# Lösningsförslag övning 2.

**Uppgift 1**

```
//before: arr.length >= 2
public static int minGap(int[] arr) {
  int min = Math.abs(arr[0] -arr[1]);
  for (int i = 1; i < arr.length-1; i = i + 1) {
      int diff = Math.abs(arr[i] - arr[i+1]);
      if (diff < min)
            min = diff;
  }
  return min;
}//minGap
```

**Uppgift 2**

```
public static double percentOfEven(int[] arr) {
  if (arr == null || arr.length == 0)
      return 0;
  int nrOfEven = 0;
  for (int i = 0; i < arr.length; i = i + 1) {
      if (arr[i] % 2 == 0)
          nrOfEven = nrOfEven + 1;
  }
  return (double) 100 * nrOfEven / arr.length;
}//percentOfEven
```

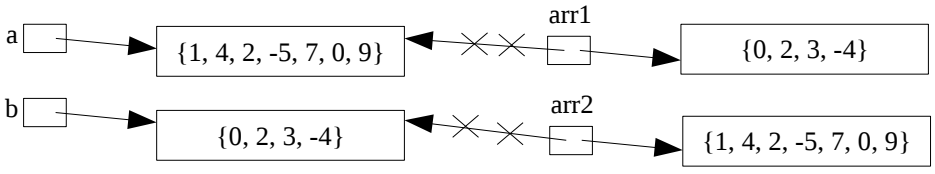
**Uppgift 3**

a)

```
//before: arr1 != null && arr2 != null && arr1.length == arr2.length
//after: the elements in arr1 and arr2 have been swapped
public static void swapAll(int[] arr1, int[] arr2) {
  for (int i = 0; i < arr1.length; i = i + 1) {
      int temp = arr1[i];
      arr1[i] = arr2[i];
      arr2[i] = temp;
  }
}//swapAll
```

b) Förvillkoren till metoden måste vara att inparametrarna inte får vara **null** och att båda fälten är lika långa. Orsaken till detta är följande:

Fält är statiska strukturer, dvs. det går inte att ändra storleken på ett fält. Om man önskar ett större eller mindre fält måste ett nytt fält med den önskade storleken skapas. Detta är i sig inget problem. Här har vi dock två fält, säg fälten a och b, som är aktuella parametrar till en metod, och Java använder *värdeanrop* vid parameteröverföring. I metoden kan vi skapa nya fält med de önskade storlekarna och låta de aktuella parametrarna, säg arr1 och arr2, referera till dessa fält. Med detta kommer inte att påverka de aktuella parametrarna a och b.



Man kan fråga sig om det överhuvudtaget finns ett behov av metoden swapAll. Behöver anroparen av metoden verkligen byta *elementen i fälten* som refereras av a och b, räcker det inte att a och b byter *referenser till fälten*? Vilket görs med satserna:

```
int[] temp = arr1;
arr1 = arr2;
arr2 = temp;
```

**Uppgift 4**

```
//before: word1 != null && word2 != null
public static boolean isAnagram(String word1, String word2) {
    String str1 = word1.toLowerCase();
    String str2 = word2.toLowerCase();
    char[] c1 = str1.toCharArray();
    char[] c2 = str2.toCharArray();
    Arrays.sort(c1);
    Arrays.sort(c2);
    if (Arrays.equals(c1, c2))
        return true;
    else
        return false;
}//isAnagram
```

**Uppgift 5**

```
//before: items != null
public static double[] removeFaulty(double[] items, double correctSize, double tolerance) {
    double[] temp = new double[items.length];
    int nrAccepted = 0;
    for (int i = 0; i < items.length; i = i + 1) {
        if (Math.abs(items[i] - correctSize) < tolerance) {
            temp[nrAccepted] = items[i];
            nrAccepted = nrAccepted + 1;
        }
    }
    return java.util.Arrays.copyOf(temp, nrAccepted);
}// removeFaulty
```

**Uppgift 6**

a)

```
//before: theDice != null
public static int[] roll(GenericDie theDice,  int nrRoll) {
    int[] throwings =  new int[theDice.getSides()];
    for (int i = 1; i <= nrRoll; i = i + 1) {
        theDice.roll();
        int dots = theDice.getValue();
        throwings[dots-1] = throwings[dots-1] +1;
    }
    return throwings;
} //roll
```

b)

```
import java.util.*;
public static class FullHouse {
   public static void main (String[] args) {
      GenericDie[] dices  = {new GenericDie(6), new GenericDie(7),  new GenericDie(8),
                                    new GenericDie(9), new GenericDie(10)};
      System.out.println("Sannolikheten få en kåk är " +   probabilityOfFullHouse(dices, 100000));
   } // main

   //before: theDice != null
   public double probabilityOfFullHouse(GenericDie[] dices, int nrOfTurns) {
      int nrFullHouse = 0;
      for (int turn = 1; turn <= nrOfTurns; turn = turn + 1) {
        int[] values = new int[5];
        for (int i = 0; i < dices.length; i = i + 1) {
             dices[i].roll();
             values[i] = dices[i].getDots();
        }
        Arrays.sort(values);
        if (values[0] == values[2] && values[3] == values[4]
                  || values[0] == values[1] && values[2] == values[4])
                 nrFullHouse = nrFullHouse + 1;
      }
      return (double) nrFullHouse/ nrOfTurns;
   }//probabilityOfFullHouse
} // FullHouse
```