

# 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( String args[] ) {
3         Thread t = new Thread(new Runnable() {
4
5             public void run() {
6                 System.out.println("Say Hello");
7             }
8         });
9         t.start();
10    }
11 }
```

Run Save Reset

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.

```
1 class Demonstration {
2     public static void main( String args[] ) {
3
4         ExecuteMe executeMe = new ExecuteMe();
5         Thread t = new Thread(executeMe);
6         t.start();
7     }
8 }
9
10 class ExecuteMe implements Runnable {
11
12     public void run() {
13         System.out.println("Say Hello");
14     }
15 }
16
17
```

Run Save Reset

## Subclassing Thread class#

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

```
1 class Demonstration {
2     public static void main( String args[] ) throws Exception {
3         ExecuteMe executeMe = new ExecuteMe();
4         executeMe.start();
5         executeMe.join();
6     }
7 }
8
9
10 class ExecuteMe extends Thread {
11
12     @Override
13     public void run() {
14         System.out.println("I ran after extending Thread class");
15     }
16 }
17
18
```

Run Save Reset

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