Problem 1

Define a (functional) interface **SFilter** declaring one method **test** which takes a **String** and returns a **boolean**. The interface defines also a static function

```
public static String[] filter(String[] arr,
                              SFilter filt) {
    // ...
}
```

which takes an array of **String**s and an object implementing the **SFilter** interface. The function returns another array of **String**s which contains only those elements of the original array for which calling **test** on filt returns **true**.
Define also a class **LenFilter** which implements the interface. The class has one field minLen of type **int** which is set by the constructor; the implementation of the function **test** returns **true** if, and only if, the string passed as the arguments is of length at least minLen.
In the **main** function create an array of **String**s and then call the **SFilter.filter** function passing this array and implementation of the **SFilter** interface in the form of:

- an object of **LenFilter** initialized in this way that its **test** method selects only strings of the length greater than 4;
- an object of an anonymous class which implements the **SFilter** interface in such a way that it selects only strings whose first letter is earlier in the alphabet that 'D' but later or equal 'A';
- a lambda which selects only those strings whose first letter is later in the alphabet than 'H' but earlier or equal 'Z'.

The following program:

download *StringFilter.java*

```
import java.util.Arrays;

@FunctionalInterface
interface SFilter {
    // ...
}

class LenFilter implements SFilter {
    // ...
}

public class StringFilter {
    public static void main (String[] args) {
        String[] arr = {"Alice", "Sue", "Janet", "Bea"};
        System.out.println(Arrays.toString(arr));

        String[] a1 = SFilter.filter(arr, /* ... */);
        System.out.println(Arrays.toString(a1));

        String[] a2 = SFilter.filter(arr, /* ... */);
        System.out.println(Arrays.toString(a2));

        String[] a3 = SFilter.filter(arr, /* ... */);
        System.out.println(Arrays.toString(a3));
    }
}
```

should print

```
    [Alice, Sue, Janet, Bea]
    [Alice, Janet]
    [Alice, Bea]
    [Sue, Janet]
```

Problem 2

Create the interface

```
    @FunctionalInterface
    interface FunDD {
        double fun(double x);

        static double xminim(FunDD f, double a, double b) {
            // ...
        }
    }
```

declaring the method **fun** (of type **double**→**double**) and *defining* the static function **xminim** which

- takes a reference `f` to an object of a class implementing the **FunDD** interface and limits `a` i `b`;
- finds the value of the argument `x` from the range $[a, b]$ for which the method **fun** of the object `f` assumes the minimum value (i.e., finds the location of the minimum of the function on $[a, b]$); a somewhat primitive way to find it would be to calculate `fun(x)` for values of the argument between `a` and `b` for every value with a (small) fixed step (e.g., 1e-5).

Write a class **Parabola** implementing the **FunDD** interface with fields `a`, `b` and `c` of type **double** in which the method **fun** calculates the value of the quadratic function $ax^2 + bx + c$.

In the **main** of a separate class, use the static method **xminim** to find the minima of a few functions in three different ways:

- passing an object of class **Parabola** and limits `a` i `b` (for example for function $x^2 - x + 5/4$ for $x \in [0,1]$);
- passing an object of an anonymous class implementing the **FunDD** interface and limits `a` i `b` (for example for function $\sqrt{(x-0.75)^2+1}$ for $x \in [0,2]$);
- passing a lambda and limits `a` i `b` (for example for function $x^2(x-2)$ for $x \in [0,2]$).

For the given examples, the results should be 1/2, 3/4 and 4/3 (with high accuracy).

Problem 3

Note: *If you don't know collections yet, you can use plain arrays instead of lists and replace* **Collections.sort** *by* **Arrays.sort**. *Similarily, if you don't know enumerations (***Enum***), you can use plain* **ints** *(for example, 0 for* RED*, 1 for* GREEN *and 2 for* BLUE*).*

Create a class **MyColor** extending **java.awt.Color**. One constructor will be sufficient; it should take three color components (red, green, blue) from the range $[0, 255]$ and refer to the analogous constructor in the base class. The class should define a natural order based on the sum of components. Also, override the **toString** method, so the string representation of a color looks like `(red,green,blue)`, where `red`, `green`, `blue` are numbers representing the components.
Additionally, define a class **MyColorCompar** implementing the interface **Comparator<MyColor>**. The constructor of the class takes an enumerator (**enum**) of type **ColComponent** with three constants: RED, GREEN and BLUE. Depending on the passed constant, the **compare** method orders colors according to the appropriate component.
The following program

download *ColorComparing.java*

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class ColorComparing {
    public static void main(String[] args) {
        List<MyColor> list = Arrays.asList(
            new MyColor(  1,  2,  3),
            new MyColor(255,  0,  0),
            new MyColor( 55, 55,100),
            new MyColor( 10,255, 10)
        );
        System.out.println(list);
        Collections.sort(list);
        System.out.println(list);
        Collections.sort(
            list, new MyColorCompar(ColComponent.RED));
        System.out.println(list);
```

```
            Collections.sort(
                list, new MyColorCompar(ColComponent.GREEN));
            System.out.println(list);
            Collections.sort(
                list, new MyColorCompar(ColComponent.BLUE));
            System.out.println(list);
        }
    }
```

should print

```
    [(1,2,3), (255,0,0), (55,55,100), (10,255,10)]
    [(1,2,3), (55,55,100), (255,0,0), (10,255,10)]
    [(1,2,3), (10,255,10), (55,55,100), (255,0,0)]
    [(255,0,0), (1,2,3), (55,55,100), (10,255,10)]
    [(255,0,0), (1,2,3), (10,255,10), (55,55,100)]
```
