Zad 1

Uruchamianie kodów w wątkach i kończenie wątków

Zbudować klasę Letters, która posłuży do równoleglego uruchamiania kodów, wypisujących co sekundę litery podane w napisie przekazanym do konstruktora klasy.
Po stworzeniu obiektu klasy Letters w metodzie main(...) klasy Main wystartować wszystkie wątki, w których mają być wypisywane podane litery.
Po wstrzymaniu działania metody main(...) na 5 sekund - zakończyć działanie wszystkich kodów, wypisujących litery.

Uruchomienie 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());
        /*<- tu uruchomić
             wszystkie kody w wątkach
         */

        Thread.sleep(5000);

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

    }
```

powinno (w tej postaci) wypisać:
Thread A
Thread B
Thread C
Thread D

(po 5 sekundach)
Program skończył działanie

Natomiast po uzupełnieniu kodu w miejscah wskazanych przez /*<- */ - coś w rodzaju:

Thread A
Thread B
Thread C
Thread D
ACDBDBACACDBCBDA
Program skończył działanie.

Uwaga 1: modyfikacje klasy Main są dopuszczalne tylko w miejscach wskazanych przez /*<- */
Uwaga 2: nie wolno stosować System.exit
Uwaga 3: warto przy definiowaniu metody run() zastosować lambda-wyrażenie

Zad 2

Uruchamianie i zatrzymywanie równoległego działania kodów

Zbudować klasę StringTask, symulująca długotrwałe obliczenia, tu polegające na konkatenacji napisow.
Konstruktor klasy otrzymuje jako argument napis do powielenia oraz liczbę oznaczającą ile razy ten napis ma być powielony.
Klasa winna implementować interfejs Runnable, a w jej metodzie run() wykonywane jest powielenia napisu, przy czym to powielenia ma się odbywac za pomoca operatora '+' stosowanego wobec zmiennych typu String (to właśnie długotrwała operacja). **Użycie '+' jest warunkiem obowiązkowe**.

Obiekt klasy StringTask traktujemy jako zadanie, które może się wykonywać równolegle z innymi. Możliwe stany zadania to:

- CREATED - zadanie utworzone, ale nie zaczęło się jeszcze wykonywać,
- RUNNING - zadanie się wykonuje w odrebnym wątku
- ABORTED - wykonanie zadania zostało przerwane
- READY - zadanie zakończyło się pomyślnie i sa gotowe wyniki.

W klasie StringTask zdefiniować metody:

- public String getResult() - zwracającą wynik konkatenacji
- public TaskState getState() - zwracającą stan zadania
- public void start() - uruchamiającą zadanie w odrębnym watku
- public void abort() - przerywającą wykonanie kodzu zadania i działanie watku
- public boolean isDone() - zwracająca true, jeśli wykonanie zadania się zakończyło normalnie lub przez przerwanie, false w przeciwnym razie

Poniższy kod 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")) {
    /*<- 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("uknown state");
      }

    }
    System.out.println("Task " + task.getState());
    System.out.println(task.getResult().length());
  }
```

```
}
```

uruchominy bez argumentu powinien wyprowadzić coś w rodzaju:
Task CREATED
R.R.R.R.R.R.R.R. ... ready.
Task READY
70000

a uruchomiony z argumentem "abort" może wyprowadzić:
Task CREATED
R. ... aborted.
Task ABORTED
31700

Uwaga 1. Plik Main.java może być modyfikowany tylko w miejscu oznaczonym /*<- */
Uwaga 2. Nie wolno uzywac metody System.exit(...)

Zad 3

Zadanie: Towary (8 punktów)

Kod, działający w wątku A czyta z pliku ../Towary.txt informacje o towarach w postaci:

id_towaru waga

tworzy obiekty klasy Towar, zawierające przeczytane informacje oraz wyprowadza na konsolę informacje o liczbie utworzonych obiektów. Informacja ma być wyprowadzana co 200 obiektów w postaci:

utworzono 200 obiektów
utworzono 400 obiektów
utworzono 600 obiektów
itd.

Kod działający **równolegle** w innym wątku (B) sięga po te obiekty, sumuje wagę towarów i wyprowadza na konsolę informację o przeprowadzonym sumowaniu co 100 obiektów np.:
policzono wage 100 towarów
policzono wage 200 towarów
policzono wage 300 towarów
itd.
Na końcu podaje sumaryczną wagę wszystkich towarów.

Uwagi:

1. Plik powinien zawierać co najmniej 10 tys. opisów towarów (należy sobie go wygenerować programistycznie, ale **na boku, nie w tym programie**), nazwa pliku (wraz ze ścieżką) jest obowiązkowa , **proszę nie dołączać tego pliku do projektu**..
2. Zapewnić synchronizację i koordynację pracy obu wątków.
3. Forma wydruku na konsoli jest obowiązakowa
4. Wszystkie klasy w programie winny być publiczne (w różnych plikach)
5. Wykonanie programu winno zaczynać się w metodzide main() obowiązakowej klasy Main.
6. Plik winien znajdować się w nadkatalogu projektu (czyli w katalogu workspace'u)

Za niespełnienie tych warunków nie będą przyznawane punkty.

Zad 4

Napisać program Author-Writer z wykładu przy użyciu blokujących kolejek.
Jako argumenty program otrzymuje napisy, które co sekundę ma generować Author.
Writer ma je wypisywać na konsoli.

Klasa Main ma następująca postać i nie można jej modyfikować:

```
public class Main {
  public static void main(String[] args) {
    Author autor = new Author(args);
    new Thread(autor).start();
    new Thread(new Writer(autor)).start();
  }
}
```