```
/**
 *  Suggested solutions to DAT043 exam 2021-03-13
 */

   // --------- 1o2 ----------        (4p)

   void program() {
      Scanner sc = new Scanner(in);
      out.print("Antal räkningar > ");
      int nObservations = sc.nextInt();

      int nPackages = 0;
      int max = 100;
      out.print(" > ");
      for (int i = 0; i < nObservations; i++) {
         int tmp = sc.nextInt();
         if (tmp > max)
            nPackages++;
         max = tmp;
      }
      out.println("Beviserligen minsta antal köpta påsar: " + nPackages);
   }

   // --------- 3 ----------        (7p)

   int[] expand(int[] a) {
      int len = getLength(a);
      int[] e = new int[len];
      int k = 0;
      for (int i = 0; i < a.length - 1; i += 2) {
         for (int j = 0; j < a[i]; j++) {
            e[k++] = a[i + 1];
         }
      }
      return e;
   }

   int getLength(int[] a) {
      int len = 0;
      for (int i = 0; i < a.length; i += 2) {
         len += a[i];
      }
      return len;
   }
```

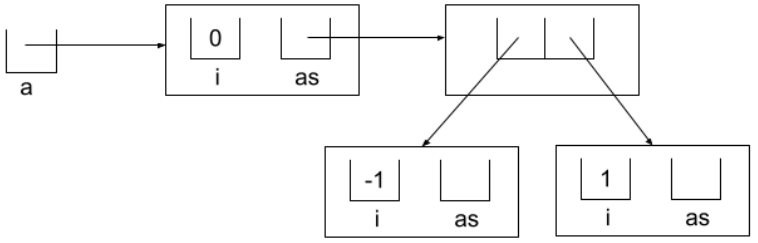
// --------- 4 ---------- (12p)

```
 int[][] intersections(int[][] m) {
    int nIntersect = m.length * (m.length - 1) / 2;  // Formula for number of intersections
    int[][] is = new int[nIntersect][0];
    int k = 0;
    for (int i = 0; i < m.length - 1; i++) {
       for (int row = i + 1; row < m.length; row++) {
          int[] ia = intersection(m[i], m[row]);
          is[k++] = ia;
       }
    }
    return is;
 }

 int[] intersection(int[] a1, int[] a2) {
    int[] tmp = new int[a1.length];   // Both have same number of columns
    int k = 0;
    for (int value : a1) {
       if (contains(a2, value)) {
          tmp[k] = value;
          k++;
       }
    }
    return tmp;
 }

 boolean contains(int[] arr, int n) {
    for (int i : arr) {
       if (i == n) {
          return true;
       }
    }
    return false;
 }
```

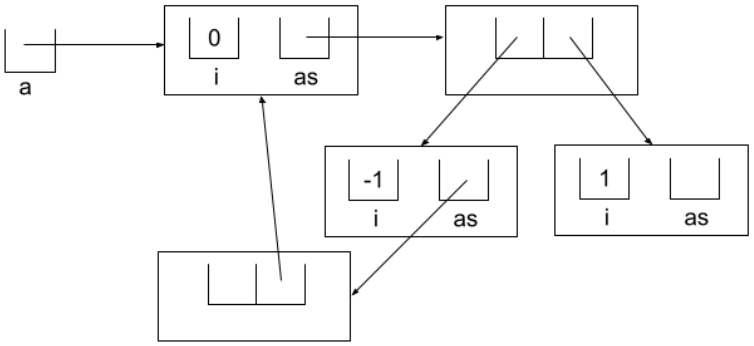
```
// --------- 5 ------------------    (10p)

    boolean containsAll(String s1, String s2) {
        StringBuilder s1b = new StringBuilder(s1);
        for (char ch : s2.toCharArray()) {
            int i = s1b.indexOf(String.valueOf(ch));
            if (i >= 0) {
                s1b.deleteCharAt(i);
            } else {
                return false;
            }
        }
        return true;
    }


    String smallestSubStr(String s1, String s2) {
        List<String> substrs = new ArrayList<>();
        for (int i = 0; i < s1.length(); i++) {
            for (int j = i + 1; j <= s1.length(); j++) {
                substrs.add(s1.substring(i, j));
            }
        }
        int len = s1.length();
        String smallest = s1;
        for (String str : substrs) {
            if (containsAll(str, s2) && str.length() < len) {
                smallest = str;
                len = str.length();
            }
        }
        return smallest;
    }
```

// -------- 6 --------- (8p)

Före:



Efter:



// -------- 7 --------- (11p)

```
public class User {
    private String name;
    private String passwd;

    public User(String name, String passwd) {
        this.name = name;
        this.passwd = passwd;
    }
    // getter setter as needed
}

public class Message {
    private final String text;
    private final User user;

    public Message(String text, User user) {
        this.text = text;
        this.user = user;
    }
     // getter/setter as needed
}

public class Thread {
    private final int id;
    private final List<Message> messages = new ArrayList<>();

    public Thread(int id) {
        this.id = id;
    }

    public boolean append(Message msg) {
        return messages.add(msg);
    }
    // getter/setter as needed
    public int getId(){
        return id;
    }
}

public class Channel {
    private final int id;
    private final List<Thread> threads = new ArrayList<>();

    public Channel(int id) {
        this.id = id;
    }
```

```
    Thread getThread(int threadId) {
        for( Thread t : threads){
            if( t.getId() == threadId){
                return t;
            }
        }
        return null;
    }
     // getter/setter as needed
}

public class Sluck {
    private final List<User> users = new ArrayList<>();
    private final List<Channel> channels = new ArrayList<>();

    boolean publish(User user, Message msg, Channel channel, int threadId) {
        if (!users.contains(user) || !channels.contains(channel)) {
            return false;
        }
        int i = channels.indexOf(channel);
        Channel c = channels.get(i);
        Thread t = c.getThread(threadId);
        if (t != null) {
            return t.append(msg);
        } else {
            return false;
        }
    }
}
```

// ------ 8 a) ---------- (4p)

Kompilatorn tillåter typomvandling från/till gränssnitt även om ingen super/sub relation existerar därför att det kan i vissa fall fungera ändå (t.ex. som nedan).

```
interface A {}
class X {}

A a;
X x = new X();

// ---------- This is why first case is allowed ----------

x = new Y();  // If Y like below exists .. (have super = sub)
a = (A) x;    // ... this will work  (X and A no super/sub)

class Y extends X implements A {

}

// ---------- This will make second case is allowed ----------

a = new Z();   // If Z like below exists...
x = (X) a;     // ... this will work

class Z extends X implements A {

}
```

```
// ------ 8 b) ----------              (4p)
   // See comments in code

   void program() {
      B b1 = new B(123, 1);
      B b2 = new B(123, 2);
      A a1 = b1;
      A a2 = new A(123);
      /*
         All equals overloaded i.e. typ of variable decides method to use.
       */
      if (a1.equals(b2)) {       // equals in A will be called
         out.println("Happy");
      } else {
         out.println("NOT happy");
      }
      if (a1.equals(a2)) {       // equals in A will be called
         out.println("Happy");
      } else {
         out.println("NOT happy");
      }
   }

   public class A {
      private int aNumb;

      public A(int aNumb) {
         this.aNumb = aNumb;
      }

      /*
           First call: Object is a B object running inherited method from A
           ("this" is of type B).
           Parameter a is an B object i.e getClass() != a.getClass() is false.
           Will return result true (first Happy printed)

           Second call: Object is an B object running inherited method from A
           ("this" is of type B).
           Parameter a is an A object i.e getClass() != a.getClass() is true.
           Will return result false (second NOT Happy printed)
       */
      public boolean equals(A a) {    // Overloading! ...
         if (this == a) {
            return true;
         } else if (a == null) {
            return false;
         } else if (getClass() != a.getClass()) {
            return false;
```

```
            }
            return aNumb == a.aNumb;
        }
        //public boolean equals(Object o) { ...}    // Inherited from Object
    }

    public class B extends A {
        // NO impact
    }


}
```