Tentamen Programmeringsteknik I 2012-03-14

Skrivtid: 0800-1100
Hjälpmedel: Java-bok

Tänk på följande

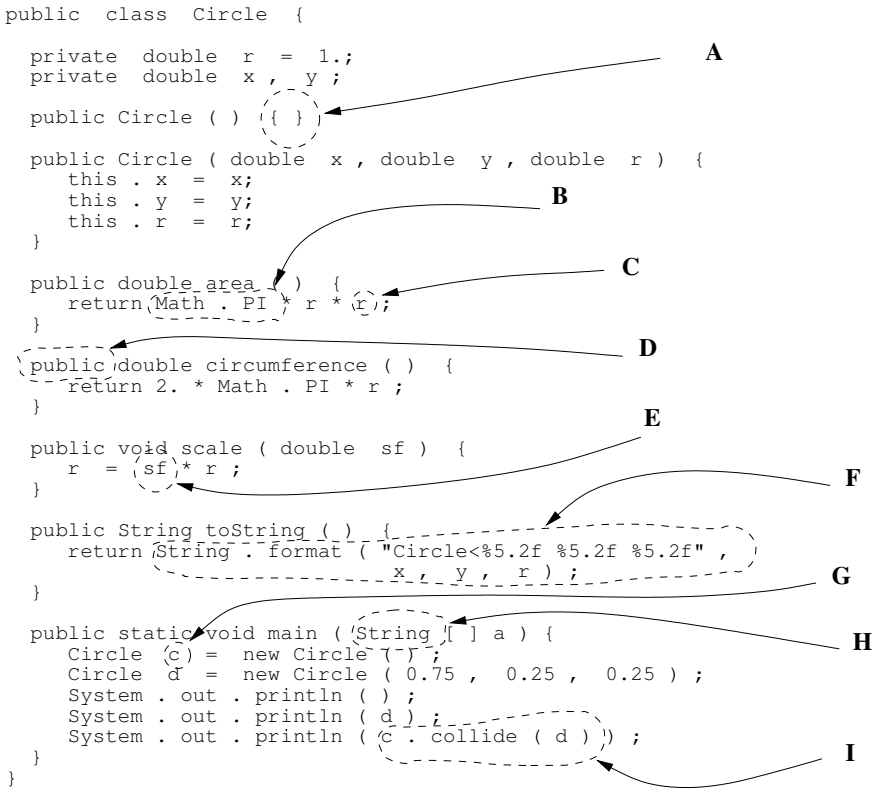
- Det finns en referensbok (Java) hos tentavakten som du får gå fram och läsa men inte ta tillbaka till bänken.
- Skriv läsligt! Använd *inte* rödpenna!
- Skriv bara på framsidan av varje papper.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer och pin-kod (eller namn om du saknar sådan) på alla papper. Skriv inte längst upp i vänstra hörnet - det går inte att läsa där efter sammanhäftning.
- Fyll i försättssidan fullständigt.
- Det är principer och idéer som är viktiga. Skriv så att du övertygar examinator om att du har förstått dessa även om detaljer kan vara felaktiga.
- Programkod 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.
- Det är totalt 30 poäng på skrivningen. Betygsgränser: 15 ger säkert 3, 21 ger säkert 4, 26 ger säkert 5.

Lycka till!

Tom

Uppgifter

1. Nedan följer ett program med ett antal inringade begrepp där varje inringning är försedd med en bokstav.




```
public  class  Circle  {

  private  double  r  =  1.;
  private  double  x ,  y ;

  public Circle ( ) { }

  public Circle ( double  x , double  y , double  r )  {
     this . x  =  x;
     this . y  =  y;
     this . r  =  r;
  }

  public double area ( )  {
     return Math . PI  * r * r;
  }

  public double circumference ( )  {
     return 2. * Math . PI * r ;
  }

  public void scale ( double  sf )  {
     r  =  sf * r ;
  }

  public String toString ( )  {
     return String . format ( "Circle<%5.2f %5.2f %5.2f" ,
                              x , y , r ) ;
  }

  public static void main ( String [ ] a ) {
     Circle  c  =  new Circle ( ) ;
     Circle  d  =  new Circle ( 0.75 ,  0.25 ,  0.25 ) ;
     System . out . println ( ) ;
     System . out . println ( d ) ;
     System . out . println ( c . collide ( d ) ) ;
  }
}
```

I tabellen nedan finns ett antal begrepp listade. Du skall para ihop begreppen vid bokstäverna med rätt rad i tabellen. Skriv den bokstav som passar för begreppet eller ett **X** om det inte finns någon inringning som passar. Svara bara med *ett* alternativ för varje rad! Samma bokstav kan dock förekomma på flera olika rader. Det kan alltså finnas bokstäver som inte har en motsvarande rad och rader som inte har en motsvarande bokstav!

aktuell parameter	1	I
formell parameter	2	E
lokal variabel	3	G
instansvariabel	4	C
klassnamn	5	H
iteration	6	X
primitiv datatyp	7	X
referensvariabel	8	G
klassmetod	9	F
omslagsklass	10	X

Du kan skriva i tabellen och lämna in sidan eller bara ge en följd av par med radnummer och bokstav. (5p)

2. Uppgifter till klassen `Circle` på föregående sida:

   a) Man kan rikta viss kritik mot den ena av de två konstruktorerna liksom mot en av metoderna? Vilka och vad är kritiken? (2p)

   Svar: *Konstruktorn med parametrar kontrollerar inte att den givna radien är ickenegativ och `scale`-metoden kontrollerar inte att skalfaktorn (`sf`) är ickenegativ*

   b) Skriv metoden `copy()` som returnerar en kopia av det egna objektet (2p)

   Svar:
   ```
   public Circle copy() {
     return new Circle(r, x, y);
   }
   ```

   c) Skriv metoden `collide(Circle c)` som returnerar `true` om det egna objektet kolliderar med cirkeln `c`, annars `false`. Två cirklar anses kollidera om avståndet mellan deras centrumpunkter är mindre än summan av deras radier. (4p)

   Svar:
   ```
   public boolean collide(Circle c) {
     double dx = x - c.x;
     double dy = y - c.y;
     double d = Math.sqrt(dx*dx + dy*dy);
     return d <= r + c.r;
   }
   ```

3. Nedan finns delar av en kod som representerar ”en värld med cirklar” dvs objekt av typen `Circle` från föregående uppgifter.

```
public class CircleWorld {

  private Circle[] theCircles;
  private int n; // Actual number of circles

  public CircleWorld(int n) { ... }

  public void add(Circle c) { ... }

  public int numberOfCollisions() { ... }
```

Testprogram:

```
  public void print() {
    String r="";
    for (int i=0; i<n; i++) {
      System.out.println(theCircles[i]);
    }
  }

  public static void main(String[] a) {
    int n = 6;
    CircleWorld cw = new CircleWorld(n);
    for (int i= 0; i<n; i++) {
      double r = Math.random()/2.;
      double x = Math.random();
      double y = Math.random();
      cw.add(new Circle(x, y, r));
    }
    cw.print();
    System.out.println("Collisions: " +
                       cw.numberOfCollisions());

    cw.add(new Circle());
  }
}
```

Output:

```
> run CircleWorld
Circle< 0,46  0,37  0,41>
Circle< 0,38  0,74  0,26>
Circle< 0,97  0,57  0,26>
Circle< 0,93  0,66  0,27>
Circle< 0,73  0,82  0,50>
Circle< 0,02  0,21  0,36>
Collisions: 9
Too many circles
>
```

a) Skriv klar konstruktorn `CircleWorld(int n)` som skall skapa en värld med *plats* för `n` cirklar (men inte lägga in några). (1p)

Svar:
```
public CircleWorld(int n) {
  this.n = 0;
  this.theCircles = new Circle[n];
}
```

b) Skriv klar metoden `add(Circle c)` som lägger till en cirkel till världen. Om den inte får plats skall ett felmeddelade ges och inget inlägg göras. (2p)

Svar:
```
public void add(Circle c) {
  if (n==theCircles.length) {
    System.out.println("Too many circles");
    return;
  } else {
    theCircles[n++] = c;
  }
}
```

c) Skriv en metod som räknar ut det totala antalet cirkelkollisioner. Metoden skall alltså titta på alla möjliga par av cirklar och se om detta par kolliderar. Vid tre cirklar kan det således vara upp till tre kollisioner (cirkel 1 och 2, cirkel 1 och 3 samt cirkel 2 och 3) vid fyra kan det kan det vara upp till sex kollisioner (1 kolliderar med 2, 3, 4 och 2 kolliderar med 3, 4 och 3 kolliderar med 4). (4p)

Svar:
```
public int numberOfCollisions() {
   int ret = 0;
   for (int i=0; i<n-1; i++) {
      for (int j=i+1; j<n; j++) {
        if (theCircles[i].collide(theCircles[j])) {
           ret++;
        }
      }
   }
   return ret;
}
```

4. En *prioritetskö* är kö där uttagsordningen i första hand styrs av elementens *prioritet* snarare än deras ankomstordning. Patientköer i sjukhusens akutmottagningar är exempel på sådana köer. Om förhållandena ändras (t ex om någon komplikation tillstöter) så kan prioriteten behöva ändras för redan köande objekt.

a) Skriv en klass `Patient`. Klassen skall representera en patient med namn och prioritet. Namnet och prioriteten skall ges som parametrar till klassens konstruktor. Förutom konstruktorn skall det finnas `get`-metoder för namn och prioritet, en metod för att ge en ny prioritet samt en `toString`-metod. Se körexemplet från testprogrammet nedan! (4p)

Svar:
```
public class Patient {
        private String name;
        private int prio;

        public Patient(String name, int prio) {
          this.name = name;
          this.prio = prio;
        }

        public String toString() {
          return "(" + name + ":" + prio + ")";
        }

        public int getPrio() {
          return prio;
        }

        public String getName() {
          return name;
        }

        public void setPrio(int p) {
          prio = p;
        }
      }
```

b) Nedan finns ett embryo till klassen `PriorityQueue`:

```
public class PriorityQueue {

   private ArrayList<Patient> q;

   public PriorityQueue() { ... }
   public boolean isEmpty() { ... }   // true if the queue is true, else false
   public void put(Patient p) { ... } // puts a patient in the queue
   public Patient get() { ... }       // gets the first patient with the highest priority

   public String toString() {
      String r ="";
      for (int i=0; i<q.size(); i++) {
         r = r + q.get(i);
         if (i<q.size()-1) {  r = r + ", "; }
      }
      return "[" + r + "]";
  }
}
```

Så här kan ett testprogram se ut:

```
public static void main(String[] args) {
  PriorityQueue pq = new PriorityQueue();
  for (int i=0; i<5; i++) {
    String name = "" + (char)('a' + i);
    int prio = (int)(Math.random()*6);
    Patient p = new Patient(name, prio);
    pq.put(p);
  }
  System.out.println("Aktuell kö: " + pq);
  System.out.println("Patient ut: " + pq.get());
  System.out.println("Patient ut: " + pq.get());
  System.out.println("Aktuell kö: " + pq);
  pq.put(new Patient("Mx X", 10));
  System.out.println("Aktuell kö: " + pq);
  while (!pq.isEmpty()) {
    System.out.println("Patient ut: " + pq.get());
  }
}
```

med utmatning så här:

```
Aktuell kö: [(a:2), (b:4), (c:1), (d:5), (e:1)]
Patient ut: (d:5)
Patient ut: (b:4)
Aktuell kö: [(a:2), (c:1), (e:1)]
Aktuell kö: [(a:2), (c:1), (e:1), (Mx X:10)]
Patient ut: (Mx X:10)
Patient ut: (a:2)
Patient ut: (c:1)
Patient ut: (e:1)
```

Skriv klar konstruktorn, metoden `isEmpty()` (enligt kommentarerna), metoden `put` (enligt kommentar och testkörning) samt metoden `get`.

Metoden `get` skall leta upp patienten med högst prioritet, ta ut den ur kön och returnera den. Om det finns flera patienter som har den högsta prioriteten skall den som stått längst tid i kön tas. Om kön är tom skall `null` returneras.

Observera att du kan få en del ledning genom att studera de delar av koden som är given inklusive testprogrammet. (6p)

Svar:

```
public PriorityQueue() {
  q = new ArrayList<Patient>();
}

public boolean isEmpty() {
  return q.size()==0;
}

public void put(Patient p) {
  q.add(p); // Put new patients in the end of the queue so
            // the order in the list reflects the arrival order
}

public Patient get() {
  if (q.size()==0) { // If the queue is empty
    return null;
  }
  int j = 0;
  for (int i = 1; i<q.size(); i++) {
    if (q.get(i).getPrio() > q.get(j).getPrio()) {
      j = i;
    }
  }
  return q.remove(j);
}
```