Problem 1

Create the interface **Func** — objects of classes implementing it represent functions $\mathbb{R} \to \mathbb{R}$

```java
interface Func {
    double apply(double x);
    static Func compose(Func f, Func g) {
        // ...
    }
}
```

The interface declares one method **apply** (of type **double→double**) and *defines* a static function **compose** which

- takes references to two objects of classes implementing the **Func** interface — `f` and `g`;
- returns an object of a class implementing the same interface which represents the composition of functions represented by `f` i `g`.

Note: composition of two functions, $f \circ g$, is defined as

$$(f \circ g)(x) = f(g(x))$$

The following **main** function

download *InterF.java*

```java
public class InterF {
    public static void main(String[] args) {
        Func f = /* ... */
        Func g = /* ... */
        Func cmp1 = Func.compose(f, g);
        Func cmp2 = Func.compose(g, f);
        Func cmp3 = Func.compose(Func.compose(g, cmp1), f);
        Func cmp4 = Func.compose(g, Func.compose(cmp2, f));
        System.out.println("Res1: " + cmp1.apply(3));
        System.out.println("Res2: " + cmp2.apply(3));
        System.out.println("Res3: " + cmp3.apply(3));
        System.out.println("Res4: " + cmp4.apply(3));
    }
}
```

where

- `f` is the reference to an object of a concrete class which implements **Func** and representing the function $x \mapsto x^2$;

- g is the reference to an object of an anonymous class which implements Func and representing the function $x \mapsto x + 1$

should print

```
Res1: 16.0
Res2: 10.0
Res3: 101.0
Res4: 83.0
```

Problem 2

Define an abstract class Singer which represents singers. Each singer has a name and a number (for example, in a talent competition), which is assigned automatically when an object of the class is being cretaed — you can use a static field incremented in the constructor. The class should have a constructror taking (only) the singer's name (as a String) and the following methods:

- abstract: `abstract String sing()`, which returns the text that is sung by the singer in the competition;
- `public String toString()` returning the information about the singer;
- static: `...loudest(...)` which takes as the argument an array of objects/singers and returns the one whose text of the sung song contains the largest number of capital letters.

In the main function of the testing class Main:

1. create several (minimum 3) objects/singers using anonymous classes which extend Singer. Implementation sets the text of a song which a singer sings in the competition;
2. create an array of singers which consists of objects from the item 1;
3. test the function loudest of class Singer.

The following function main in class Main:

download *SSpiewacyLL.java*

```
public class Main {
    public static void main(String[] args) {
        Singer s1 = new Singer("Martin"){
            /*<- ... */
        };

        Singer s2 = new Singer("Joplin"){
            /*<- ... */
        };

        Singer s3 = new Singer("Houston"){
            /*<- ... */
        };
```

```
            Singer sng[] = {s1, s2, s3};
            for (Singer s : sng) System.out.println(s);
            System.out.println("\n" + Singer.loudest(sng));
        }
    }
```

should print

```
    (1) Martin: Arrivederci, Roma...
    (2) Joplin: ...for me and my Bobby MacGee
    (3) Houston: I will always love youuuu

    (2) Joplin: ...for me and my Bobby MacGee
```

Important: The code of the class **Main** should be changed only in places marked by `/*<- ... */` comments.

Problem 3

Define a (functional) *generic* interface **Transform<T,R>** declaring one method **apply** which takes a **T** and returns an **R**. Define also a class **StrToInt** which implements the interface for **T=String** and **R=Integer**. The implementation of **apply** returns just the length of the string passed as the argument.
In the main class define a static function

```
    private static <T, R>
    void transform(T[] in, R[] out, Transform<T, R> trans) {
        // ...

    }
```

which takes two arrays of equal size, one of references of type **T** and the other of type **R**, and also an object, say trans, implementing the **Transform** interface. The function fills the second array with results of applying the **apply** function invoked on trans to all objects from the first array.
In the **main** function create two arrays of the same size and call the **transform** function passing the arrays and an implementation of the **Transform** interface. Do it in three ways:

- with an object of **StrToInt** type — types of arrays are then **String** and **Integer**;
- with an object of an anonymous class which implements the **Transform** interface in such a way that its **apply** method takes a **String** and returns its first character (as **Character**);
- with a lambda which transforms strings into the same strings but in upper case.

The following program:

download *GenTrans.java*

```
    import java.util.Arrays;
```

```java
@FunctionalInterface
interface Transform<T, R> {
    R apply(T s);
}

 // class StrToInt

public class GenTrans {
    private static <T, R>
    void transform(T[] in, R[] out, Transform<T, R> trans) {
        // ...
    }

    public static void main (String[] args) {
        String[]  sin = {"Alice", "Sue", "Janet", "Bea"};
        System.out.println(Arrays.toString(sin) + '\n');

        Integer[] iout = new Integer[sin.length];
        transform(sin, iout, /* ... */);
        System.out.println(Arrays.toString(iout));

        Character[] cout = new Character[sin.length];
        transform(sin, cout, /* ... */);
        System.out.println(Arrays.toString(cout));

        String[] sout = new String[sin.length];
        transform(sin, sout, /* ... */);
        System.out.println(Arrays.toString(sout));
    }
}
```

should print

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

[5, 3, 5, 3]
[A, S, J, B]
[ALICE, SUE, JANET, BEA]
```
