Setting-up Threads

This lesson discusses how threads can be created in Java.

We'll cover the following

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` (https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) 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.

```
class Demonstration {
```

```
class Demonstration {
    public static void main( String args[] ) {
        Thread t = new Thread(new Runnable() {

            public void run() {
                System.out.println("Say Hello");
            }
        });
        t.start();
    }
}
```

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( String args[] ) {

        ExecuteMe executeMe = new ExecuteMe();
        Thread t = new Thread(executeMe);
        t.start();
    }
}

class ExecuteMe implements Runnable {

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

}

```

Subclassing Thread class#

The second way to set-up threads is to subclass the `Thread` (https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html) class itself as shown below.

```java
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` (https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html) class which limits code's flexibility. Passing in an object of a class implementing the `Runnable` (https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) interface may be a better choice in most cases.

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

Interviewing soon? We've partnered with Hired so that companies apply to yo
utm_source=educative&utm_medium=lesson&utm_location=CA&utm_camp

Back

Next Steps

Next

Basic Thread Handling

✓ Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/setting-up-threads__java-concurrency-reference__java-multithreading-for-senior-engineering-interviews)