Java Programming
Lab FP 4

Functional Interfaces

1. Lab objectives

This lab is designed to help you understand functional interfaces using threads as an example

2. Lab Setup

For this lab, you should probably create a new project and new main class.

3. Lab Solutions

The solution for this lab is in the code repository.

3. The old fashioned way

The old standard way of creating threads was to use either a custom Thread object with a run() function that extended the thread class or to create a Thread object but pass it an object in its constructor that implemented the Runnable interface.

In this section, you get to do both. First create a custom thread object as shown below:

```
class MyThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("MyThread" + Thread.currentThread() + "count=" + i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

And then create a class that implements the Runnable interface - ie. a functional interface.

```
class MyOtherThread implements Runnable {

    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("MyOtherThread" + Thread.currentThread()+ "count=" +
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

    }
}
}
```

Now these classes can be run as threads using the standard Java model. One of the things to note is all of the boiler plate code needed to package up the chunk of code in the run() method so that it can be run as a thread.

```
public static void main(String[] args) {
    MyThread t = new MyThread();
    Thread ot = new Thread(new MyOtherThread());
    ot.start();
    t.start();

}
```

4. Using a functional interface

Lambda functions are inherently functional interfaces. To get rid of all of the boilerplate code, we can just pass the body of a the run() method to a thread object.

To do that, create a Lambda function as shown below where the body is just cut and pasted from the run() method in the previous example. In the first case it is assigned to a variable of type Runnable so that it easy to read, but this step is unnecessary.

```
public static void main(String[] args) {
    Runnable r = () -> {
        for (int i = 1; i <= 5; i++) {
            System.out.println("MyThread" + Thread.currentThread() + "count=" + i)
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    };
```

Now you can create the threads directly using the variable or the Lambda function directly. Normally, when the code is just a couple of lines, we pass the Lambda directly to the Thread.

```
        // Using r for readability
        Thread t1 = new Thread(r);
        // passing the function directly
        Thread t2 = new Thread(() -> {
            for (int i = 1; i <= 5; i++) {
                System.out.println("MyThread" + Thread.currentThread() + "count=" + i);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        t1.start();
        t2.start();

    }
```

5. Using an executor service

We can eliminate even the boilerplate code above by using and executor service. This object creates a queue of Runnable objects and runs each of them as a thread. Defining the variable “r” exactly like the previous section , we can replace all of the code about with the following:

```
        //create a new executor service
        ExecutorService e = Executors.newFixedThreadPool(5);
        e.execute(r);
        e.execute(r);
        e.shutdown();
```

End of Lab