*Anmälningskod:*

**Tentamen Programmeringsteknik I 2017-10-23**

Skrivtid: 0800–1300

**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 på föregående uppgifter även om dessa inte har lösts.
- På B-delen är det 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 A-delen ska skrivas in i de tomma rutorna och den delen ska lämnas in. Rutorna är tilltagna i storlek så att de ska rymma svaren.

Lösningarna till uppgifterna på B-delen skrivs på lösa papper.

För att bli godkänd (betyg 3) krävs att minst ca 75% av 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. Vi bedömning av betyg 4 och 5 tas också hänsyn till kvalitén på lösningarna i A-delen.

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.

Ringa in rätta svar och lämna in tillsammans med dina övriga svar!

a) Hur många objekt skapas (synbart från den givna koden) av följande kod?

```
World w = new World();
Turtle t1, t2, t3;
t1 = new Turtle(w);
t2 = t1;
t3 = t2;
```

1) 0
2) 1
③) 2
4) 3
5) 4
6) Obestämbart antal

b) Vilken eller vilka returtyper kan nedanstående metod ha?

```
public typ? positive(int value) {
  return value > 0;
}
```

1) `int`
②) `boolean`
3) `void`
4) `String`
5) Det blir kompileringsfel oavsett.

c) Antag att klassen `PhoneBook` har en instansvariabel `ArrayList<PhoneBookEntry>`. På vilket eller vilka sätt kan en metod i klassen `PhoneBook` med huvudet `public PhoneBook find(String partialName)` anropas, givet att det finns en referens till ett `PhoneBook`-objekt `pb`?

1) `PhoneBookEntry pb2 = pb.find("Eva");`
2) `String pb2 = pb.find("Eva");`
③) `PhoneBook pb2 = pb.find("Eva");`
4) `ArrayList pb2 = pb.find("Eva");`
5) `PhoneBookEntry pb2 = pb.find("Eva");`

d) Vad blir resultatet av nedanstående kod?

```
int a = 11;
int b = 13;
double c = a;
a = b;
b = (int)c;
System.out.println(a + ", " + b);
```

1) Kompileringsfel
2) Exekveringsfel
3) Utskriften `11, 11`
4) Utskriften `11, 13`
⑤) Utskriften `13, 11`
6) Utskriften `13, 13`

e) En serie heltal skall lagras i en `ArrayList`. Hur deklareras den så att det inte blir kompileringsfel och rätt sorts data kan lagras.

1) `ArrayList<int> numbers;`
②) `ArrayList<Integer> numbers;`
3) `ArrayList<double> numbers;`
4) `String numbers;`
5) `int[] numbers;`

f) Satsen

```
double x = (int)((5/2) * 2.0);
```

resulterar i

1) Kompileringsfel
2) Exekveringsfel
3) `x` får värdet `4`
④) `x` får värdet `4.0`
5) `x` får värdet `5`
6) `x` får värdet `5.0`

## Snake - problembeskrivning och illustration

Uppgifterna i denna tentamen består av att skriva färdigt ett litet Snake-spel som för enkelhets skull här ska styras via text-input. Spelet har en enkel mekanik med några få regler.

- Spelet uppdateras i diskreta/fasta tidssteg.
- Maskens huvud (representeras av `@`) har en riktning. Varje tidssteg flyttar sig huvudet i den riktningen. Maskens kropp följer efter.
- När masken tar en frukt (representerad av `X`) förlängs masken med en bit.
- Om masken kolliderar med sig själv eller med kanterna avslutas spelet.

Exempel på några exekverade tidssteg med tillhörande användarinput:

```
---------@OO
----------O
----------O
---------X-O
----------O
--------OOOO
--------O---
---OOOOOO---
(-1: Left, 0: Forward, 1: Right):
-1
```

```
---------OOO
---------O-O
---------O-O
---------@-O
----------O
--------OOOO
--------O---
-----OOOO-X-
(-1: Left, 0: Forward, 1: Right):
1
```

```
---------OOO
---------@-O
----------O
---------X-O
----------O
--------OOOO
--------O---
----OOOOO---
(-1: Left, 0: Forward, 1: Right):
0
```

```
---------OOO
---------O-O
---------O-O
--------@O-O
----------O
--------OOOO
--------O---
------OOO-X-
(-1: Left, 0: Forward, 1: Right):
0
```

```
---------OOO
---------O-O
---------@-O
---------X-O
----------O
--------OOOO
--------O---
-----OOOO---
(-1: Left, 0: Forward, 1: Right):
0
```

```
---------OOO
---------O-O
---------O-O
-------@OO-O
----------O
--------OOOO
--------O---
-------OO-X-
(-1: Left, 0: Forward, 1: Right):
```

När spelet är slut meddelas vilken poäng spelaren uppnådde (antal tagna frukter/maskens längd - 1):

```
Game Over!
Score: 18
```

*Anmälningskod:*

Här följer ett antal uppgifter på klassen `SnakePart` som representerar en bit av masken i ett Snake-spel. Denna klass ska lagra två heltal (int) $x$ och $y$, för att representera en position i ett 2d-rutnät, och ett `Direction` objekt (se given klass) *direction* för att representera en riktning, som instansvariabler.

`x` representeras av tal `0..bredd` och `y` representeras av tal `0..höjd`.

A2. Skriv deklarationen av de tre instansvariablerna.

```
private int x;
private int y;
private Direction direction;
```

A3. Skriv en konstruktor som tar emot och sätter värden på de tre instansvariablerna. Parametrar ska vara av typ `int` och `Direction`.

```
public SnakePart(int x, int y, Direction direction) {
    this.x = x;
    this.y = y;
    this.direction = direction;
```

A4. Skriv en parameterlös konstruktor som sätter positionen till (0, 0) och riktiningen till höger. Tips: Se `Direction`–klassens `main`–metod för exempel på hur man skapar en viss riktning.

```
public SnakePart() {
    this(0, 0, Direction.createRight());
}
```

A5. Skriv en `toString`-metod som returnerar en strängrepresentation: `(3, 4: Right)`. Den ska ge en sträng på formen `(x, y: direction)`.

```
public String toString() {
 return "("( + this.x + ", "+ this.y + ": "+ this.direction + ")";
}
```

A6. Skriv en `main`-metod som demonstrerar användningen av konstruktorerna och `toString`-metoden.

```
 public static void main(String[] args) {
    SnakePart p1 = new SnakePart(5, 2, Direction.createRight());
    SnakePart p2 = new SnakePart();
    System.out.println("p1: " + p1.toString());
    System.out.println("p2: " + p2.toString());
}
```

*Anmälningskod:*

A7. Skriv metoden `testPosition` i klassen SnakePart med metodhuvud
`public boolean testPosition(int tx, int ty).`
Metoden skall returnera `true` om `tx` och `ty` överrensstämmer med aktuell position, med andra ord om en kollision har inträffat.

```
public boolean testPosition(int tx, int ty) {
    return tx == this.x && ty == this.y;
}
```

A8. Skriv metoden `move` i klassen SnakePart med metodhuvud
`public SnakePart move(int relativeDirection).`
Metoden skall ta en relativ riktning i form av ett heltal som parameter och generera ett nytt objekt, också av typen SnakePart, förflyttad ett steg enligt den nya riktningen.

Obs: Det ursprungliga objektet skall inte ändras.
Tips: Se metoderna `moveX/moveY/turn` i klassen `Direction`.

Exempel: (Givet att `sp` är en referens till ett objekt av typen `SnakePart` som ger strängen `"(3, 4: Down)"` när man anropar dess `toString`-metod.

```
System.out.println(sp.move(0).toString()) // Ger utskrift (3, 5: Down)
System.out.println(sp.move(-1).toString()) // Ger utskrift (4, 4: Right)
System.out.println(sp.move(1).toString()) // Ger utskrift (2, 4: Left)
```

```
public SnakePart move(int relativeDirection) {
    Direction newDirection = this.direction.turn(relativeDirection);
    return new SnakePart(this.x + newDirection.moveX(),
        this.y + newDirection.moveY(),
        newDirection);
}
```

A9. Skriv metoden `isInside` i klassen SnakePart med metodhuvud
`public boolean isInside(int gameWidth, int gameHeight).`
Metoden skall ta returnera `true` om objektet är inuti spelplanen och `false` annars. Giltiga positioner är i intervallen `0 <= x < gameWidth` och `0 <= y < gameHeight`.

```
public boolean isInside(int gameWidth, int gameHeight) {
    return x >= 0 && y >= 0 && x < gameWidth && y < gameHeight;
}
```

*Anmälningskod:*

A10. Skriv en klass `TestSnakePart` med enbart en `main`–metod som ska göra följande:

- Deklarera och skapa en `SnakePart`-array med 10 platser.
- Skriv en loop som skapar de 10 objekten, där första objektet har position `(0, 0)`, det andra `(1, 1)`, ..., `(9, 9)`. Objekten ska ha slumpvisa riktningar. Giltiga riktningar är {`0, 1, 2, 3`}. Se klassen `Direction`.
- Efter att alla objekten har skapats: Skriv en loop som går igenom alla array-element och skriv ut dem, med hjälp av `System.out.println` och `toString`.

```
public class SnakePartTest {
    public static void main(String[] args) {
        SnakePart[] sp = SnakePart[10];
        for(int i = 0; i < 10; ++i) {
            sp[i] = new SnakePart(i, i, new Direction((int)(Math.random() * 4)));
        }
        for(int i = 0; i < 10; ++i) {
            System.out.println(sp[i].toString());
        }
    }
}
```

## Del B (för betyg 4 och 5)

*Svaren skrivs på lösa papper med ny uppgift på nytt papper.*
Alla uppgifter handlar om klassen `Snake`. Studera problembeskrivningen och illustrationen noggrant som en vägledning om hur programmet, och följande metoder, ska fungera.

B1. Skriv deklarationen av en instansvariabel `snakeList` av typen `ArrayList` som skall lagra maskens alla delar, av typen `SnakePart`.

Skriv en konstruktor till klassen `Snake`. Den ska ta emot bredd, höjd och start-position som heltal, samt en initial riktning (`Direction`) för masken. Alla instansvariabler ska initieras och nödvändiga objekt skapas. Den första biten av masken skall skapas, på angiven startposition och riktning, och läggas till i `snakeList`.

```
private ArrayList<SnakePart> snakeList;

public Snake(int width, int height, int x, int y, Direction initialDirection) {
this.gameWidth = width;
this.gameHeight = height;

snakeList = new ArrayList<SnakePart>();
snakeList.add(new SnakePart(x, y, initialDirection));
```

B2. Skriv metoden `placeFruit` som slumpvis väljer en plats där en frukt skall placeras ut. Den valda positionen måste naturligtvis vara inuti spelplanen. Frukten får heller inte placeras på en ruta där masken befinner sig. Om den valda positionen är okej, ändras fruktens position till den, och `true` returneras. Om den valda positionen är ogiltig returneras `false` och den gamla frukt-positionen står kvar.

```
public boolean placeFruit() {
    fruitX = (int)(Math.random() * this.gameWidth);
    fruitY = (int)(Math.random() * this.gameHeight);

    for(SnakePart p : snakeList) {
        if(p.testPosition(fruitX, fruitY)) {
            return false;
        }
    }

    return true;
}
```

B3. Skriv klart metoden `move` i klassen `Snake`.

Först ska vi testa om den nya positionen gör att masken kolliderar med sig själv. Om det händer, ska `gameOver` bli tilldelat `true` och `false` returneras.

Det nyskapade `newHead` skall läggas till först i `snakeList`. Om huvudet träffar frukten skall en ny frukt placeras ut. (Tänk på att placerandet av frukter kan misslyckas, så upprepade försök krävs.) Om huvudet inte träffar en frukt skall den sista biten av masken tas bort så att masken behåller sin längd.

```
public boolean move(int relativeDirection) {
    SnakePart newHead = snakeList.get(0).move(relativeDirection);

    if(!newHead.isInside(this.gameWidth, this.gameHeight)) {
        gameOver = true;
        return false;
    }

    if(collideWithSnake(newHead.getX(), newHead.getY())) {
        gameOver = true;
        return false;
    }

    snakeList.add(0, newHead);
    if(newHead.testPosition(fruitX, fruitY)) {
        while(placeFruit() == false)
            ;
    } else {
        snakeList.remove(snakeList.size()-1);
    }

    return true;
}
```

B4. Skriv klart den inre loopen i `Snake`–klassens `main`–metod. Följande skall hända i ordning:

- Spelplanen skrivs ut (se klassens `toString`).
- En rad läsas från tangentbordet.
- Koppla en scanner till den inlästa raden, och undersök om det finns ett heltal att läsa.
- Om inget heltal finns, skriv ut ett felmeddelande och börja om loopen.
- Annars, läs heltalet.
- Om talet är 0, flytta masken framåt. Om talet är positivt, flytta masken åt höger (relativt sett). Om talet är negativt, flytta masken åt vänster relativt sett.
- Uppdatera `moveSuccessful` för att markera om förflyttningen var tillåten eller om spelet ska avslutas. Den informationen fås från metoden `move`.

```
while(moveSuccessful) {
    System.out.println(s.toString());
    System.out.println("(-1: Left, 0: Forward, 1: Right):");
    java.util.Scanner lineSc = new Scanner(sc.nextLine());
    if(lineSc.hasNextInt()) {
        relativeDirection = lineSc.nextInt();
    } else {
        System.out.println("Wrong input, please try again! ");
        continue;
    }

    if(relativeDirection <= -1) {
        relativeDirection = Direction.LEFT_TURN;
    } else if(relativeDirection >= 1) {
        relativeDirection = Direction.RIGHT_TURN;
    } else {
        relativeDirection = Direction.NO_TURN;
    }
    moveSuccessful = s.move(relativeDirection);
}
```