Assignment 4
This is an example of the Observer pattern but using Java Sockets. Observers connect to the observable server by connecting to a socket.
It was suggested after watching the PMI Scott Bain video on the Observer Pattern
https://www.pmi.org/disciplined-agile/the-design-patterns-repository/the-observer-pattern

The interface Observable and FactoryClimateServer are provided.
There is a video explaining how this observable server works and how the observers Sensor and HVACObserver execute.
Video Link in Panopto: https://binghamton.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=704fc892-452f-45dc-bc73-b21501670bc9
You need to code the Sensor and the HVACObserver.

We assume that a warehouse has 2 Zones. There are sensors for the temperatures in Zone 1 and in Zone 2. There are also sensors for the humidity in each of the two Zones.

The server calls for the temperatures and humidities from the sensors every 5 seconds. It then shares the 4 sensor values in an array of doubles. The Observable interface gives names for indices 0, 1, 2, 3 of the array (TEMP_ZONE1 = index 0, TEMP_ZONE1 = index 1, HUMIDITY_ZONE1 = index 2, HUMIDITY_ZONE2 = index 3).

The HVACObservers can connect at any time. The assumption is that when they receive the temperature/humidity values, they turn on heating, cooling, humidifiers or dehumidifiers units to move the Zones toward the normal temperature and humidity in the Observable interface (70 degree Fahrenheit and 40% humidity).

There will one HVACObserver for each Zone.

The Sensor class

The imports are:
```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
```

The class Sensor *only needs the main method*. There will be 4 sensors but each runs in a separate process, so we do not need to make instances of the class.

Start the main method with
```
if(args.length != 4) {
   System.out.println("Usage is: "
            + "java Sensor <type> <initial value> <host name> <port number>");
   System.exit(1);
}
```

Read the 4 parameters into variables, parsing the ints using Integer,parseInt (look at the start of the main method of the server and the Echo Client and Echo Server code here:
https://docs.oracle.com/javase/tutorial/displayCode.html?code=https://docs.oracle.com/javase/tutorial/networking/sockets/examples/EchoClient.java
https://docs.oracle.com/javase/tutorial/displayCode.html?code=https://docs.oracle.com/javase/tutorial/networking/sockets/examples/EchoServer.java)

I have used the variables `int type` for the <type>, `int value` for the <initial value>, `String hostName` for the <host name>, and `int portNumber` for the <port number>.

Add this code to make sure the type is between 0 and 3. The code does not work if there are not 4 sensors with types 0, 1, 2, 3.

```
if(type < Observable.TEMP_ZONE1 || type > Observable.HUMIDITY_ZONE2) {
   System.out.println("The simulations only works with 4 different types 0 through 3");
   System.exit(1);
}
```

Now use a try-with-resources to open the 3 resources used by the Sensor.

```
try (Socket deviceSocket = new Socket(hostName, portNumber);
     DataOutputStream out = new DataOutputStream(deviceSocket.getOutputStream());
     BufferedReader in = new BufferedReader(
                                    new InputStreamReader(deviceSocket.getInputStream()))
    )
{
       // we will describe this code below
} catch (UnknownHostException e) {
       e.printStackTrace();
} catch (IOException e) {
       e.printStackTrace();
}
```

The benefit of try-with-resources is that Java guarantees to close them properly when the block exits through a correct or error termination.

The code inside the try-catch block

The EchoServer code example shows how to read lines coming in from the socket's input stream in a while loop.
At the start of the while loop, please write
`System.out.print("Sensor " + type + " received " + fromServer);`
Where `fromServer` is the string coming from the socket.

The server will send one of 3 messages: "ID?", "VAL?", or "SETNORM"
If `fromServer` is "ID?", you send `type` through the DataOutputStream: `out.writeInt(type);`
If `fromServer` is "VAL?", you first randomly add 1 to value or subtract 1 from value. `if(Math.random() <= 0.5)` will give a random choice. Then you send `value` through the DataOutputStream: `out.writeDouble(value);`
If `fromServer` is "SETNORM", you change `value` to Observable.NORMAL_TEMP if `type` indicates a temperature sensor (Observable.TEMP_ZONE1 or Observable.TEMP_ZONE2) and otherwise you change `value` to Observable.NORMAL_HUMIDITY. Then output `value` on the `out` stream as above.

After the if statements, print the current value held by the Sensor
System.out.println(". Value = " + value);

The HVACObserver class

Again all the code is inside the main method.

The imports are

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.Socket;
```

```
import java.net.UnknownHostException;
import java.util.Arrays;
```

The main method starts with

```
if(args.length != 3) {
   System.out.println("Usage is: "
            + "java HVACObserver <observer id> <host name> <observer port number");
   System.exit(1);
}

int observerId = // parse the first input
String hostName = // copy second input
int observerPortNumber = // parse the third input
```

The simulation depends on correct values of `observerId`:

```
if(observerId < Observable.TEMP_ZONE1 || observerId > Observable.TEMP_ZONE2) {
    System.out.println("The simulations only works with 2 different types 0 through 1");
    System.exit(1);
}
```

Start a try-with-resources with the Socket:

```
Socket deviceSocket = new Socket(hostName, observerPortNumber)
```

And an object input stream for serialized data:

```
ObjectInputStream in = new ObjectInputStream(deviceSocket.getInputStream())
```

The catches are for the exceptions UnknownHostException e, IOException e, and ClassNotFoundException e.

The while loop is infinite: `while(true)`

In the loop the first line is to un-serialize the array that is sent from the server:

```
double[] values = (double[])in.readObject();
```

Print the id and the values

```
System.out.println("Observer " + observerId + " " + Arrays.toString(values));
```

Next depending on whether the observerId is 0 (Zone 1) or 1 (Zone 2), and then depending on whether the temperature and/or humidity are above or below Observable.NORMAL_TEMP and Observable.NORMAL_HUMIDITY, write the appropriate message to simulate moving the actual values toward the NORMAL ones:

```
System.out.println("Heater on in Zone 1");
System.out.println("A/C on in Zone 1");
System.out.println("Humidifier on in Zone 1");
System.out.println("Dehumidifier on in Zone 1");
System.out.println("Heater on in Zone 2");
System.out.println("A/C on in Zone 2");
System.out.println("Humidifier on in Zone 2");
System.out.println("Dehumidifier on in Zone 2")
```

These outputs will also be explained in the video.

RUNNING THE CODE.

Put the 4 class files for the Server in a subdirectory server,
FactoryClimateServer$1.class
FactoryClimateServer$SensorConnectThread.class
FactoryClimateServer.class
Observable.class
Put the 3 class files for the Sensors and HVAC systems in a subdirectory devices:
HVACObserver.class

Observable.class
Sensor.class

Then run the program as shown in the video.