complex b) {
      double re = this.re*b.re - this.im*b.im;
      double im = this.re*b.im + this.im*b.re;
      return new Complex(re, im);
   }

   public static void main(String[] args) {
      Complex zero = new Complex();
      Complex one  = new Complex(1);
      Complex i    = new Complex(0, 1);
      System.out.println("" + one + " + " + i + " = " + one.add(i));
      System.out.println("" + i   + " * " + i + " = " + i.multiply(i));
      System.out.println(one.equals(i));
      System.out.println(i.equals(i));
      System.out.println(one.add(i).equals(i.add(one)));
   }
}
```