**Notes**

- Previous exam assignments and solutions are not allowed.
- Use NetBeans to edit, compile, and execute programs.
- Create a **Surname_firstname.zip** file that contains only the *.java* source code.
- The zip file has to be uploaded using the form indicated by the teacher.
- At the end of the exam, the teacher will show you a possible solution. Then, if you want, you can withdraw your zip file.

**Exercise**

A *SortingTask* is a sorting operation of a list of integers to be performed at a specific instant in the future. Once a *SortingTask* has been executed, the sorted list is stored in a file and it is possible to retrieve the result of the operation through the file path. Create a *SortingTask* class that has at least the following methods and constructor:

- *SortingTask(List<Integer> list, long scheduledTime, int allowedDelay:* creates a *SortingTask* where *list* is the list of integers to be sorted, *scheduledTime* is the instant in which the task must be executed expressed in milliseconds according to the convention of *System.currentTimeMillis()*, and *allowedDelay* is the maximum allowed delay relative to its *scheduledTime.*
- *void execute()*: sorts the list and stores the result in a text file named *scheduledTime*.txt.
- *String getResult() throws Exception:* returns the path of the file that contains the sorted list; the method returns the result immediately, if possible. Otherwise, the method blocks the calling thread until the task has been completed. If the operation is not successful (for example because it is not possible to save the ordered list in the given file or if the delay is larger than the maximum allowed delay) the method does not return anything to the caller and just throws an exception.
- *boolean isComplete():* returns *true* if the task has been successfully carried out, *false* otherwise.
- *int getAllowedDelay():* returns the maximum allowed delay.
- *long getScheduledTime():* returns the time when the task has to be executed according to the convention of *System.currentTimeMillis()*.
- *void setException(Exception e):* set the exception that will be thrown in *getResult()* in case of problems.

Create a *FutureExec* class that allows executing sorting tasks in the future. A *FutureExec* has its own execution flow. The execution flow is unique for all the managed tasks. The execution of a task can be delayed if the previous task takes too much time to complete. The *FutureExec* class provides a *void add(SortingTask s)* method that can be called to add a new task to the managed ones. The *FutureExec* executes the managed tasks according to their *scheduledTime*. A *SortingTask* can be subject to some delay because a previous task is still running. In this case, the maximum allowed delay is equal to *allowedDelay*. If the delay is larger than *allowedDelay*, the task is not executed and *its getResult()* method will throw an exception when called.

The two classes can be used as shown in the following example:

```java
public class Prova {
    private static List<Integer> creaLista() {
        List<Integer> l = new ArrayList<Integer>();
        for(int i=0; i<1000000; i++)
            l.add((int)(Math.random()*10000));
        return l;
    }
    public static void main(String[] args) throws Exception {
        FutureExec ef = new FutureExec();
        SortingTask c1 = new SortingTask(creaLista(), System.currentTimeMillis() + 10000, 1000);
        ef.add(c1);
        System.out.println("First SortingTask added");
        SortingTask c2 = new SortingTask(creaLista(), System.currentTimeMillis() + 10010, 100);
        ef.add(c2);
        System.out.println("Second SortingTask added");

        String r1 = c1.getResult();
        System.out.println("File path: " + r1);

        String r2 = c2.getResult();
        System.out.println("File path: " + r2);

    }
}
```

Possible output:

```
First SortingTask added
Second SortingTask added
…
<Ten seconds>
…
File path: /Users/vecchio/Documents/prg-compiti-2023/2023-02-
03/1675367603158.txt
Exception in thread "main" esercizio1.TooMuchDelay
     at esercizio1.FutureExec.run(FutureExec.java:38)

      at java.base/java.lang.Thread.run(Thread.java:834)
```