Homework：

1、 **利用 JDK 的 java.util 包中提供的 Observable 类以及 Observer 接口实现课堂的例子（对随机数的观察输出），将程序进行你要的修改或完善。**

答：在课堂例子的原始设计中，NumberGenerator 作为一个抽象类，是被设计来被继承的，以便实现具体的数字生成逻辑。但是在使用 Java JDK 的 Observable 类后，可以直接让 RandomNumberGenerator 继承自 Observable 类，并在其中实现数字生成和观察者通知的逻辑。这种改动的主要好处是简化了代码结构，并且更加直接地利用了 Java 标准库提供的观察者模式实现，减少了自定义的部分。

```
de\homework17 ort java.util.Observable;
import java.util.Random;

2个用法
public class RandomNumberGenerator extends Observable {
    1 个用法
    private Random random = new Random();
    3 个用法
    private int number;

    0 个用法
    public int getNumber() {
        return number;
    }

    1 个用法
    public void execute() {
        for (int i = 0; i < 10; i++) {
            number = random.nextInt( bound: 50);
            setChanged();
            notifyObservers(number);
        }
    }
}
```

接着实现观察(DigitObserver 和 GraphObserver)，这些类实现了 Observer 接口，并定义了如何响应被观察对象的变化：

```
public class DigitObserver implements Observer {
    @Override
    public void update(Observable o, Object arg) {
        System.out.println("DigitObserver: " + arg);
        try {
            Thread.sleep( millis: 100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

```java
public class GraphObserver implements Observer {
    @Override
    public void update(Observable o, Object arg) {
        System.out.print("GraphObserver: ");
        int count = (Integer) arg;
        for (int i = 0; i < count; i++) {
            System.out.print("*");
        }
        System.out.println();
        try {
            Thread.sleep( millis: 100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

可以再增加一个观察者，输出随机数的平方：

```java
public class SquareObserver implements Observer {
    @Override
    public void update(Observable o, Object arg) {
        int value = (Integer) arg;
        System.out.println("SquareObserver: " + value * value);
        try {
            Thread.sleep( millis: 100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

最后在 Main 类中调用并测试：

```java
public class Main {
    0 个用法
    public static void main(String[] args) {
        RandomNumberGenerator generator = new RandomNumberGenerator();
        Observer observer1 = new DigitObserver();
        Observer observer2 = new GraphObserver();
        Observer observer3 = new SquareObserver();
        generator.addObserver(observer1);
        generator.addObserver(observer2);
        generator.addObserver(observer3);
        generator.execute();
    }
}
```

运行结果：

```
运行:  Main
F:\Java\bin\java.exe "-javaagent:F:\IntelliJ IDEA Community Edition 2022.
SquareObserver: 1369
GraphObserver: ************************************
DigitObserver: 37
SquareObserver: 64
GraphObserver: ********
DigitObserver: 8
SquareObserver: 9
GraphObserver: ***
DigitObserver: 3
SquareObserver: 529
GraphObserver: ***********************
DigitObserver: 23
SquareObserver: 1369
GraphObserver: ************************************
DigitObserver: 37
SquareObserver: 729
GraphObserver: ***************************
DigitObserver: 27
SquareObserver: 1764
```

```
SquareObserver: 1156
GraphObserver: ***********************************
DigitObserver: 34
SquareObserver: 729
GraphObserver: ***************************
DigitObserver: 27
SquareObserver: 324
GraphObserver: ******************
DigitObserver: 18

进程已结束,退出代码0
```

2、 附录

1） RandomNumberGenerator

```java
import java.util.Observable;
import java.util.Random;

public class RandomNumberGenerator extends Observable {
    private Random random = new Random();
    private int number;

    public int getNumber() {
        return number;
    }
```

```
    public void execute() {
        for (int i = 0; i < 10; i++) {
            number = random.nextInt(50);
            setChanged();
            notifyObservers(number);
        }
    }
}
```

2) GraphObserver

```
import java.util.Observer;
import java.util.Observable;

public class GraphObserver implements Observer {
    @Override
    public void update(Observable o, Object arg) {
        System.out.print("GraphObserver: ");
        int count = (Integer) arg;
        for (int i = 0; i < count; i++) {
            System.out.print("*");
        }
        System.out.println();
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

3) DigitObserver

```
import java.util.Observer;
import java.util.Observable;

public class DigitObserver implements Observer {
    @Override
    public void update(Observable o, Object arg) {
        System.out.println("DigitObserver: " + arg);
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
```

```
    }
}
```

4) SquareObserver

```
import java.util.Observer;
import java.util.Observable;

public class SquareObserver implements Observer {
    @Override
    public void update(Observable o, Object arg) {
        int value = (Integer) arg;
        System.out.println("SquareObserver: " + value * value);
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

5) Main

```
import java.util.Observer;

public class Main {
    public static void main(String[] args) {
        RandomNumberGenerator generator = new RandomNumberGenerator();
        Observer observer1 = new DigitObserver();
        Observer observer2 = new GraphObserver();
        Observer observer3 = new SquareObserver();
        generator.addObserver(observer1);
        generator.addObserver(observer2);
        generator.addObserver(observer3);
        generator.execute();
    }
}
```