Threading

* Exercise #1:

Rewrite the following program to use an appropriate thread pool:

```
public class Threads {

      public static void main(String[] args) {

            for (int i = 1; i <= 10; i++) {

                  new Task("Task " + i).start();

            }

      }

}
```

And here is the code of the `Task` class:

```
class Task extends Thread {

      Task(String name) {

            super(name);

      }

      public void run() {

            System.out.println(getName() + " is running");

      }

}
```

* Exercise #2:

The `Thread` class has a static method that can be used to retrieve all live threads in the Java Virtual Machine:

`static Map<Thread,StackTraceElement[]>` **`getAllStackTraces`**`()`

Write a program that lists all live threads in the JVM with their name, state, priority and daemon status. Here's a sample output:

```
Finalizer - WAITING - 8 - Daemon

Attach Listener - RUNNABLE - 5 - Daemon

Signal Dispatcher - RUNNABLE - 9 - Daemon

Reference Handler - WAITING - 10 - Daemon
```

```
main - RUNNABLE - 5 - Normal
```

* Exercise #3:

Given the following three Thread classes.

Thread #1:

```
public class ProcessorMonitor extends Thread {
	public void run() {
		while (true) {
			System.out.println("Monitoring CPU...");
			try {
				Thread.sleep(500);
			} catch (InterruptedException ex) {
				ex.printStackTrace();
			}
		}
	}
}
```

Thread #2:

```
public class FanMonitor extends Thread {
	public void run() {
		while (true) {
			System.out.println("Monitoring cooling fan...");
			try {
				Thread.sleep(1000);
			} catch (InterruptedException ex) {
				ex.printStackTrace();
			}
		}
	}
}
```

Thread #3:

```
public class DiskMonitor extends Thread {
      public void run() {
            while (true) {
                  System.out.println("Monitoring hard disk drive...");
                  try {
                        Thread.sleep(600);
                  } catch (InterruptedException ex) {
                        ex.printStackTrace();
                  }
            }
      }
}
```

You are required to update these classes so they can be grouped into a `ThreadGroup` and you can stop all the threads by simply calling:

```
       group.interrupt();
```

Also write a test program to execute 3 threads in a group and stop them after 10 seconds.

* Exercise #4:

Given the following class:

```
public class SharedNumber {
      private int number;
      public SharedNumber(int initialValue) {
            this.number = initialValue;
      }
      public int increment() {
            return ++number;
      }
      public int decrement() {
            return --number;
      }
      public int add(int delta) {
            return (number + delta);
        }
        public void set(int newValue) {
            this.number = newValue;
        }
        public int getValue() {
            return number;
        }
}
```

Update this class using an atomic variable so its instance can be used safely by multiple threads.

Also write a test program that simulates 20 threads concurrently update (increment) a shared instance of this class. Each thread updates 20 times. Compare the final value of the SharedNumber object between 2 cases: with atomic variable and without atomic variable.

* Exercise #5:

Given the following Dictionary class:

```
import java.util.*;

public class Dictionary {
    private Map<String, String> map = new HashMap<>();
    public Dictionary() {
    }
    public void put(String word, String meaning) {
        map.put(word, meaning);
    }
    public String lookFor(String word) {
        return map.get(word);
    }
}
```

This class will be used as a data structure for a multi-threaded dictionary application. You are required to update this class in order to make its instance can be safely used by multiple threads: many users can use the dictionary to lookup the meanings of words, and only one user can update (add words) to the dictionary.

Next, write a test program that initially loads the dictionary with the following data (word: meaning):

```
      agree: have the same opinion about something

      business: a person's regular occupation, profession, or trade

      change: make or become different

      doctor: a qualified practitioner of medicine; a physician

      energy: the strength and vitality required for sustained physical or
mental activity

      freedom: the state of not being imprisoned or enslaved

      grateful: feeling or showing an appreciation of kindness; thankful

      healthy: in good health

      interest: the state of wanting to know or learn about something or
someone

      justice: just behavior or treatment
```

In this program, create 10 threads that repeatedly look up for the meanings of the words in the above list. Each thread looks up the word randomly and sleeps about 1 second before looking up next word.

And there's only one writer thread that updates the dictionary by adding the following words:

```
      kind: a group of people or things having similar characteristics

      lucky: having, bringing, or resulting from good luck

      memory: something remembered from the past; a recollection

      news: information not previously known to someone

      opportunity: a set of circumstances that makes it possible to do
something

      program: a planned series of future events, items, or performances

      quiet: making little or no noise

      realtime: the actual time during which a process or event occurs

      student: a person who is studying at a school or college

      unique: being the only one of its kind; unlike anything else
```

The writer thread updates the dictionary every 3 seconds, until reaching the end of the above list.

* Exercise #6:

Given the following class provides a method that checks if an integer number is a prime number or not:

```
public class Prime {

      public static boolean isPrime(int number) {

            int maxLookup = (int) Math.sqrt(number) + 1;

            for (int i = 2; i < maxLookup; i++) {

                  if (number % i == 0) {

                        return false;

                  }

            }

            return true;

      }

}
```

and here is a code snippet that prints all the prime numbers in the range 2…100:

```
for (int i = 2; i < 100; i++) {

      if (Prime.isPrime(i)) {

            System.out.println(i);

      }

}
```

Using Fork/Join framework, develop a program that finds all prime numbers in the range 2…20_000_000 (two to twenty millions). The program should divide the task into sub tasks which are executed concurrently.

* Exercise #7:

Develop a kind of producer-consumer program using ArrayBlockingQueue to simulate a web server:

- **Producer**: when a client makes a request to the server, the request is placed on a queue. The queue is limited for only 1024 requests. There's only one producer thread in the server.

- **Consumers**: takes requests from the queue, processes them and sends responses to the client. There are 20 consumer threads in the server.

NOTE: this program is just a simulation to help you understand the concept and technique. You don't have to implement a real web server.

* Exercise #8:

Write a program that simulates how a web crawler works using `LinkedBlockingQueue`.

- **Producer**: finds hyperlinks from a webpage and place the links on a queue. The queue is unbounded.

- **Consumers**: takes links from the queue and download web pages from the links.

There's only one producer thread and 10 consumer threads.

NOTE: this program is just a simulation to help you understand the concept and technique. You don't have to implement a real web crawler.