Del A (obligatorisk för alla)

A1. Ringa in rätt svarsalternativ eller skriv svar i ruta om sådan

a) Satsen

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

resulterar i

1) Kompileringsfel
2) x får värdet 5.0
3) x får värdet 5
4) x får värdet 5.5

b) Satsen

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

resulterar i

1) Kompileringsfel
2) x får värdet 5.0
3) x får värdet 5
4) x får värdet 5.5

c) Hur många objekt skapas av koden

```
int[] a = new int[3];
ArrayList<Turtle> t = new ArrayList<Turtle>();
t.add(new Turtle(new World());
```

[4]

d) Antag att nedanstående kod går att kompilera och köra

```
Test t = new Test();
System.out.println(t.ps[0].getName());
```

Markera ett alternativ som *måste* vara sant?

1) ps är en klassmetod
2) ps är en objektmetod
3) ps är en instansvariabel
4) ps är en lokal variabel
5) ps är en formell parameter
6) ps är en array

e) Samma kod som i ovanstående uppgift men markera nu ett *annat* alternativ som *kan* vara sant.

1) ps är en klassmetod
2) ps är en objektmetod
3) ps är en instansvariabel
4) ps är en lokal variabel
5) ps är en formell parameter
6) ps är en array

f) Antag följande metodhuvud

```
void foo(String s)
```

Markera ett anrop som är *felaktigt*!

1) `foo("Lotta" + 3);`
2) `foo("" + 3);`
3) `foo(3);`
4) `foo(null);`
5) `foo("");`

g) Vad leder nedanstående kod till?

```
ArrayList<String> a = new ArrayList<String>();
a.add("Kalle");
a.add("Anka");
a.remove(0);
System.out.println(a.get(0));
```

1) Utskrift av Kalle
2) Utskrift av Anka
3) Utskrift av null
4) Utskrift av en tom sträng ("")
5) NullPointerException

För att denna uppgift ska betraktas som godkänd bör du ha minst 5 rätta deluppgifter.

Klassen `Container` ska lagra ett antal heltal (för enkelhetens skull — i praktiken skulle man antagligen lagra någon typ av objekt).

En container har ett ”tak” dvs en maxstorlek för hur många tal som ryms. Maxstorleken anges när containern skapas.

Exempel på kod som använder container-klassen. Till höger står de till koden svarande utskrifterna.

```
public static void main(String[] args) {
   Container c = new Container(6);
   System.out.println("c = " + c.toString());            c = <6 : []>
   c.add(5); c.add(3); c.add(5); c.add(4);
   System.out.println("c = " + c.toString());            c = <6 : [5, 3, 5, 4]>
   System.out.println("c.isFull() = " + c.isFull());     c.isFull() = false
   c.add(2); c.add(1);
   System.out.println("c = " + c.toString());            c = <6 : [5, 3, 5, 4, 2, 1]>
   System.out.println("c.isFull() = " + c.isFull());     c.isFull() = true

   Container d = c.copy();
   System.out.println("d = " + d);                       d = <6 : [5, 3, 5, 4, 2, 1]>

   System.out.println("c.removeMin: ");                  c.removeMin:
   while( !c.isEmpty() ) {
       System.out.print("  " +c.removeMin());              1  2  3  4  5  5
   }
   System.out.println();
   System.out.println("c = " + c.toString());            c = <6 : []>

   System.out.println("d = " + d);                       d = <6 : [5, 3, 5, 4, 2, 1]>
   System.out.println("d.sorted() = " +                  d.sorted() = <6 : [1, 2, 3, 4, 5, 5]>
                      d.sorted());
   int[] arr = d.toArray(2, 4);
   System.out.println("d.toArray(2, 4) = " +             d.toArray(2, 4) = [3, 4, 2]
                      Arrays.toString(arr));
}
```

Klassen `Container` har vissa likheter med klassen `Measurements` i lektion 7 men, till skillnad från den klassen, ska den

1. implementeras med en `ArrayList`,
2. lagra heltal och
3. ej kunna utvidgas dvs högst lagra det antal värden som angavs när objektet skapas.

Dina lösningar på uppgifterna nedan ska fungera enligt ovanstående exempel!

A2. Klassen `Container` ska ha en instansvariabel av typen `ArrayList` med namnet `contents` som lagrar värden och en av typ `int` som anger maximal storlek. Skriv deklarationerna för dessa två instansvariabler.

```
private ArrayList<Integer> contents;
private int max;
```

A3. Skriv den konstruktor som som används i exempelkoden dvs den som tar emot max-antalet.

```
public Container(int max) {
   this.contents = new ArrayList<Integer>(max);
   this.max = max;
}
```

A4. Skriv `toString()`-metoden som producerar resultat enligt exempelkoden.

```
public String toString() {
   return "<" + max + " : " + contents + ">";
}
```

A5. Skriv metoden `boolean isFull()` som returnerar `true` om det maximala antalet tal är lagrade, annars `false`.

```
public boolean isFull() {
   return contents.size() == max;
}
```

A6. Skriv metoden `void add(int x)`. Om det finns plats för ytterligare ett tal ska det lagras sist i arraylistan. Om containern är full ska ett undantag kastas med koden
`throw new RuntimeException("Container full")`

```
public void add(int x) {
   if (isFull()) {
      throw new RuntimeException("The container is full");
   } else {
      contents.add(x);
   }
}
```

A7. Skriv metoden `public boolean equals(Container c)` som returnerar `true` om detta objekt är lika som objektet `c`. Med *lika* menas att de har samma maxstorlek och exakt samma innehåll i samma ordning.
Tips: Metoden `equals` i klassen `ArrayList` kan vara mycket användbar här!

```
public boolean equals(Container c) {
   return this.max == c.max && this.contents.equals(c.contents);
}
```

eller

```
public boolean equals(Container c) {
   return this.toString().equals(c.toString());
}
```

A8. Skriv en metod `public int compareTo(Container c)` som jämför den egna containern med containern `c` storleksmässigt. Storleken bestäms av det maximala utrymmet (inte av hur många eller vilka tal som är lagrade).

Metoden ska returnera `-1` om den egna containern är mindre än containern `c`, `0` om containrarna är lika stora, annars `1`.

```
public int compareTo(Container c) {
   if (this.max < c.max) {
      return -1;
   } else if (this.max == c.max) {
      return 0;
   } else {
      return 1;
   }
}
```

A9. Skriv de satser som behövs för att skapa ett `Scanner`-objekt kopplat till tangentbordet, skriva ut en fråga till användaren om en storlek (heltal), läsa in storleken med hjälp av `Scanner`-objektet och sedan skapa en container med denna storlek.

```
Scanner scan = new Scanner(System.in);
System.out.print("Container size: ");
int max = scan.nextInt();
Container c = new Container(max);
```

A10. Skriv metoden `public Container copy()` som skapar och returnerar ett nytt `Container`-objekt som är lika stort och har samma innehåll som det egna objektet.

```
public Container copy() {
   Container res = new Container(max);
   for (int value : contents) {
      res.add(value);
   }
   return res;
}
```

A11. Skriv metoden `public int indexMin()` som returnerar *index* för det minsta värdet i containern.

```
public int indexMin() {
   int index = 0;
   for (int i=1; i < contents.size(); i++) {
      if (contents.get(i) < contents.get(index)) {
         index = i;
      }
   }
   return index;
}
```

A12. Skriv metoden `public int removeMin()` som tar bort och returnerar minsta elementet ur containern.

```
public int removeMin() {
   return contents.remove(indexMin());
}
```

Del B (för betyg 4 och 5)

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

B1. Skriv en metod `public Container sorted()` i klassen `Container` som skapar ett nytt `Container`-objekt med samma data som i detta objekt men där värdena är lagrade i storleksordning. Se körexemplet!

```
public Container sorted() {
   ArrayList<Integer> result = new ArrayList<Integer>();
   for (int value : contents) {
      int i = 0;
      while (i<result.size() && value > result.get(i)) {
         i++;
      }
      result.add(i, value);
   }
   return new Container(result);
}


public Container(ArrayList<Integer> al) {
   this.contents = al;
   this.max = al.size();
}
```

B2. Skriv en metod `public int[] toArray(int low, int high)` i klassen `Container` som skapar och returnerar en array med de tal som är större än eller lika med `low` och mindre än eller lika med `high`. Se körexemplet!

```
public int[] toArray(int low, int high) {
     int n = 0;
     for (int i: contents) {
        if (i >= low && i <= high) {
           n++;
        }
     }
     int [] result = new int[n];
     n = 0;
     for (int i: contents) {
        if (i >= low && i<=high) {
           result[n++] = i;
        }
     }
     return result;
}
```

B3. Skriv en metod `public boolean sameContents(Container c)` i klassen `Container` som returnerar `true` om denna container har samma innehåll som containern `c` oavsett ordning och storlek, annars `false`.

```
public boolean sameContents(Container c) {
   return this.sorted().equals(c.sorted());
}
```

B4. Skriv en klass `MeanFilter` som representerar ett *medelvärdesfilter*. Filtret har en storlek n som sätts när filtret skapas. För varje värde som matas (metoden `add`) så ska medelvärdet av detta värde och de närmast föregående $n - 1$ värdena returneras.

Körexempel:

```
public static void main(String[] args) {
   double[] testData = {1, 5, 6, 4, 5,
                        3, 1, 5, 9, 4};
   MeanFilter mf = new MeanFilter(3);

   System.out.println("In   Ut   Lagrade");
   for (double d: testData) {
      System.out.println(d + "  " + mf.add(d) + "  " + mf.toString());
   }
}
```

Resulterande utskrift:

```
In   Ut   Lagrade
1.0  1.0  [1.0]
5.0  3.0  [1.0, 5.0]
6.0  4.0  [1.0, 5.0, 6.0]
4.0  5.0  [5.0, 6.0, 4.0]
5.0  5.0  [6.0, 4.0, 5.0]
3.0  4.0  [4.0, 5.0, 3.0]
1.0  3.0  [5.0, 3.0, 1.0]
5.0  3.0  [3.0, 1.0, 5.0]
9.0  5.0  [1.0, 5.0, 9.0]
4.0  6.0  [5.0, 9.0, 4.0]
```

Skriv klassen med instansvariabler, konstruktor och metoder så att filtret fungerar i enlighet med ovanstående exempel.

```
public class MeanFilter {
   private ArrayList<Double> theValues;
   private int size;

   public MeanFilter(int size) {
      this.size = size;
      this.theValues = new ArrayList<Double>(size);
   }

   public double add(double value) {
      if (theValues.size() == size) {
         theValues.remove(0);
      }
      theValues.add(value);
      return mean();
   }

   public double mean() {
      double sum = 0;
      for (double d : theValues) {
       sum += d;
      }
      return sum/theValues.size();
   }

   public String toString() {
      return theValues.toString();
   }

}
```