Tentamen Programmeringsteknik I 2012-09-01

Skrivtid: 0900-1200
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. För vart och ett av punkterna A till H skall du ange det alternativ 1 - 14 som passar. Ange bara *ett* alternativ! Om du tycker att flera alternativ passar så välj det som passar bäst! Observera att flera av alternativen A – J kan ha samma svar!

   A. `System`
   B. `println()`
   C. Det reserverade ordet **static**
   D. Det reserverade ordet **private**
   E. Konstruktor
   F. Begreppet array
   G. `toString`
   H. Det reserverade ordet **null**
   I. `Math`
   J. Heltalet 0

   1. Sådana används alltid för att skapa objekt
   2. Det behövs minst två sådana för att skapa objekt
   3. En datatyp som representerar flera värden av samma typ
   4. Ett objekt som kan användas för utskrift i ett terminalfönster
   5. Metoden finns i Java både med och utan parametrar
   6. `main`-metoden deklareras så
   7. En metod som finns alltid i alla klasser
   8. Anger att en instansvariabel får avläsas men inte ändras från andra klasser
   9. Om en lokal variabeln deklareras med t ex `int x;` så får den detta värde
   10. En datatyp som representerar flera värden som kan vara av olika datatyp
   11. En klass som kan användas utan import eller specifikation var den finns
   12. Returnerar en sträng med alla värden på ett objekts instansvariabler
   13. En variabel deklarerad så får inte ändras
   14. Om en instansvariabel deklareras med t ex `String name;` så får den detta värde
   15. Kan användas för att ange att en instansvariabel inte får användas från andra klasser
   16. Inget alternativ passar

   (5p)

2. Nedan finns en så kallad *containerklass* dvs en klass som kan lagra ett antal värden. För enkelhetens skull hanterar denna container heltalsvärden. Containerns storlek anges när den skapas. Det går att lägga in värden (`put`), se om ett visst värde finns lagrat (`contains`), ta bort ett visst värde (`remove`) och ta reda på minsta värdet (`smallest`). Det finns också en `main`-metod som demonstrerar användningen av containern.

   Containern är implementerad med en array. Från början betraktas arrayen som tom.

```
public class Container {
  private int[] box;   // Stores the numbers
  private int n;       // Number of stored numbers

  public Container(int size) { ... }               // Deluppgift a

  public String toString() {
    String s = "";
    for (int i= 0; i<n; i++)
      s = s + " " + box[i];
    return "{" + s + "}";
  }

  public boolean contains(int v) { ... }           // Deluppgift b
  public void put(int v) { ... }                   // Deluppgift c
  public void remove(int v) { ... }                // Deluppgift d

  public int smallest() { // Returns the smallest value
    if (n<0) {
      System.out.println("Empty container");
      return 0;
    } else {
      int i = 0;
      for (int j=1; j<n; j++) {
        if (box[j] < box[i])
          i = j;
      }
      return box[i];
    }
  }

  public static void main(String[] a) {
    Container c = new Container(5);
    System.out.println(c);
    c.put(2);
    System.out.println(c);
    c.put(1);
    c.put(4);
    c.put(7);
    System.out.println(c);
    c.remove(1);
    c.remove(7);
    System.out.println(c);
    c.remove(8);
    c.remove(2);
    c.remove(4);
    System.out.println(c);
    c.remove(42);
  }
}
```

Utskrifter från körning:

```
{}
{ 2}
{ 2 1 4 7}
{ 2 4}
Could not remove 8. Not found
{}
Could not remove 42. Not found
```

a) Skriv klart konstruktorn `Container(int size)` som skapar ett container-objekt med plats för `size` tal. (3p)

b) Skriv klart metoden `contains(int v)` som returnerar `true` om talet `v` finns lagrat i containern, annars `false`. (4p)

c) Skriv klart metoden `put(int v)` som lagrar `v` i containern. Om containern redan är full skall ett felmeddelande skrivas ut. Observera att värdena i containern *inte* behöver lagras i någon speciell ordning. (4p)

d) Skriv klart metoden `remove(int v)` som tar bort talet `v` ur containern. Om talet finns flera gånger skall bara *ett* av värden tas bort. Om talet inte finns skall en felutskrift ges. (5p)

3. För att simulera en trafiksituation finns följande (delvis ofullständiga) klasser:

   För bilar:

```
public class Car {
   private int bornTime;  // Tidpunkt då bilen skapas

   public Car(int born) {
      this.bornTime = born;
   }

   public String toString() {
      return "" + bornTime;
   }

   public int getBorn() {
      return bornTime;
   }
}
```

   För bilköer:

```
public class CarQueue {
  private ArrayList<Car> theQ;   // Bilarna i kön

  public CarQueue() {
    theQ = new ArrayList<Car>();
  }

  public void put(Car c) { // Ställer en bil sist i kön
    theQ.add(c);
  }

  public Car get() {  // Tar bort och returnerar första bilen i kön
    return theQ.remove(0);
  }

  public String toString() {
    String s = "";
    for (int i=0; i<theQ.size(); i++)
      s += theQ.get(i) + " ";
    return "[" + s + "]";
  }

  public boolean isEmpty() { // true om kön tom, annars false
    return theQ.size() == 0;
  }
}
```

   För trafikljus:

```
public class TrafficLight {
  private int period;        // Total period (gröntid + rödtid)
  private int greenPeriod;   // Antal steg signalen är grön
  private int time;          // Intern klocka

  public TrafficLight(int p, int g) {
    period = p;
    greenPeriod = g;
    time = 0;
  }
                               // Deluppgift a:
  public void step() { ... }      // Stegar fram signalen ett tidssteg
  public boolean isGreen() { ... }  // true om signalen är grön, annars false
  public String toString() { ... }  // "(G)" om signalen är grön, annars "(R)"
}
```

Med hjälp av dessa klasser kan man bygga klasser för representera trafiksystem. Exempel:

```
public class TrafficSystem {
  private CarQueue queue = new CarQueue();
  private TrafficLight light;
  private double intensity;
  private int time;      // Systemets klocka
  private int totalTime; // Total kötid för bilar som passerat
  private int nCars;     // Antal bilar som passerat

  public TrafficSystem(int period, int green, double intens) {
    light = new TrafficLight(period, green);
    queue = new CarQueue();
    intensity = intens;
    time = 0;
    totalTime = 0;
    nCars = 0;
  }

  public void step() {
    time++;
    light.step();
    .......
    .......                        // Deluppgift b
    .......
  }

  public void print() {
    System.out.println("\nTimestep: " + time);
    System.out.println("Current situation: " + light + queue);
    System.out.println("Number of passed cars: " + nCars);
    System.out.println("Average queue time   : " + (float)totalTime/nCars);
  }

  public static void main(String[] args) {
    TrafficSystem ts = new TrafficSystem(10, 5, 0.5);
    for (int i=1; i<20; i++) {
      ts.print();
      ts.step();
    }
  }
}
```

Klassen definierar alltså ett system med ett trafikljus och en kö. Metoden `main` gör en tidssimulering genom tidsstegning. Klassen samlar statistik på hur många bilar som kommit förbi ljuset och vad deras genomsnittliga kötid är.

En körning av programmet producerar följande output:

```
Timestep: 0
Current situation: (G)[]
Number of passed cars: 0
Average queue time   : NaN

Timestep: 1
Current situation: (G)[1 ]
Number of passed cars: 0
Average queue time   : NaN

Timestep: 2
Current situation: (G)[]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 3
Current situation: (G)[]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 4
Current situation: (G)[4 ]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 5
Current situation: (R)[4 ]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 6
Current situation: (R)[4 6 ]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 7
Current situation: (R)[4 6 7 ]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 8
Current situation: (R)[4 6 7 ]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 9
Current situation: (R)[4 6 7 9 ]
Number of passed cars: 1
Average queue time   : 1.0

Timestep: 10
Current situation: (G)[6 7 9 10 ]
Number of passed cars: 2
Average queue time   : 3.5

Timestep: 11
Current situation: (G)[7 9 10 ]
Number of passed cars: 3
Average queue time   : 4.0

Timestep: 12
Current situation: (G)[9 10 ]
Number of passed cars: 4
Average queue time   : 4.25

Timestep: 13
Current situation: (G)[10 13 ]
Number of passed cars: 5
Average queue time   : 4.2

Timestep: 14
Current situation: (G)[13 ]
Number of passed cars: 6
Average queue time   : 4.1666665

Timestep: 15
Current situation: (R)[13 ]
Number of passed cars: 6
Average queue time   : 4.1666665

Timestep: 16
Current situation: (R)[13 16 ]
Number of passed cars: 6
Average queue time   : 4.1666665

Timestep: 17
Current situation: (R)[13 16 17 ]
Number of passed cars: 6
Average queue time   : 4.1666665

Timestep: 18
Current situation: (R)[13 16 17 18 ]
Number of passed cars: 6
Average queue time   : 4.1666665
```

a) Ett trafikljus karakteriseras av en *period* som är summan av den tid (antalet tidssteg) som den är grön och den tid som den är röd (vi bortser från färgen gul) samt dess *grönperiod* som anger tiden (antalet tidssteg) som den är grön.

   Ett ljus med perioden 7 och grönperioden 3 kommer alltså vara

   `G G G R R R R G G G R R R R G ...`

   Skriv metoderna `isGreen()`, `toString()` och `step()` så att klassen fungerar enligt körexemplet. Låt signalen börja med sin grönperiod som i exemplet och testkörningen!

   Anmärkning: du behöver inte införa fler instansvariabler — det är relationerna mellan signalens interna klocka, period och grönperiod som avgör om signalen är grön eller röd. (4p)

b) Skriv klart metoden `step` i klassen `TrafficSystem`. Metoden skall utföra ett tidssteg dvs stega signalen och, om den är grön och det finns bilar i kön, ta ut den första bilen ur kön. Med en sannolikhet som anges av instansvariabeln `intensity` skall en ny bil skapas och ställas i kön.

   Metoden skall också hålla reda på hur många bilar som passerat ljuset och vad deras sammanlagda kötid är. (4p)

c) Utskriften av den genomsnittliga kötiden blir `NaN` för tidssteg 0 och 1. Vad är det och varför blir det så? (2p)