Setting-up Threads

This lesson discusses how threads can be created in Java.

Creating Threads

To use threads, we need to first create them. In the Java language framework, there are multiple ways of setting up threads.

Runnable Interface

When we create a thread, we need to provide the created thread code to execute or in other words we need to tell the thread what *task* to execute. The code can be provided as an object of a class that implements the `Runnable` interface. As the name implies, the interface forces the implementing class to provide a `run` method which in turn is invoked by the thread when it starts.

The runnable interface is the basic abstraction to represent a logical task in Java.

```
1  class Demonstration {
2      public static void main( S
3          Thread t = new Thread(
4
5              public void run() 
6                  System.out.pri
7              }
8          });
9          t.start();
10     }
11 }
```

We defined an anonymous class inside the `Thread` class's constructor and an instance of it is instantiated and passed into the Thread object. Personally, I feel anonymous classes decrease readability and would prefer to create a separate class implementing the Runnable interface. An instance of the implementing class can then be passed into the Thread object's constructor. Let's see how that could have been done.

```
class Demonstration {
    public static void main( S
        ExecuteMe executeMe = 
        Thread t = new Thread(
        t.start();
    }
}

class ExecuteMe implements Runn

  public void run() {
    System.out.println("Say He
  }

}
```

Subclassing Thread class

The second way to set-up threads is to subclass the `Thread` class itself as shown below.

```
class Demonstration {
    public static void main( String args[] ) throws Exception {
        ExecuteMe executeMe = new ExecuteMe();
        executeMe.start();
```

```
        executeMe.join();

    }
}

class ExecuteMe extends Thread {

  @Override
  public void run() {
    System.out.println("I ran after extending Thread class");
  }

}
```




The con of the second approach is that one is forced to extend the `Thread` class which limits code's flexibility. Passing in an object of a class implementing the `Runnable` interface may be a better choice in most cases.

In next lesson, we'll study ways of manipulating threads