

- I. Stwórz klasę **Letters**, która posłuży do równoległego uruchamiania wątków, wypisujących co sekundę litery podane w napisie (obiekt typu **String**) przekazanym do konstruktora klasy. Po stworzeniu obiektów klasy **Letters** w metodzie **main** należy wystartować wszystkie utworzone wątki, w których mają być wypisywane litery. Po uruchomieniu wszystkich wątków należy wstrzymać działanie metody **main** na 5 sekund (metoda **sleep** w klasy **Thread**). Po wstrzymaniu działania należy zakończyć działanie wszystkich wątków wypisujących litery.

Dla poniższego programu:

```
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());
        /*<- w tym miejscu należy uruchomić
           wszystkie kody w wątkach
        */

        Thread.sleep(5000);

        /*<- tu należy zapisać
           fragment, który kończy działanie wątków, wypisujących
           litery
        */
        System.out.println("\nProgram skończył działanie");
    }
}
```

Oczekiwanym wynikiem jest:

```
Thread A
Thread B
Thread C
Thread D
(po 5 sekundach)
Program skończył działanie
```

Natomiast po uzupełnieniu kodu we wskazanych miejscach oczekiwanym wynikiem jest na przykład (uzyskane wyniki mogą nieznacznie różnić się od prezentowanych poniżej):

```
Thread A
Thread B
Thread C
Thread D
ACDBDBACACDBCDBA
Program skończył działanie.
```

Wskazówki:

- Nie wolno stosować *System.exit*
- Przy definiowaniu metody **run()** warto zastosować lambda-wyrażenie

II. Należy stworzyć klasę ***StringTask***, symulująca długotrwałe obliczenia (w przypadku tego zadania jako długotrwałe obliczenia potraktujemy sobie konkatenację napisów).

Konstruktor klasy ***StringTask*** otrzymuje dwa argumenty. Pierwszym z nich jest napis, który będzie podlegał konkatenacji (powieleniu), zaś drugi argument wskazuje liczbę ile razy przekazany napis ma zostać powielony.

Klasa powinna implementować interfejs ***Runnable***, a w dostarczonej przez interfejs, metodzie ***run()*** wykonywane jest powielenie napisu. Dla celów zadania, aby zasymulować długotrwałe obliczenia powielenie napisu powinno się odbywać za pomocą operatora '+' stosowanego wobec obiektów typu ***String*** (taki sposób konkatenacji jest kosztowny czasowo).

Obiekt klasy ***StringTask*** traktujemy jako wątek, który może się wykonywać równolegle z innymi.

Możliwe stany zadania to (należy zastosować typ wyliczeniowy):

- ***CREATED*** – zadanie zostało utworzone, jednak jeszcze nie zostało uruchomione
- ***RUNNING*** – zadanie zostało uruchomione i wykonuje się w osobnym wątku
- ***ABORTED*** – zadanie zostało przerwane w trakcie wykonywania
- ***READY*** – zadanie zostało zakończone pomyślnie

W klasie ***StringTask*** należy zdefiniować metody:

- **`public String getResult()`** – zwraca wynik konkatenacji
- **`public TaskState getState()`** – zwraca stan zadania
- **`public void start()`** – uruchamia zadanie w odrębnym wątku
- **`public void abort()`** – przerywa wykonanie kodu zadania i działanie wątku
- **`public boolean isDone()`** – zwraca ***true***, jeśli wykonanie zadania zakończyło się lub zostało przerwane, w przeciwnym przypadku zwrócona zostaje wartość ***false***

Dla poniższego programu:

```
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")) {
            /*-> tu zapisać kod przerywający działanie tasku po sekundzie
               i uruchomic go w odrębnym wątku
            */
        }
        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("unknown state");
            }
        }
        System.out.println("Task " + task.getState());
        System.out.println(task.getResult().length());
    }
}
```

Efekt wykonania programu (bez przekazanego argumentu) powinien być zbliżony do poniższego:

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

W przypadku uruchomienia programu z argumentem "*abort*" efekt powinien być zbliżony do:

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