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 av följande kod?

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

1) 0
2) 1
3) 2
4) 3
5) 4

b) Antag att klassen `PersonList` har en instansvariabel `ArrayList<Person> myFriends`. På vilket eller vilka sätt kan en metod med huvudet

```
public PersonList sort()
```

i klassen `PersonList` anropas?

1) `Person[]result = sort();`
2) `String result = sort();`
3) `PersonList result = sort();`
4) `ArrayList<Person> result = sort();`
5) `Person result = sort();`

c) Vilken eller vilka returtyper kan nedanstående metod ha?

```
public typ? check(int value) {
  return value < 0;
}
```

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

d) Vad blir resultatet av nedanstående kod?

```
int a = 3;
int b = 4;
a = a + b;
b = a - b;
a = a - b;
System.out.println(a + ", " + b);
```

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

e) I klassen `Friends` finns deklarationen

```
private ArrayList<Person> myFriends;
```

I en metod i denna klass finns uttrycket

```
this.myFriends.get(3).getName().equals(n)
```

I vilken klass finns metoden `getName`?

1) I klassen `Friends`
2) I klassen `Person`
3) I klassen `ArrayList`
4) I klassen `String`

f) Satsen

```
double x = (int)(1 + 3/2.0) + 3;
```

resulterar i

1) Kompiteringsfel
2) Exekveringsfel
3) `x` får värdet `5.0`
4) `x` får värdet `5`
5) `x` får värdet `5.5`

Här följer ett antal uppgifter på klassen `Pair`. Denna klass ska lagra två heltal x och y som instansvariabler.

A2. Skriv deklarationen av de två instansvariablerna.

```
private int x;
private int y;
```

A3. Skriv en konstruktor som tar emot och sätter värden på de två instansvariablerna.

```
public Pair(int x, int y) {
    this.x = x;
    this.y = y;
}
```

A4. Skriv en parameterlös konstruktor som sätter instansvariablerna till 1.

```
public Pair() {
    this.x = 1;
    this.y = 1;
}
```

A5. Skriv en `toString`-metod som returnerar paret omgivet av parenteser och separerat av kommatecken. Exempel på resultat: `(3, 4)`.

```
public String toString() {
    return "(" + x + ", " + y + ")";
}
```

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

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

A7. Skriv de satser som behövs för att läsa in två heltal från tangentbordet och skapa ett `Pair`-objekt med av dessa värden som instansvariabler. Du behöver inte skriva några ledtexter och inte heller kontrollera att det verkligen går att läsa heltal.

```
Scanner scan = new Scanner(System.in);
int x = scan.nextInt();
int y = scan.nextInt();
Pair p = new Pair(x, y);
```

Kommentar: Många missade denna uppgift. I den bilagda referenssidan står det hur man kopplar en scanner till tangentbordet och hur man läser heltal.

Resten av uppgifterna handlar om klasserna `Customer`, `Desk` och `Store` som används i programmet för butikssimulering. Se beskrivningen och koden i bilagorna!
Varje klass innehåller en `main`-metod och utskrifterna från dessa ligger som kommentarer sist i varje klass.

A8. Skriv metoden `done()` i klassen `Customer` som returnerar `true` om kunden har 0 (eller negativt antal) varor annars `false`.

```
public boolean done() {
  return load <= 0;
}
```

A9. Skriv en parameterlös konstruktor till klassen `Desk` som skapar en stängd kassa och ett arraylist-objekt med 0 kunder i.

```
public Desk() {
  this.open = false;
  this.queue = new ArrayList<Customer>();
}
```

A10. Skriv metoden `add` i klassen `Desk`. Metoden ska ta emot ett `Customer`-objekt som parameter och lägga det sist i kassans kö. Metoden ska inte returnera något värde.

```
public void add(Customer c) {
  queue.add(c);
}
```

A11. Klassen `Store` innehåller en array av `Desk`-objekt (se kodbilagan). Skriv klart konstruktorn nedan som skapar en array med n `Desk`-objekt och öppnar det första av dessa.

```
public Store(int n) {
  theDesks = new Desk[n];
  for (int i=0; i<n; i++) {
    theDesks[i] = new Desk();
  }
  theDesks[0].open();
}
```

A12. Metoden `findFirstClosed()` i klassen `Store` ska leta upp index för den första stängda kassan (alltså den stängda kassa som har lägst index). Om det inte finns någon stängd kassa ska -1 returneras.

Skriv klart metoden!

```
public int findFirstClosed() {
  for (int i=0; i<theDesks.length; i++) {
    if (!theDesks[i].isOpen()) {
        return i;
    }
  }
  return -1;
}
```

Del B (för betyg 4 och 5)

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

B1. Skriv metoden `step()` i klassen `Desk`. Metoden ska ta ett tidssteg i kassan. Om första kunden i kön är färdigbehandlad ska den tas bort ur kön. Om kunden ej är klar ska en av kundens varor skannas.

```
public void step() {
  if (!queue.isEmpty()) {
    Customer c = queue.get(0);
    if (c.done()) {
       queue.remove(0);
    } else {
       c.unload();
    }
  }
}
```

Observera att man måste kontrollera att kön inte är tom - det går inte att göra get(0) om kön är tom!

B2. Skriv metoden `public ArrayList<Customer> removeHalfQueue()` i klassen `Desk` som flyttar kunderna från den bakre halvan av kassans kö till en ny arraylist och returnerar denna som värde. Se programkörningen för klassen `Store`! Om köns längd är udda ska den kvarvarande delen vara större. Exempel: Om kön innehåller 7 kunder ska de 3 bakersta tas ut och de 4 främsta stå kvar.

```
public ArrayList<Customer>
             removeHalfQueue()
{
   ArrayList<Customer> result =
       new ArrayList<Customer>();
   int n = queue.size();
   int i = n/2;
   if (n%2 == 1) {
     i++;
   }
   while (i < queue.size()) {
     result.add(queue.remove(i));
   }
   return result;
}
```

Observera att `size()` minskar när man gör `remove`. Dessutom flyttas alla bakomvarande element fram ett steg.

B3. Skriv metoden `openNewDesk()` i klassen `Store` som, med hjälp av `findFirstClosed()` (uppgift A12), öppnar en ny kassa. Om det inte finns någon stängd kassa görs ingenting. Om det inte är den första kassan som ska öppnas ska hälften av kön från kassan omedelbart före denna kassa läggas in som den nyöppnade kassans kö. Se bilagan som beskriver butikssimuleringen!

Tips: Se vilka metoder det finns för att öppna en kassa!

```
public void openNewDesk() {
   int i = findFirstClosed();
   if (i == -1) {
      return;
   } else if (i==0) {
      theDesks[i].open();
   } else {
      theDesks[i].open(theDesks[i-1].removeHalfQueue());
   }
}
```

B4. Skriv metoden `Desk findShortestQueue()` som returnerar den öppna kassa som har kortast kö. (Om flera kassor har lika kort kö så går det bra med vilken som helst av dem.)

```
public Desk findShortest() {
     Desk shortest = null;
     for (Desk d: theDesks) {
        if (d.isOpen() && shortest==null) {
           shortest=d;
        } else if (d.isOpen() &&
                   d.queueLength() <=
                   shortest.queueLength()) {
           shortest = d;
        }
     }
     return shortest;
}
```

Observera att man måste kontrollera att man väljer en *öppen* kassa och att man, på något sätt, ska hantera fallet att alla kassor är stängda. I detta fall har vi valt att returnera `null`.

B5. Antag att man vill samla statistik (typ maxtid, medelvärde, standardavvikelse, ...) för hur lång tid (hur många tidsteg) kunderna tillbringat i systemet från det att de ställde sig i kön tills de blivit färdigbehandlade.

a) Skriv den nya eller ändrade kod som behövs i klassen `Customer`.

b) Skriv den nya eller ändrade kod som behövs i klassen `Desk`.

c) Skriv den nya eller ändrade kod som behövs i klassen `Store`.

Observera: Du ska *inte* skriva koden för statistikberäkningarna utan bara se till att data för dessa beräkningar finns tillgängliga till exempel i en array, en araylista eller ett `Measurements`-objekt.

a) I klassen `Customer`:
Kunderna måste förses ytterligare en instansvariabel som anger ankomsttid. Den kan sättas av konstruktorn när kunden skapas.

Alternativt kan man föra in en global klocka `public static int getTime()` i klassen `Store`.

(Ett sämre alternativ är att ge kunden åldern 0. Då måste man för varje steg gå igenom alla kunder och öka deras ålder.)

b) I klassen codeDesk
Låt `step`-metoden returnera den kund som tas ut (eller, om man har en global klocka, kundens ”livstid”).

c) I klassen `Store`
Använd en arraylist eller ett `Measurements`-objekt som lagrar tider för kunder som blir klara. Butikens `step`-metod ska ta emot kunderna som kommer ut från de olika kassorna, beräkna deras tid i kösystemet och lägga den tiden i arraylistan.