Lab CC 5: Shared Objects

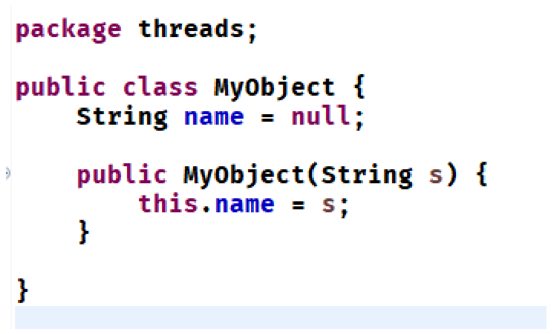
Objectives

In this lab, you will create several versions of an object. In one version, different threads have their own copies, and in the second version, different threads share a single copy of the object

Part One: Non-shared Objects

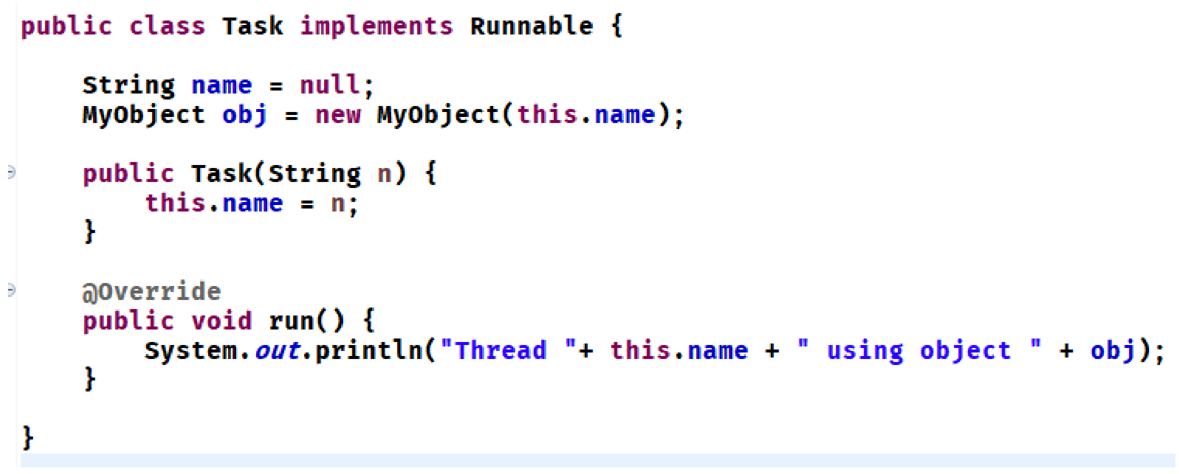
Step 1 Create the MyObject class

1. This object is not a thread, it's just an arbitrary object that occupies memory on the heap when instantiated.



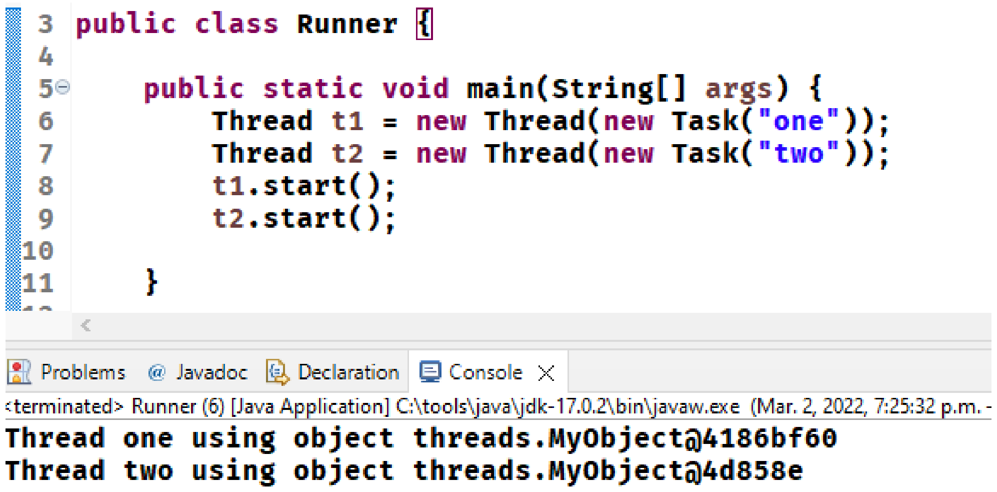
```java
package threads;

public class MyObject {
    String name = null;

    public MyObject(String s) {
        this.name = s;
    }

}
```

2. Create the runnable Task class that will use MyObject. In this first version, each Task object creates its own MyObject instance. These objects are not shared across threads



```java
public class Task implements Runnable {

    String name = null;
    MyObject obj = new MyObject(this.name);

    public Task(String n) {
        this.name = n;
    }

    @Override
    public void run() {
        System.out.println("Thread "+ this.name + " using object " + obj);
    }

}
```

3. Note that the output statement prints the name of Task object and the address of the MyObject object.
4. Now create the runner class which creates and starts the threads



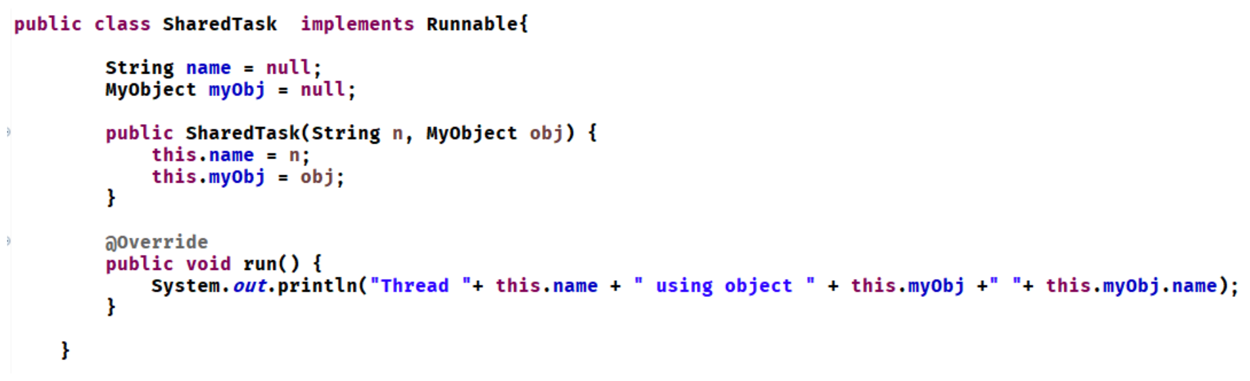
```
public class Runner {

    public static void main(String[] args) {
        Thread t1 = new Thread(new Task("one"));
        Thread t2 = new Thread(new Task("two"));
        t1.start();
        t2.start();

    }
```

```
<terminated> Runner (6) [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Mar. 2, 2022, 7:25:32 p.m. -
Thread one using object threads.MyObject@4186bf60
Thread two using object threads.MyObject@4d858e
```

5. The important thing to note is that two objects are at different memory addresses which means they are distinct objects.

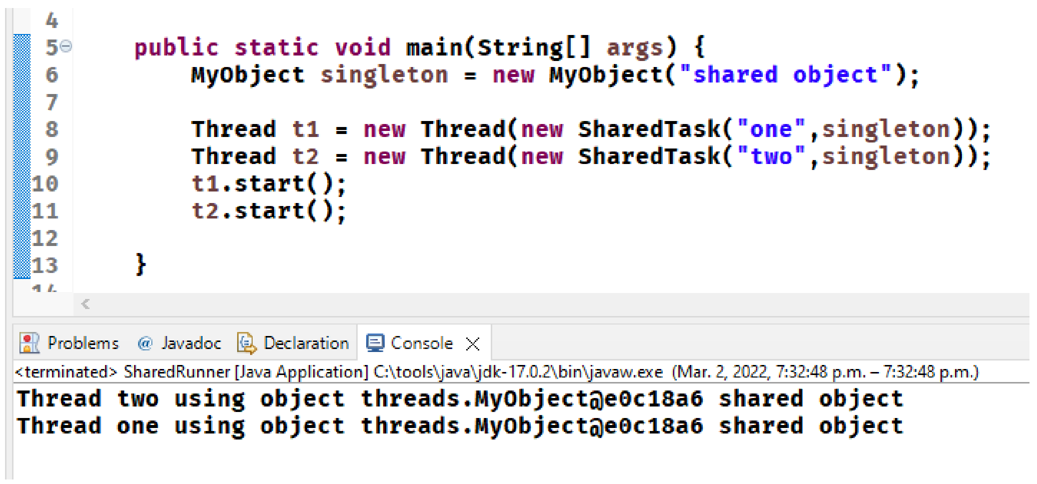
Part Two: Shared Objects

1. Recall that the main thread is also a thread so it shares heap memory with the other threads spawned from it.
2. This section uses the same definition of the MyObject class as before.
3. However, there is one change to the Task class. Instead of each Task object creating its own copy of the MyObject, the main thread will create the object and pass a reference to the MyObject to each of the Task objects.
4. This is an example of Constructor dependency injection.



```java
public class SharedTask  implements Runnable{

        String name = null;
        MyObject myObj = null;

        public SharedTask(String n, MyObject obj) {
            this.name = n;
            this.myObj = obj;
        }

        @Override
        public void run() {
            System.out.println("Thread "+ this.name + " using object " + this.myObj +" "+ this.myObj.name);
        }

    }
```

5. Now when the threads run, we can see by the addresses that they are both sharing the same MyObject instance.



```java
    public static void main(String[] args) {
        MyObject singleton = new MyObject("shared object");

        Thread t1 = new Thread(new SharedTask("one",singleton));
        Thread t2 = new Thread(new SharedTask("two",singleton));
        t1.start();
        t2.start();

    }
```

Problems Javadoc Declaration Console

```
<terminated> SharedRunner [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe  (Mar. 2, 2022, 7:32:48 p.m. – 7:32:48 p.m.)
Thread two using object threads.MyObject@e0c18a6 shared object
Thread one using object threads.MyObject@e0c18a6 shared object
```