Anmälningskod:

Tentamen Programmeringsteknik I 2016-03-17

Skrivtid: 1400–1900

Tänk på följande

- Skriv läsligt. Använd *inte* rödpenna.
- Skriv bara på framsidan av varje papper.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer (gäller B-delen) och din kod *överst i högra hörnet* på alla papper
- Fyll i försättssidan fullständigt.
- Såvida inget annat anges, både får och ska man bygga på lösningar till föregående uppgifter även om dessa inte har lösts.
- Det är tillåtet att införa hjälpmetoder och hjälpklasser. Uttrycket ”skriv en metod som” skall alltså *inte* tolkas så att lösningen inte får struktureras med hjälp av flera metoder.
- Du behöver inte skriva `import`-satser för klasserna `Scanner`, `ArrayList`, `Locale` och inte heller för klasser i `java.io`.
- 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 betyget påverkas negativt av
  - icke-privata eller onödiga instansvariabler,
  - dålig läslighet,
  - upprepning av identisk kod,
  - underlåtenhet att utnyttja given eller egen tidigare skriven metod.

Skrivningen består av två delar. Lösningarna till uppgifterna på del A1 – A7 ska skrivas in i de tomma rutorna och lämnas in. Lösningarna till uppgift A8 samt uppgifterna på B-delen skrivs på lösa papper.

För att bli godkänd (betyg 3) krävs att hela A-delen är i stort sett rätt löst.

För betyget 4 krävs dessutom att minst hälften av uppgifterna på B-delen och betyg 5 att alla uppgifterna på B-delen är i stort sett rätt lösta.

Observera att B-delen inte rättas om inte A-delen är godkänd.

Lycka till!

Anmälningskod:

Del A (obligatorisk för alla)

A1. Koden följer garanterat namnkonventionerna dvs klasser börjar med stor bokstav, metoder och variabler med liten.

Ringa in rätt svar och lämna in tillsammans med dina övriga svar! Svara bara med *ett* alternativ per fråga.

a) Givet koden
```
Turtle t;
```
Vilket påstående är *felaktigt*?

1) `t` är en referenstyp
2) `t` är en objekttyp
3) `t` är ett objekt

b) I uttrycket
```
t.move(30)
```
är `30`

1) en formell parameter
2) en aktuell parameter
3) en referens?

c) I uttrycket
```
MyClass.sort(a)
```

1) måste `sort` vara en klassmetod
2) måste `sort` vara en instansmetod
3) kan man inte avgöra om `sort` är en instans- eller klassmetod

d) `Math` är *inte*

1) en typ
2) en klass
3) en referens

e) Vilken returtyp bör metoden nedan ha dvs vad ska ... bytas mot?
```
public ... giveBirth(Person dad) {
  String n = this.n + "-"+ dad.n;
  String address = this.address.trim();
  int age = 0;
  return new Person(n, address, age);
}
```

1) `String`
2) `int`
3) `Person`
4) `void`

f) Hur många objekt skapas av koden nedan? (Du ska bara räkna de objekt som man ser att koden skapar)
```
Turtle[] family = new Turtle[5];
World earth = new World(400, 400);
for(int i=0; i<3; i++) {
  family[i] = new Turtle(earth);
}
```

1) 3 objekt
2) 4 objekt
3) 5 objekt
4) 6 objekt
5) 7 objekt

Anmälningskod:

g) En instansvariabel

1) måste deklareras utanför konstruktorerna
2) måste deklareras i en konstruktor
3) måste ges ett värde i en konstruktor

h) Vilket av vidstående påstående om metoden `calculate` *kan* vara sant när man ser nedanstående sats?

```
int x = calculate(4);
```

1) Den kan ha en `String` som formell parameter
2) Den kan ha `double` som returtyp
3) Den kan ha `double` som formell parameter

i) Vilket uttalande om koden nedan är sant?

```
if ("Nisse".equals(friend)) {
  System.out.println("My friend"); }
```

1) `equals` som anropas finns i klassen `Friend`
2) `equals` som anropas finns i klassen `String`
3) `equals` som anropas finns i klassen `ArrayList`
4) Koden är felaktig

j) Uttrycket `(double)(1 + 3/2) + 3`

1) har värdet `5.0`
2) har värdet `5`
3) har värdet `5.5`

Resten av skrivningen handlar om ett bokningssystem för säten i ett flygplan. Sätena är organiserade i rader och varje rad innehåller ett antal säten.

En rad representeras av klassen `Row` som innehåller en array av strängar. Varje element i arrayen innehåller antingen en passagerares namn eller **null** om platsen är obokad.

Klassen `Cabin` innehåller en array av `Row`-objekt. Klassen har metoder för att boka platser, visa bokningar och passagerarlistor samt för att spara och läsa från fil.

En utskrift av en kabin med 8 rader och 5 stolar på varje rad kan se ut så här:

```
    0  1  2  3  4
   --------------
0  [ B  B  .  .  B ]
1  [ .  .  .  .  B ]
2  [ .  B  .  B  B ]
3  [ B  .  B  .  . ]
4  [ .  .  B  .  B ]
5  [ .  .  .  .  . ]
6  [ .  .  .  .  . ]
7  [ .  B  B  .  B ]
```

Bokstaven `B` anger att platsen är bokad och punkten att den är ledig.

Såväl säten som rader numreras (för programmerarens bekvämlighet) från 0 och uppåt.

Båda klasserna finns i bilagan men vissa delar av koden är utelämnade.

Uppgifterna i A-delen handlar bara om klassen `Row`

Anmälningskod:

A2. Skriv klart konstruktorn som tar emot antalet platser som ska finnas på raden

```
public Row(int size) {

    this.seats = new String[size];

}
```

A3. Skriv en metod `get` som returnerar namnet den som är bokad på en viss plats. Platsen ska ges som parameter. Platserna är numrerade från 0 och uppåt. Om ingen är inbokad på platsen ska **null** returneras. Ingen kontroll av att parametern har ett legalt värde behöver göras.

```
public String get(int seat) {
    return seats[seat];
}
```

A4. Skriv klart metoden `printPassengers()` som skriver ut namen på passagerarna skrivet en per utskriftsrad. Obokade platser ska inte skrivas. Se körexemplet!

```
public void printPassengers() {
    for (int i=0; i<seats.length; i++) {

        String p = seats[i];
        if (p != null) {
            System.out.println(p);
        }

    }
}
```

Anmälningskod:

A5. Metoden `find(String name)` i klassen `Row` letar efter en passagerare med angivet namn på denna rad. Om namnet finns returneras platsnumret, annars -1.

Skriv klart metoden!

```
public int find(String name) {
```

```
for (int i=0; i<seats.length; i++) {
    if (name.equals(seats[i])) {
        return i;
    }
}
return -1;
```

```
}
```

A6. Metoden `toString()` i klassen `Row` ska returnera en sträng som visar vilka platser som är upptagna/lediga på raden. En ledig plats markeras med strängen `" . "` och en bokad men strängen `" B "`. Båda dessa strängar är tre tecken långa och har `B` respektive `.` som sitt mittersta tecken.

Exempel: Raden

```
[ B  B  .  .  B  .  B ]
```

beskriver en rad med 7 säten varav plats 0, 1, 4 och 6 är bokade.

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

```
for (String s: seats) {
  if (s==null) {
    result += " . ";
  } else {
    result += " B "
  }
}
```

```
   return "[" + result + "]";
}
```

Anmälningskod:

A7. Metoden `book` används för att boka en plats på raden. Metoden tar emot ett platsnummer och ett passagerarnamn som parametrar. Den kontrollerar att platsnumret är giltigt samt att platsen är obokad. Om detta är uppfyllt så läggs passagerarnamnet in på platsen och **true** returneras. Om något av villkoren inte är uppfyllt görs ingen bokning och **false** returneras.

Fyll i den kod som fattas!

```
public boolean book(int seat, String passenger) {

                    seat<0 || seat >= seats.length
  if ( [                                                ] ) {

        System.out.println("*** Illegal seat number :"+ seat);
        return false;

                    seats[seat]!=null
  } else if ( [                                         ] ) {

        System.out.println("*** Seat already booked")
        return false;

  } else {

       seats[seat] = passenger;
       [                                                ] ) {

       return true;
  }
}
```

A8. Skriv på separat papper en klass `Passenger` som representerar en passagerare med namn och plats dvs rad- och sätesnummer. Förse klassen med en konstruktor som tar emot dessa tre värden. Skriv också en `toString`-metod som returnerar namn och platsinformation samt en `public int compareTo(Passenger p)`. Metoden `compareTo` ska jämföra passagerarnas namn och, på vanligt sätt, returnera något negativt, 0 eller något positivt beroende på ordningen mellan namnen.

Skriv också en en `main`-metod som skapar två passagerar-objekt, skriver ut dem med hjälp av `toString`-metoden och visar vad `compareTo`-metoden returnerar med dessa två objekt.

```
public class Passenger {
   private String name;
   private int rowNumber;
   private int seatNumber;

   public Passenger(String name, int rowNumber, int seatNumber) {
      this.name = name;
      this.rowNumber = rowNumber;
      this.seatNumber = seatNumber;
   }

   public String toString() {
      return String.format("%12s in row %2d at seat %2d",
                           name, rowNumber, seatNumber);
   }

   public int compareTo(Passenger p) {
      return name.compareTo(p.name);
   }

   public static void main(String[] args) {
     Passenger p1 = new Passenger("Lisa", 1, 3);
     Passenger p2 = new Passenger("Olle", 0, 4);
     System.out.println("p1: " + p1);
     System.out.println("p2: " + p2);
     System.out.println("p1.compareTo(p2) : " + p1.compareTo(p2));
     System.out.println("p2.compareTo(p1) : " + p2.compareTo(p1));
   }
}
```

Del B (för betyg 4 och 5)

Svaren skrivs på lösa papper med ny uppgift på nytt papper.

B1. Metoden `public static Row load(Scanner scan)` i klassen `Row` läser radinformation med det `Scanner`-objekt som ges som parameter. Informationen är lagrad av `save`-metoden dvs först en textrad med ett heltal som anger antalet säten på raden och därefter en textrad per säte som innehåller ett namn om sätet är bokat, annars texten `none`.

Så här kan informationen om en rad med 5 säten se ut:

```
5
Kalle
none
Olle
none
Zaphod
```

Metoden ska skapa ett `Row`-objekt utifrån den lästa informationen.

Skriv metoden!

```
public static Row load(Scanner fscan) {
    int size = fscan.nextInt();
    fscan.nextLine();
    Row row = new Row(size);
    for (int i= 0; i<size; i++) {
       String line = fscan.nextLine();
       if (!line.equals("none" )) {
          row.seats[i] = line;
       }
    }
    return row;
}
```

B2. Klassen `Cabin` representerar alla platser i kabinen genom att ha en *array* av `Row`-objekt. Klassen har följande instansvariabler:

```
private Row[] theRows;
private String flightId;
```

Skriv konstruktorn

```
Cabin(String flightId, int numberOfRows, int rowLength)
```

som skapar `Cabin`-objekt med angivna värden på instanvariablerna. Konstruktorn ska skapa såväl arrayen med rader som själva radobjekten.

```
public Cabin(String flightId, int numberOfRows, int rowLength) {
  this.flightId = flightId;
  theRows = new Row[numberOfRows];
  for (int i = 0; i<numberOfRows; i++) {
    theRows[i] = new Row(rowLength);
  }
}
```

B3. Skriv en metod `ArrayList<Passenger> passengerList()` som skapar och returnerar en arraylist med `Passenger`-objekt. Metoden ska alltså gå igenom alla platser i kabinen och, för varje bokad plats, skapa ett `Passenger`-objekt och lägga in det i en arraylist som sedan ska returneras.

```
public ArrayList<Passenger> passengerList() {
   ArrayList<Passenger>  res = new ArrayList<Passenger>();
   for (int i=0; i<theRows.length; i++ ) {
      for (int j=0; j<theRows[i].rowLength(); j++) {
         if (theRows[i].get(j)!=null)
            res.add(new Passenger(theRows[i].get(j), i, j));
      }
   }
   return res;
}
```

B4. Skriv en metod

```
public static ArrayList<Passenger> sort(ArrayList<Passenger> list)
```

som tar emot en arraylista med passagerare och skapar och returnerar en ny arraylista som innehåller samma passagerare men sorterade på namn.

```
public static ArrayList<Passenger> sort(ArrayList<Passenger> list) {
   ArrayList<Passenger>  result = new ArrayList<Passenger>();
   for (Passenger p : list) {
      int i=0;
      while (i<result.size() && p.getName().compareTo(result.get(i).getName())>0) {
         i++;
      }
      result.add(i, p);
   }
   return result;
}
```

B5. Skriv en metod `printPassengers()` som skriver ut en lista över planets passagerare i bokstavsordning med en passagerare per utskriftsrad.

Exempel på utskrift:

```
Anna      in row  0 at seat  2
Tom       in row  2 at seat  3
Trillian  in row  2 at seat  2
Zaphod    in row  0 at seat  3
```

```
public void printPassengers() {
   for(Passenger p : sort(passengerList())) {
      System.out.println(p);
   }
}
```