Tentamen i Programmeringsteknik I 2017-01-03

Skrivtid: 8.00-13.00

Tänk på följande

- Skriv läsligt! Använd *inte* rödpenna!
- Skriv bara på framsidan av varje papper.
- Skriv ej högst upp i det vänstra hörnet på papperet, ty det utrymmet är reserverat för häftklammer.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer och tentamenskod *överst i högra hörnet* på alla papper.
- Uppgift A1 skall rivas ur från tentamen och lämnas in tillsammans med övriga svar som skrivs på lösa papper.
- Fyll i försättssidan fullständigt.
- 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 och
  - underlåtenhet att utnyttja given eller egen tidigare skriven metod.
- För att bli godkänd (betyg 3) skall varje uppgift på A-delen vara rätt löst till cirka 80%.

  För betyget 4 krävs dessutom att minst hälften av uppgifterna på B-delen och betyg 5 att alla uppgifter på B-delen är i stort sett lösta. Vi bedömningen 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.
- Sist i detta dokument finns det enda hjälpmedlet till tentamen: En kortfattad lista med några java nyckelbegrepp.

Lycka till!
Torsten

Del A (obligatorisk för alla)

Tentamenskod

A1. Ringa in rätt svar och lämna in denna sida tillsammans med dina övriga svar.

a)	Vilket värde har z efter sista satsen ?
	`double x=14.1, y=0.5;`
	`int z = (int) (x+y);`
	`z-- ;`
		1. 14
		X. 13
		2. 13.6

b)	Vad ger följande satser för resultat?
	`int [] x = new int[3];`
	`x[0]=1; x[1]=2; x[2]=3;`
	`x[2]=x[0]+x[1]+x[2];`
	`System.out.println(x[2]);`
		1. Ger utskriften 6
		X. Medför **`ArrayIndexOutOfBoundsException`**
		2. Ger kompileringsfel

c)	Ange antalet variabler av primitiv typ som du ser i koden
	`double p = Math.random();`
	`int x,y;`
	`Dice d = new Dice(8);`
	`boolean a;`
		1. 2 st
		X. 4 st
		2. 3 st

d)	Vilka värden har a resp b efter sista satsen?
	`int a=4, b=3;`
	`a=b;`
	`b=a;`
		1. a=3, b=4
		X. a=3, b=3
		2. a=4, b=4

e)	Givet följande metodhuvud `void doThings(double g)` Vilken av vidstående satser ger kompileringsfel	1. **`doThings(5);`** X. **`double x = doThings(3.6);`** 2. **`doThings(4.7);`**

f)	Vilken av vidstående satser är korrekt att ha efter sista satsen?
	`ArrayList<Vehicle> q;`
	`q = new ArrayList<Vehicle>();`
	`Vehicle v = new Vehicle('W');`
	`q.add(v);`
		1. **`int x = q.get(0);`**
		X. **`char y = q.get(0);`**
		2. **`Vehicle v2 = q.get(0);`**

g)	Vad ger följande satser för resultat?
	`String s = "";`
	`int i = 34;`
	`s = s + "12" + i;`
	`System.out.println(s);`
		1. Det blir kompileringsfel
		X. Skriver ut: **`1234`**
		2. Skriver ut: **`46`**

h)	Vad ger följande satser för resultat?
	`int n=4, f=1;`
	`for (int i=1; i<=n; i++) {`
	`f=f*i;`
	`}`
	`System.out.println(f);`
		1. Skriver ut **`24`**
		X. Skriver ut **`10`**
		2. Ger kompileringsfel.

i)	Givet koden:
	`public class MyClass {`
	`private double w;`
	`public MyClass(double g) {`
	`this.w = g;`
	`}`
	`public void set(double a) {`
	`this.w = a*Math.sqrt(99.0);`
	`}`
	`}`
		1. **a** är en formell parameter
		X. **a** är en aktuell parameter
		2. 99.0 är en formell parameter

Uppgiften fortsätter på nästa sida →

j)	När man kör mainmetoden blir resultatet
	`public Pollax {`
	`  private int x;`
	`  public Pollax(int x) {`
	`  this.x = x ;`
	`  }`
	`  public int do(int y) {`
	`  this.x = this.x + y;`
	`  return this.x;`
	`  }`
	`  public static void main (String [] a) {`
	`  int z = 1;`
	`  Pollax p = new Pollax(z);`
	`  System.out.println(p.do(14));`
	`  }`
	`  }`
		1. Skriver ut **14**
		X. Skriver ut **15**
		2. Skriver ut **`java.lang.NullPointerException`**

A2.
Skriv en klass **Dice6** som skall representera en tärning med sex sidor och som har tre instansvariabler:
(1) **identity**, tärningens identitet, en sträng. (2) **value**, den sida som tärningen visar uppåt, ett heltal. (3) **numberOfThrows**, antal kast som gjorts med tärningen, ett heltal.
Det finns en mainmetod nedan som visar hur klassen skall fungera och en programkörning. Skriv:

a) Instansvariablerna
b) Konstruktorn, se mainmetoden vad den skall göra.
c) Metoden **getId** som returnerar identiteten
d) Metoden **getValue** som returnerar det värde som tärningen har
e) Metoden **getNrOfThrows** som returnerar antalet kast som gjorts med tärningen
f) Metoden **roll** som slår (”kastar”) tärningen.
g) Metoden **toString** som returnerar den information som framgår av programkörningen.

Här är mainmetoden som skall passa klassen **Dice6**:

```
public static void main(String[] arg){
   Dice6 d = new Dice6("DiceX");
   System.out.println(d.getId() + ", nr of throws=" + d.getNrOfThrows());
   int n = (int) (5*Math.random()+1);
   for (int i=1; i<=n; i++) {
     d.roll();
     System.out.println(d.getId()+", showing " + d.getValue());
   }
   System.out.println(d.getId()+", nr of throws=" + d.getNrOfThrows());
   System.out.println(d);
 }
```

Programkörning:

```
DiceX, nr of throws=0
DiceX, showing 6
DiceX, showing 1
DiceX, showing 6
DiceX, showing 1
DiceX, nr of throws=4
DiceX:nrOfThrows=4,value=1
```

A3.
Skriv de satser som saknas i nedanstående skal till en mainmetod i klassen **TestDice6** som skapar 20 st tärningar och kastar varje tärning tills den visar värdet sex. För varje tärning skall dess värden skrivas ut. Därefter när alla tärningar är kastade och fått värdet sex, skall programmet skriva ut hur många kast som hade gjorts för varje tärning. Se nedanstående programkörning. Tips: (1) Använd en array eller ArrayList. (2) En tärning ”vet” själv hur många gånger den har kastats.

```
public class TestDice6 {
  public static void main (String[] arg) {
    System.out.println("Throw each dice until it shows 6");
    … // Här saknas satser
    System.out.println("Some statistics...");
    … // Här saknas satser
  } // main
} // TestDie6
```

Mainmetoden skall använda klassen **Dice6** och fungera som nedanstående programkörning visar:

```
Throw each dice until it shows 6
Throwing Dice0 showing: 2 2 5 3 1 2 2 5 3 5 5 2 4 3 2 6
Throwing Dice1 showing: 1 4 6
Throwing Dice2 showing: 6
...                                                    Utskrifter är utelämnade
Throwing Dice17 showing: 1 4 5 6
Throwing Dice18 showing: 6
Throwing Dice19 showing: 4 2 3 3 5 2 6

Some statistics...
Dice0 is thrown 16 times
Dice1 is thrown 3 times
Dice2 is thrown 1 times
...                                                    Utskrifter är utelämnade
Dice17 is thrown 4 times
Dice18 is thrown 1 times
Dice19 is thrown 7 times
```

Del B (Endast för dem som siktar på betyget 4 eller 5)

För att simulera köbildningen av kunder vid en kassa behöver man följande tre javaklasser:

- **`Customer`**: Representerar en kund. En kund ”vet om” när den skapades, dvs tiden den kom till kassan, och den vet om hur många varor (**`items`**) den har med sig till kassan.
- **`Clock`**: Håller reda på tiden, dvs är en global klocka.
- **`CashDeck`**: Representerar en kassa som består av två saker: (1) Den kund vars varor just nu hanteras (betjänas) i kassan (dvs varor scannas en i taget). Vi betecknar denna som *aktuell* kund. (2) En kö av kunder som väntar på att hanteras.

De tre klasserna finns i en bilaga till tentamen, men tre delar i klassen **`CashDesk`** saknas.

Klassen **`Customer`** innehåller en mainmetod (se bilagan) som ger följande resultat när man kör den:

```
<X:0,3>
time=1:Scanning item 3 for customer X
time=2:Scanning item 2 for customer X
time=3:Scanning item 1 for customer X
time=4:No items left to scan for customer X
<Y:5,4>
Id=Y,born=5,items=0
```

Klassen **`CashDeck`** innehåller en mainmetod (se bilagan) som simulerar en köbildning. Den ger följande utskrifter (under förutsättning att de saknade delarna är åtgärdade):

```
current=<         >         queue=[]

Statistics
Idle time:           0
Processed customers: 0
Maximal queue time:  0

time=1:       current=<         >         queue=[]
time=2:       current=<         >         queue=[<cust1:2,2>]
time=3:       current=<cust1:2,2>         queue=[]
time=4:       current=<cust1:2,1>         queue=[<cust2:4,1>]
time=5:       current=<cust1:2,0>         queue=[<cust2:4,1>]
time=6:       current=<         >         queue=[<cust2:4,1>]
time=7:       current=<cust2:4,1>         queue=[]
time=8:       current=<cust2:4,0>         queue=[<cust3:8,2>]
time=9:       current=<         >         queue=[<cust3:8,2>, <cust4:9,3>]
time=10:      current=<cust3:8,2>         queue=[<cust4:9,3>]

Statistics
Idle time:           1
Processed customers: 2
Maximal queue time:  4
```

I ovan programkörning simuleras vad som händer under 10 st tidssteg. Förklaring till utskrifterna:

- Vid tiden=1 finns ingen aktuell kund i kassan (**`current=<  >`**), dvs inga varor scannas. Ingen kund finns i kön (**`queue=[]`**) som väntar.
- Vid tiden=2 finns ingen aktuell kund i kassan. En kund med identiteten **`cust1`** har anlänt till kön till kassan och den har 2 varor (`queue=[`**`<cust1:2,2>`**`]`**).**
- Vid tiden=3 har **`cust1`** blivit en aktuell kund som strax skall betjänas, dvs vars varor skall scannas (`curr`**`ent=<cust1:2,2>`**`)`**.** Kön är tom (**`queue=[]`**).
- Vid tiden=4 har en vara scannats för den aktuella kunden **`cust1`** (**`current=<cust1:2,1>`**). Till kön har en ny kund, **`cust2`** anlänt med 1 vara (**`queue=[<cust2:4,1>]`**).
- Vid tiden=5 har ytterligare en vara scannats (den sista) för den aktuella kunden **`cust1`**, (**`current=<cust1:2,0>`**). Kön är oförändrad.
- Vid tiden=6 har **`cust1`** lämnat kassan, dvs det finns ingen aktuell kund. Kön är oförändrad, men kunden där, **`cust2`**, kommer i nästa tidssteg att bli aktuell kund.
- …
- Vid tiden=10 är **`cust3`** aktuell kund (`curr`**`ent=<cust3:8,2>`**`)`**.** Kunden **`cust4`** står i kön (**`queue=[<cust4:9,3>]`**).

Före och efter simuleringen skrivs statistik ut:

- *Idle time* är det antalet tidssteg som kassan räknas som sysslolös, vilket är fallet när inga varor scannas (dvs finns ingen aktuell kund) och när kön är tom.
- *Processed customers* är det antal kunder som totalt har hanterats, dvs som har lämnat kassan.
- *Maximal queue time* är den längsta tid som någon färdigbehandlad kund tillbringat systemet (kötid + betjäningstid).

Klassen **`Clock`** är lite speciell eftersom den innehåller en s.k. *klassvariabel* och tre s.k. *klassmetoder*, vilket ordet *static* anger. Den innehåller inga instansvariabler och ingen konstruktor. Man skapar alltså inget objekt av typen **`Clock`**. Anrop av metoderna sker genom att skriva klassens namn, punkt och metodens namn. Exempel på användning:

```
Clock.resetTime();        // Nollställ klockan
int t = Clock.getTime();  // Ta reda på klockan, ger värdet 0
Clock.tic();              // Ticka klockan ett steg (+1)
t = Clock.getTime();      // Ger värdet 1
```

Uppgift B1:

Skriv klart de tre delar som saknas i klassen **`CashDesk`**, så att klassen passar mainmetoden och ger den utskrift som körexemplet ovan visar. De delar som saknas är:

a) Konstruktorn skall skapa en kassa där det inte finns någon *aktuell* kund och inte heller någon kund i kön.

b) Metoden **`addCustomer`** skall ställa en kund sist i kön till kassan.

c) Metoden **`step`** tar ett tidssteg när det gäller simuleringen. Något av följande skall ske vid ett tidssteg:

- Om det finns en aktuell kund som har varor kvar så scannas en vara.
- Om aktuell kund inte har några varor kvar räknas den som färdigbehandlad och lämnar kassan.
- Om det inte finns en aktuell kund och kön inte är tom så hämtas en ny kund från kön som blir aktuell kund.
- Om det inte finns en aktuell kund och kön är tom så räknas kassan som sysslolös (*idle*).

Metoden **`step`** skall också hålla reda på hur många kunder som totalt har behandlats (*processed*) samt den längsta tid (*maxTime*) som någon färdigbehandlad kund tillbringat i systemet (kötid + betjäningstid). Se utskrifterna från metoden **`printStatistics`** i körexemplet ovan.

Uppgift B2:

Klassen **`Store`** representerar en butik med en kassa. Butiken har även ett namn. I bilagan finns ett skal till klassen **`Store`** där vissa delar saknas (a,b,c,d) och en mainmetod som simulerar köbildningen i en butik. Skriv de delar som saknas för att klassen **`Store`** skall passa mainmetoden som ger den utskrift som följande körexempel visar:

```
Pollax:current=<         >  queue=[]
Time=1         current=<         >        queue=[]
Time=2         current=<         >        queue=[<cust1:2,4>]
Time=3         current=<cust1:2,4>        queue=[]
Time=4         current=<cust1:2,3>        queue=[<cust2:4,1>]
Time=5         current=<cust1:2,2>        queue=[<cust2:4,1>]
Time=6         current=<cust1:2,1>        queue=[<cust2:4,1>, <cust3:6,3>]
Time=7         current=<cust1:2,0>        queue=[<cust2:4,1>, <cust3:6,3>]
Time=8         current=<         >        queue=[<cust2:4,1>, <cust3:6,3>, <cust4:8,3>]
Time=9         current=<cust2:4,1>        queue=[<cust3:6,3>, <cust4:8,3>]
Time=10        current=<cust2:4,0>        queue=[<cust3:6,3>, <cust4:8,3>]

Statistics
Idle time:           1
Processed customers: 2
Maximal queue time:  6
```