Tentamen Programmeringsteknik I 2015-03-19

Skrivtid: 14:00 – 19:00
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.
- 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.
- 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 poängavdrag 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.

Lycka till!

1 Del A (obligatorisk för alla)

För att bli godkänd krävs att alla uppgifter på denna del är väsentligen korrekta.

A1. Ange för var och en av punkterna a) - h) vilket av de tre alternativen till höger som är korrekt.

a) `int` är
- 1 en inbyggd klass i Java
- x det minsta värdet som går att representera
- 2 en primitiv typ

b) Metoden `toString` kan anropas
- 1 bara för omslagsklasser
- x för alla klasser
- 2 bara för de klasser där den är implementerad

c) Koden
```
Integer x = null;
System.out.println(x.toString());
```
- 1 ger kompileringsfel
- x ger `NullPointerException`
- 2 ger inga fel alls

d) `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

e) Antag följande metodhuvud:
```
double foo(int arg)
```
Vilken av vidstående satser är korrekt (ger ej kompileringsfel)?
- 1 `double x = foo(5);`
- x `double y = foo(7.3);`
- 2 `int z = foo(4);`

f) Koden
```
int[] a = new int[5];
a[5] = 7;
System.out.println(a[5]);
```
- 1 ger utskriften `7`
- x medför `ArrayIndexOutOfBoundsException`
- 2 ger utskriften `null`

g)
```
ArrayList<String> a =
  new ArrayList<String>();
a.add("Hej");
a.add("Tjena");
a.remove(0);
System.out.println(a.get(0));
```
- 1 ger utskriften `Hej`
- x ger utskriften `null`
- 2 ger utskriften `Tjena`

h) En konstruktor
- 1 skapar en ny klass
- x anropas alltid när ett objekt skapas
- 2 har returtypen `void`

Resten av skrivningen handlar om tre klasser som kan användas för att betygsätta tentamensskrivningar och producera statistik. Tentamensskrivningarna består av en A-del och en B-del där poängen för delarna hanteras var för sig. Klasserna finns i bilagan men delar av koden är utelämnade.

Klassen `Exam` innehåller information om poängen på uppgifterna på en tentamensskrivning. Klassen `ExamCollection` innehåller en samling av `Exam`-objekt. Klassen `ExamReader` är en hjälpklass som läser information om skrivningarna från en fil och lämnar ifrån sig `Exam`-objekt.

Klasserna `Exam` och `ExamCollection` innehåller också var sin `main`-metod inklusive utskrifter som exemplifierar.

A2. Skriv färdigt konstruktorn `Exam(String student)`.

```
this.student = student;
scoresPartA = new ArrayList<Integer>();
scoresPartB = new ArrayList<Integer>();
```

A3. Skriv färdigt metoderna `addPartA(int score)` och `addPartB(int score)` som lägger in poängen för en deluppgift i del A respektive del B.

```
public void addPartA(int score) {
  scoresPartA.add(score);
}
```

A4. Skriv färdigt metoden `String toStringPartA()` som returnerar en strängrepresentation av del A. Se körexempel! Du behöver inte skriva metoden `String toStringPartB()`.

```
public String toStringPartA() {
  return scoresPartA.toString();
}
```

A5. Skriv metoden `int sumPartB()` som beräknar och returnerar summan av poängen på del B.

```
public int sumPartB() {
   int sum = 0;
   for (int score : scoresPartB) {
      sum += score;
   }
   return sum;
}
```

A6. Skriv metoden `public boolean passed(int minScore, int limit3)` som returnerar `true` om skrivningen är godkänd, annars `false`. För att en skrivning skall vara godkänd krävs dels att alla uppgifter på del A har minst `minScore` poäng *och* dels att summan av poängen på del A är minst `limit3`.

Se exempelkörningen!

```
public boolean passed(int minScore, int limit3) {
   int sum = 0;
   for (int score : scoresPartA) {
      sum += score;
      if (score < minScore) {
         return false;
      }
   }
   return sum >= limit3;
}
```

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

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

Betyg 4 kräver att minst hälften och betyg 5 att alla uppgifter på denna del är väsentligen korrekt lösta.

B1. Skriv färdigt metoden

```
int getGrade(int minScore, int limit3, int limit4, int limit5)
```

i klassen `Exam` som returnerar skrivningens betyg. Om skrivningen inte uppfyller kraven för godkänt enligt uppgift A6 ovan ska betyget 0 (för U) returneras oavsett resultatet på del B.

Om den är godkänd sätts betyget beroende på poängsumman på del B enligt följande

- summan minst `limit5` ger betyget 5,
- summan minst `limit4` men mindre är `limit5` ger betyget 4 och
- summan mindre än `limit4` ger betyget 3.

```
public int getGrade(int minScore, int limit3, int limit4, int limit5) {
   if (!this.passed(minScore, limit3)) {
      return 0;
   }

  int theGrade = 3;

  int sum = this.sumPartB();
  if (sum >= limit5)
     return 5;
  else if (sum >= limit4)
     return 4;
  else
     return 3;
}
```

B2. Skriv färdigt konstruktorn i klassen `ExamCollection`. Konstruktorn tar emot ett filnamn som parameter och använder den bifogade klassen `ExamReader` för att hämta `Exam`-objekt skapade utifrån innehållet i filen.

```
public ExamCollection(String filename) throws IOException {
   theCollection = new ArrayList<Exam>();
   ExamReader examReader = new ExamReader(filename);
   Exam  exam = examReader.next();
   while (exam!=null) {
      theCollection.add(exam);
      exam = examReader.next();
  }
}
```

B3. Skriv färdigt metoden `printStatistics`. Metoden ska beräkna och skriva ut betygsfördelningen dvs antalet U, antalet 3-or, antalet 4-or och antalet 5-or. Se körexempel!

```
  public void printStatistics(int limit3, int limit4, int limit5) {
      int[] distribution = new int[6];
      for (Exam exam : theCollection) {
         int grade = exam.getGrade(limit3, limit4, limit5);
         distribution[grade]++;
      }
      /////////////// Nedanstående del given ///////////////
      System.out.println("\nStatistik");
      System.out.format("U:%2d,  3:%2d,  4:%2d,  5:%2d\n",
                        distribution[0], distribution[3],
                        distribution[4], distribution[5]);
   }
```

B4. Metoden `getGrade` tar i vårt exempel fyra parametrar. Egentligen skulle man vilja representera gränserna tillsammans i ett objekt, eftersom de hör ihop. Vi kallar klassen för sådana objekt `Limit`, så att signaturerna för de två metoderna `getGrade` och `passed` i `Exam` blir `int getGrade(Limit limit)` och `boolean passed(Limit limit)` och på motsvarade sätt för metoderna i `ExamCollection`.

Skriv ett kodskelett till klassen `Limit`. Det ska i den framgå vilka instansvariabler, konstruktorer och metoder som finns. Minimipoängen (`minScore`) och de tre betygsgränserna ska kunna lagras och användas. Tänk på synligheten för de olika delarna i koden du skriver.

Skriv inget innehåll i konstruktorer och metoder, bara deras huvuden!

```
public class Limit {
  private int minScore, limit3, limit4, limit5;

  public Limit(int minScore, int limit3, int limit4, int limit5)

  public String toString()

  public int getMinScore()

  public int getLimit3()

  public int getLimit4()

  public int getLimit5()

}
```