Lab 1: Create Threads in Java

There are two ways to create threads:

a) Extend from the Thread class and override the run() method.
b) Implement the runnable interface.

In this lab, we will be looking at these two ways of creating threads in Java, how we could set the get the name of a thread we created, and the difference between daemon and non-daemon thread.

A. Extend from the Thread Class

As shown below is an example of how you can use the extends keyword to extend from the Thread class and override the run() method. We define what we need the thread to execute inside the run() method.

```
// Example 1
class MyThread extends Thread {
        public void run() {
                // Instructions that this thread will execute (task assigned to this thread).
                System.out.println("Creating thread using subclass.");
                System.out.println("This is example 1.");
        }
}

public class TestClass1 {
        public static void main ( String[] args ) {
                MyThread t = new MyThread();
                // Remember to call the start() method or else the thread will not start.
                t.start();
        }
}
```

Note:
- Although the instructions from different threads will be able to run concurrently, the instructions within the run() method will still be executed sequentially.

Note:
- Inheritance allows code defined in one class to be reused in other classes.
- You can have a general superclass, and then extend it with more specific subclasses.
- The subclass can access the instance variables and methods defined by the superclass.
- The subclass can also choose to override the methods defined in the superclass.

B. Implement Runnable Interface

As shown below is an example of how you can implement the runnable interface and define what you need the thread to execute using the run() method. Please take note on the key difference when creating the thread object. In this case, we need to first create the runnable object, and pass it to the Thread class constructor to create the thread object that we want. After that, we still need to call the start() method.

```
// Example 2
class MyRunnable implements Runnable {
        public void run() {
                // Instructions that this thread will execute (task assigned to this thread).
                System.out.println("Creating thread using runnable interface.");
                System.out.println("This is example 2.");
        }
}

public class TestClass2 {
        public static void main ( String[] args ) {
                // Create a runnable object and pass it to the Thread class constructor.
                MyRunnable mr = new MyRunnable();
                Thread t = new Thread(mr);
                // Call the start() method of the thread object, not the runnable object.
                t.start();
        }
}
```

Note:

- Interface is used to define method(s) that a class must define.

Exercise 1

a) Try to combine the code shown in example 1 and 2 above into one. In other words, demonstrate how we can use the two different ways of creating threads in the same piece of code.
b) Describe the difference(s) between the code you wrote for (a) with the code shown below (you might have to run the code you wrote for (a) several times).

```
public class ExClass1 {
        public static void main ( String[] args ) {
                System.out.println("Creating thread using subclass");
                System.out.println("This is example 1.");
                System.out.println("Creating thread using runnable interface");
                System.out.println("This is example 2.");
        }
}
```

C. Set and Get the Name of a Thread

Sometimes, if you want to know what thread is doing what (e.g. for debugging purpose), we can use the getName() method to get the name of a thread object. But before that, we need to use the static currentThread() method to get the refence to the thread object that is currently running. As shown below is an example of how you can make use of the getName() method.

```
// Example 3
class MyThread implements Runnable {
        public void run() {
                // Instructions that this thread will execute (task assigned to this thread).
                System.out.println("Creating Thread with Subclass Method");
                System.out.println("This is done by " + Thread.currentThread().getName());
        }
}

public class TestClass3 {
        public static void main ( String[] args ) {
                Thread t0 = new Thread(new MyThread());
                Thread t1 = new Thread(new MyThread());
                // Remember to call the start() method or else the thread will not start.
                t0.start();
                t1.start();
        }
}
```

In this case, since we did not specify the name of a thread, the default naming convention will be used (Thread-x, where x is the thread number). We can of course set the name of a thread object by using the setName() method, so that it will be easier for us to recognize them. Because we are now trying to set the name of a thread object, we can only do it after the thread object is created.

```
// Example 4
class MyThread implements Runnable {
        public void run() {
                // Instructions that this thread will execute (task assigned to this thread).
                System.out.println("Creating Thread with Subclass Method");
                System.out.println("This is done by " + Thread.currentThread().getName());
        }
}

public class TestClass4 {
        public static void main ( String[] args ) {
                Thread t0 = new Thread(new MyThread());
                Thread t1 = new Thread(new MyThread());
                // Set the name of thread t0 and thread t1.
                t0.setName("Subclass Thread");
                t1.setName("Runnable Thread");
                // Remember to call the start() method or else the thread will not start.
                t0.start();
                t1.start();
        }
}
```

Note:
- Always try to use meaningful name for everything that you create (e.g. variables, etc).

Exercise 2

a) Create the following class:

```
class PrintNameAndValue implements Runnable {
        public void run() {
                for(int x = 1; x <= 10; x++) {
                        System.out.println("By " + Thread.currentThread().getName() + ", x is " + x);
                }
        }
}

public class ExClass2 {
        public static void main ( String[] args ) {
                PrintNameAndValue pnav = new PrintNameAndValue();
                Thread t0 = new Thread(pnav);
                Thread t1 = new Thread(pnav);
                Thread t2 = new Thread(pnav);
                t0.start();
                t1.start();
                t2.start();
                System.out.println("This is the end of main() method.");
        }
}
```

b) Run the code above for several times and describe what you can observe from the output.
c) Modify the code above so that each thread can run for different number of iterations.
d) Modify the code above so that each thread will sleep for 1s after every single print out.

D. Daemon and Non-daemon Thread

The key difference between a daemon thread and non-daemon thread, is that the daemon thread will terminate automatically if the parent thread that created it terminates, whereas non-daemon will just continue to run. In summary, the daemon status is affected by the parent thread, and the two differ in what would happen the parent thread terminates. It can affect the state of the process, because as long as there is one thread that is still alive (even though it is not running), the process that contains the thread will stay alive.

Which type a thread will take depends on the thread that created it. For example, if the parent thread is a non-daemon thread, then the thread that it spawned will also be a non-daemon thread. But again, we can of course try to set the type by using the setDaemon() method.

In the following example, we first create a class that will increase and print numbers to the console continuously without stopping. Because the thread that runs the main() method (also known as the main thread) is a non-daemon thread, when we create the thread in the main() method, the thread will also be a non-daemon thread. In order to change it to a daemon thread, we use the setDaemon() method and set the parameter to true. This will mark the thread as a daemon thread. However, it is important to take note that we must call the setDaemon() method before the start() method. Once a thread has started to run, we can no longer change the type.

After we have set the thread, we proceed to call the start() method, so that the thread can start to run (eligible to run, still need to wait for the scheduler to pick it). We then put the main thread to sleep for 5 seconds. While the main thread is sleeping, the daemon thread will continue to print number to the console. Try to run the code several times, and you should be able to see that the last number printed to the console will not be the same. The reason is that the time allocated for the thread to run is not always the same. It depends on the scheduler as well as the scheduling algorithm adopted by the system.

```
// Example 5
class MyThread implements Runnable {
        private int x = 0;
        public void run() {
                //Will just continue to run without stopping.
                while (true) {
                        System.out.println("By " + Thread.currentThread().getName() + ", x is " + x);
                        x = x + 1;
                }
        }
}

public class TestClass5 {
        public static void main ( String[] args ) {
                Thread t0 = new Thread(new MyThread());
                // Change the daemon status of thread t0.
                t0.setDaemon(true);
                t0.start();
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException e) {
                }
                System.out.println("This is the end of main() method.");
        }
}
```

<u>Exercise 3</u>

a) Change the parameter for the setDaemon() method in example 5 to false.
b) Run the code, observe the output, and describe the difference(s) between daemon thread and non-daemon thread.