I. Create class ***Letters***, which will be used to run threads in parallel. Threads will print every second the letters given in the string (an object of type ***String)*** passed to the class constructor. After creating the ***Letters*** object in the ***main*** method, start all created threads in which letters are to be printed. After running all threads, you should sleep the ***main*** method for 5 seconds (***sleep*** method in the ***Thread*** class). After pausing, you must end all threads printing letters. After sleeping, interrupt the running threads from the Letters class.

For the following program:

```
public class Main {

    public static void main(String[] args) throws InterruptedException {
        Letters letters = new Letters("ABCD");
        for (Thread t : letters.getThreads())
            System.out.println(t.getName());
            /*<- all threads should be started here */

            Thread.sleep(5000);

            /*<- all threads should be interrupted here */
            System.out.println("\nThe program has finished");
    }
}
```

Oczekiwanym wynikiem jest:

```
Thread A
Thread B
Thread C
Thread D
(after 5 seconds)
The program has finished
```

After completing the code in the indicated places, the expected sample result is:

```
Thread A
Thread B
Thread C
Thread D
ACDBDBACACDBCBDA
The program has finished
```

Hints:

- Do not use *System.exit*
- When defining the ***run()*** method, it's a good idea to use a lambda expression

II. Create ***StringTask*** class that simulates long-running calculations (we treat the concatenation of strings as long-term calculations). The constructor of the ***StringTask*** class receives two arguments. The first is the string that will be subject to concatenation, and the second argument indicates the number of how many times the string passed to be duplicated.

The class should implement the ***Runnable*** interface. The ***run()*** method performs duplicating the String.

For the purpose of the task, to simulate long-term calculations, duplication of the String should be done using the '+' operator used for objects of type ***String***.

We treat the object of ***StringTask*** as a thread that can run in parallel with others.

Possible task states are (use enumerated type):

- ***CREATED*** – task has been created but has not yet been started
- ***RUNNING*** – task has been started and is carried out in a separate thread
- ***ABORTED*** – the job was interrupted during execution
- ***READY*** – task completed successfully

Define the methods in the StringTask class:

- `public String getResult()` – returns the result of concatenation
- `public TaskState getState()` – returns the status of the task
- `public void start()` – runs the task in a separate thread
- `public void abort()` – interrupts thread execution
- `public boolean isDone()` – returns ***true*** if task completed/interrupted, otherwise ***false***

For the following program:

```
public class Main {

  public static void main(String[] args) throws InterruptedException {
    StringTask task = new StringTask("A", 70000);
    System.out.println("Task " + task.getState());
    task.start();
    if (args.length > 0 && args[0].equals("abort")) {
    /*<- code that interrupts the task after a second and runs it in a
       separate thread */
    }
    while (!task.isDone()) {
      Thread.sleep(500);
      switch(task.getState()) {
        case RUNNING: System.out.print("R."); break;
        case ABORTED: System.out.println(" ... aborted."); break;
        case READY: System.out.println(" ... ready."); break;
        default: System.out.println("uknown state");
      }
    }
    System.out.println("Task " + task.getState());
    System.out.println(task.getResult().length());
  }
}
```

The program execution result (without argument) should be similar to the following:

```
Task CREATED
R.R.R.R.R.R.R.R. ... ready.
Task READY
70000
```

If you run the program with argument "*abort*", the execution result should be similar to:

```
Task CREATED
R. ... aborted.
Task ABORTED
31700
```