University of Westminster
Faculty of Science and Technology

Module code and title: 5COSC004W-Client Service Architecture Tutorial Manual	
Tutorial title	**Vectors and iteration loops**
Tutorial type	**Guided and indepenent and non-marked**
Week 01	**06/02/2020**

Contents

Learning Goals

This tutorial focuses on two main learning goals:

- To refresh the use of ArrayLists and iteration loops.
- To stress the correct use of types

It is divided into two separate sections, the student will perform the first task (1-13) following the instructions of the tutor, and then, will complete the other tasks independently.

TASKS to be Performed under the instruction of the Tutor (from Task 1 to Task 13)

1) Start Netbeans in your system. If Netbeans is not present in your system, use AppsAnywhere to launch it: (https://www.westminster.ac.uk/sites/default/public-files/general-documents/Using%20AppsAnywhere.pdf)

2) Create a new java project in Netbeans (Figure 1 to 3). We will call this project SimpleTestThree and it will consist of a simple temperature-recording system based on a client/server dummy architecture. Modify the name of the standard main class to SimpleClient

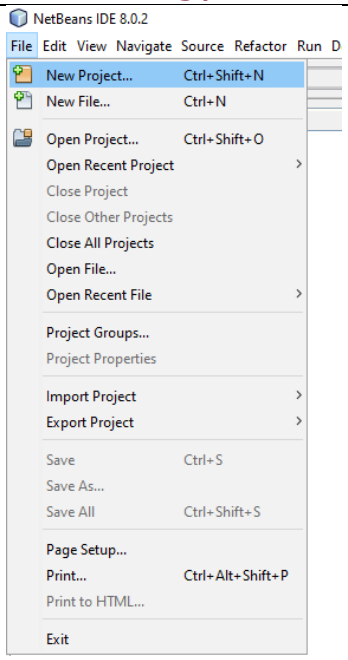

Figure 1, Create a New Project in NetBeans (1)

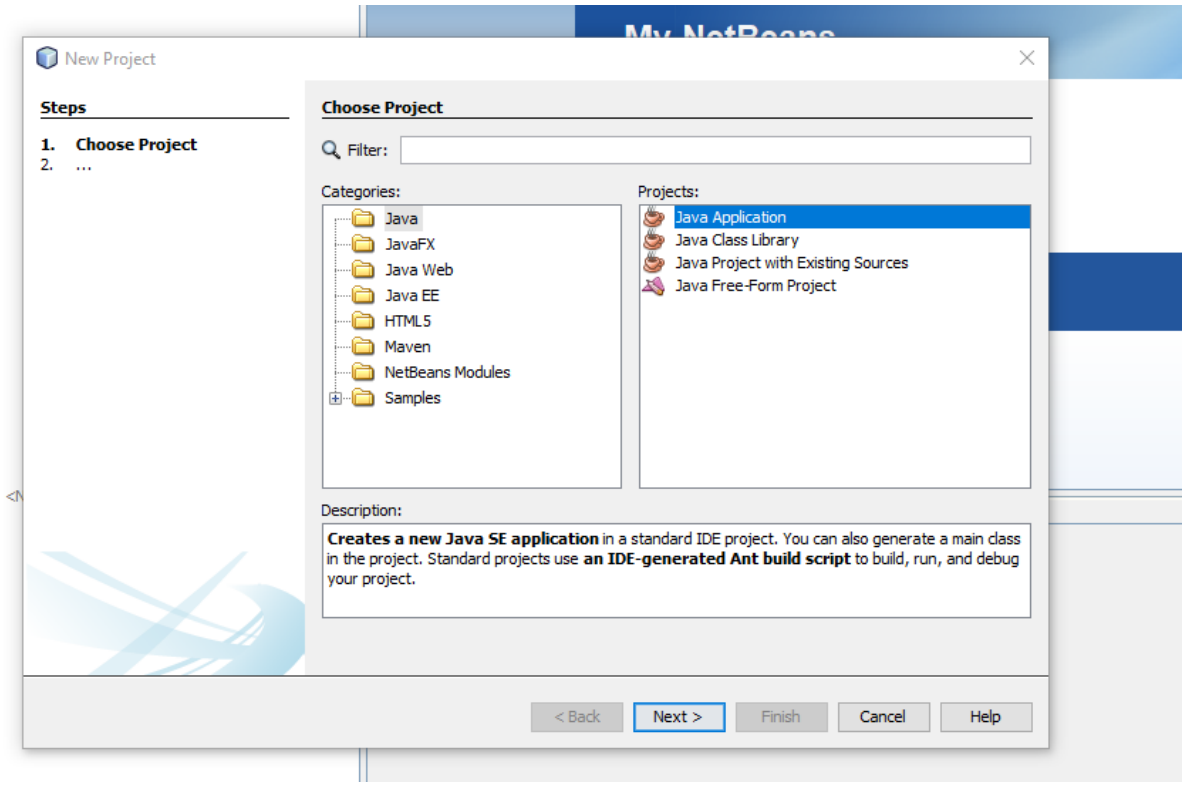

Figure 2, Create a New Project in NetBeans (2)

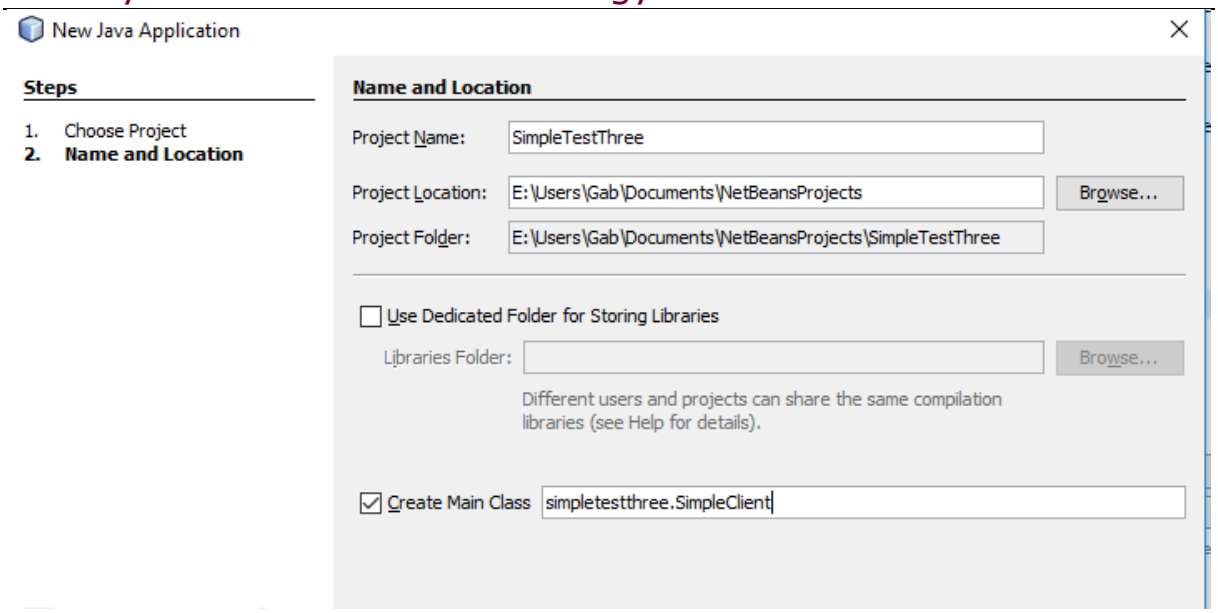

Figure 3, Create a New Project in NetBeans (3)

3) Create a new java class in the project which will represent our SimpleTemperatureRecorderServer (Figure 4)

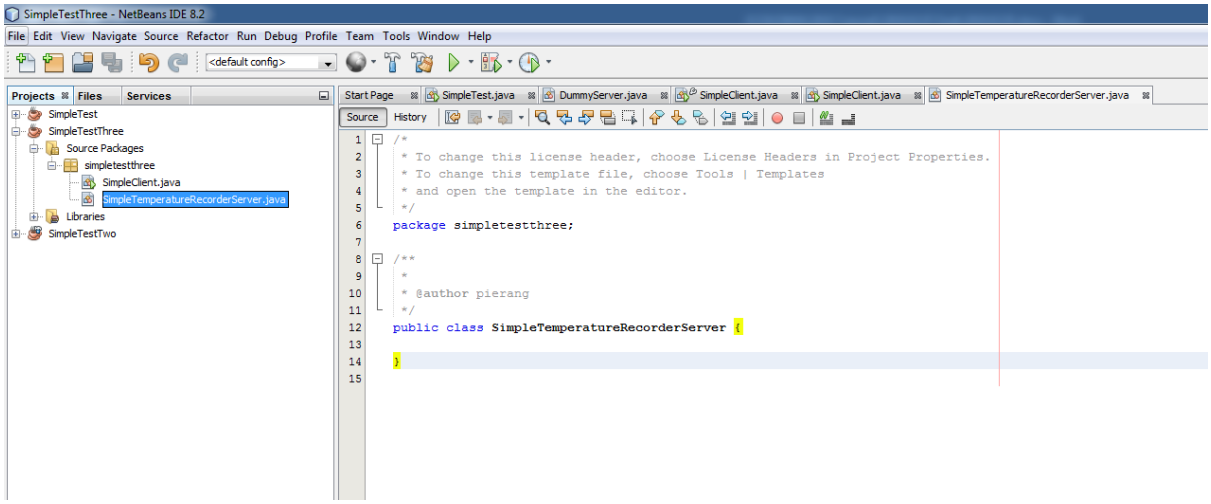

Figure 4, Create a new Java Class in NetBeans (1)

4) As always, we add a method to the server that returns a simple true which we can test to see if the server is connected (Figure 5)

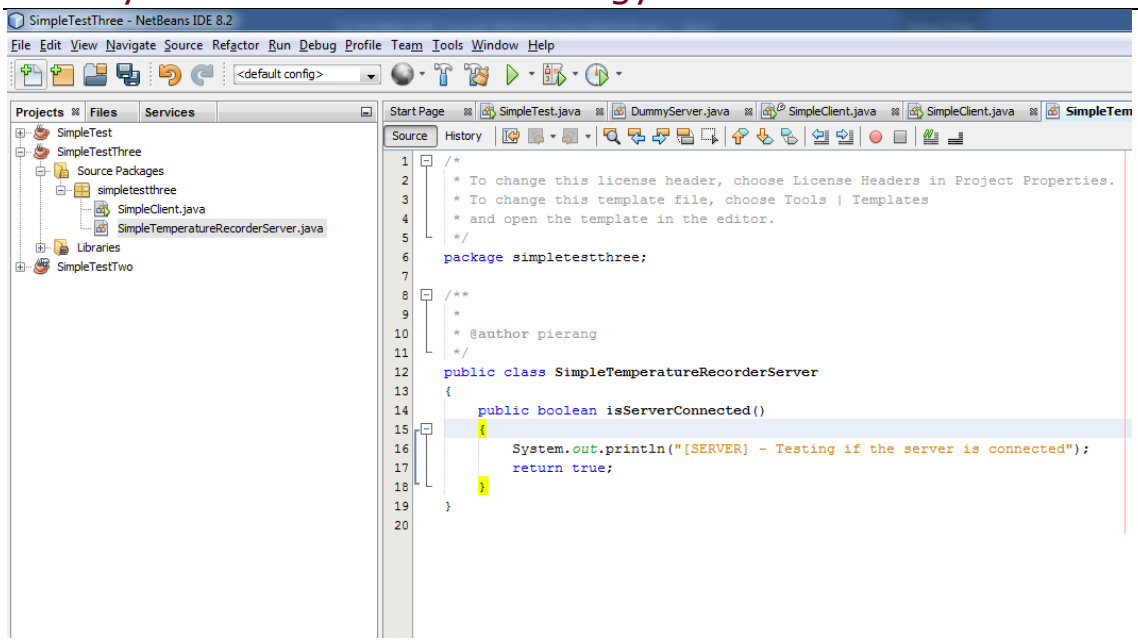

Figure 5, Add simple method to the DummyServer Class

1) Try to modify the primitive data type boolean (which is not a class)[1] with the equivalent type Boolean[2] (which is a class). Note that this modification does not require you any further change in the code as the Java interpreter performs all the conversions automatically if their meaning is unique. See the example with int and float later to better understand the concept of uniqueness in type conversion).

```
public Boolean isServerConnected()
{
    System.out.println("[SERVER] - Testing if the server is connected");
    return true;
}
```

Figure 6, modify boolean to Boolean

2) Create an instance of the SimpleClient class and an empty method execute and add the invocation of the server in the execute method, add the invocation of the connection test (Figure 6).

[1] https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
[2] https://docs.oracle.com/javase/7/docs/api/java/lang/Boolean.html

```
12  public class SimpleClient
13  {
14
15      /**
16       * @param args the command line arguments
17       */
18      public static void main(String[] args)
19      {
20          SimpleClient    client  =   new SimpleClient();
21          client.executeTest();
22
23      }
24
25      private void executeTest()
26      {
27          SimpleTemperatureRecorderServer server  =   new SimpleTemperatureRecorderServer();
28          if(server.isServerConnected())
29          {
30              System.out.println("[CLIENT] - The Server is connected. The test can proceed");
31          }
32          else
33          {
34              System.out.println("[CLIENT] - The Server is NOT connected, terminating test");
35          }
36
37
38
39      }
40
41  }
42
```

Figure 7, Create instance and execute method.

3) Run the client to test if everything is ok.

4) Now we need to add to the server the possibility to record the temperatures. In order to do that, we will add to the server an ArrayList with all the temperatures and a method that adds one temperature sample to the server. We will use Double and not double as the use of proper types (Double) instead of primitive data types (double) is preferable.
   NetBeans will show you a syntax error as you have not imported the definition of ArrayList, add it (You can do it by right-clicking on the lightbulb with the red dot and following NetBeans suggestions) – Figure 8 and Figure 9. Note the System.out.println in the method that prints all the recorded temperatures (the second one) implicitly transforms an ArrayList into a String This is executed because ArrayList has a toString method that invokes automatically the toString method of all his members and concatenates all this values separated by a comma, such String is then passed as an argument to the System.out.println invocation.

```
6   package simpletestthree;
7
8   /**
9    *
10   * @author pierang
11   */
12  public class SimpleTemperatureRecorderServer
13  {
        ArrayList<Double>  temperatures    =   new ArrayList<Double>();
15
16      public Boolean isServerConnected()
17      {
18          System.out.println("[SERVER] - Testing if the server is connected");
19          return true;
20      }
21
        public  void    addTemperature(Double sample)
23      {
24          System.out.println("[SERVER] - Adding sample: " + sample);
25          temperatures.add(sample);
26          System.out.println("[SERVER] - All samples now are: " + temperatures);
27      }
```

Figure 8, Add temperatures ArrayList

```
package simpletestthree;

import java.util.ArrayList;

/**
 *
 * @author pierang
 */
public class SimpleTemperatureRecorderServer
{
    ArrayList<Double>  temperatures    =   new ArrayList<Double>();

    public Boolean isServerConnected()
    {
        System.out.println("[SERVER] - Testing if the server is connected");
        return true;
    }

    public  void    addTemperature(Double sample)
    {
        System.out.println("[SERVER] - Adding sample: " + sample);
        temperatures.add(sample);
        System.out.println("[SERVER] - All samples now are: " + temperatures);
    }
```

Figure 9, Add definition of ArrayList

5) Let's write the code on the client (Figure 10) to invoke the addTemperature method on the server.

```
private void executeTest()
{
    SimpleTemperatureRecorderServer server  =   new SimpleTemperatureRecorderServer();
    if(server.isServerConnected())
    {
        System.out.println("[CLIENT] - The Server is connected. The test can proceed");
        server.addTemperature(10);
    }
    else
    {
        System.out.println("[CLIENT] - The Server is NOT connected, terminating test");
    }
}
```

Figure 10, Testing the addTemperature method(1)

Note the interesting fact that this time (as opposite to the Boolean example in step 8) the Java interpreter refuses to automatically convert 10 to a Double, Why is that ? If it automatically converted true to a Boolean, why does it refuse to convert 10 to a Double ? The answer is that 10 can be converted both to an Integer and to a Double, therefore the type conversion is not univocally determined and the interpreter refuses to select one solution on behalf of the user (as it is proper) thus raising a syntax error instead. There are two ways to solve this. One is to do the conversion manually as in Figure 11 or to make the conversion univocally clear by modifying the syntax of the invocation as in Figure 12. Note that 10.0 is still semantically identical to 10 but it is not equivalent syntactically and the Java Interpreter can assume that we want to use numbers that have a fractional part. We will adopt the second solution as it is faster to implement.

```
private void executeTest()
{
    SimpleTemperatureRecorderServer server  =   new SimpleTemperatureRecorderServer();
    if(server.isServerConnected())
    {
        System.out.println("[CLIENT] - The Server is connected. The test can proceed");
        server.addTemperature(new Double((10)));
    }
    else
    {
        System.out.println("[CLIENT] - The Server is NOT connected, terminating test");
    }
}
```

Figure 11, Manual Conversion

```
private void executeTest()
{
    SimpleTemperatureRecorderServer server  =   new SimpleTemperatureRecorderServer();
    if(server.isServerConnected())
    {
        System.out.println("[CLIENT] - The Server is connected. The test can proceed");
        server.addTemperature(10.0);
    }
    else
    {
        System.out.println("[CLIENT] - The Server is NOT connected, terminating test");
    }
}
```

Figure 12, Automatic Conversion

6) Let's execute the code on the client. You can observe the results on Figure 13, Executing the client. Everything is ok !

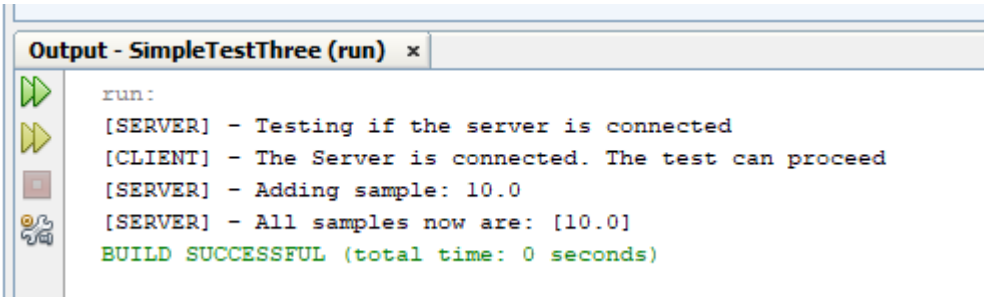

Figure 13, Executing the client

7) Now let's add some more temperature samples to the server (Figure 14) and let's execute the Client (Figure 15).

```
private void executeTest()
{
    SimpleTemperatureRecorderServer server  =   new SimpleTemperatureRecorderServer();
    if(server.isServerConnected())
    {
        System.out.println("[CLIENT] - The Server is connected. The test can proceed");
        server.addTemperature(10.0);
        server.addTemperature(1.0);
        server.addTemperature(12.0);
        server.addTemperature(-0.1);
        server.addTemperature(-18.0);
        server.addTemperature(18.0);
    }
    else
    {
        System.out.println("[CLIENT] - The Server is NOT connected, terminating test");
    }
}
```

Figure 14, Adding more samples to the Server

```
run:
[SERVER] - Testing if the server is connected
[CLIENT] - The Server is connected. The test can proceed
[SERVER] - Adding sample: 10.0
[SERVER] - All samples now are: [10.0]
[SERVER] - Adding sample: 1.0
[SERVER] - All samples now are: [10.0, 1.0]
[SERVER] - Adding sample: 12.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0]
[SERVER] - Adding sample: -0.1
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1]
[SERVER] - Adding sample: -18.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1, -18.0]
[SERVER] - Adding sample: 18.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1, -18.0, 18.0]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 15, Executing Client Test

8) Let's now add a method that finds the coldest sample. On the server we have to check all the values in the ArrayList and find the lowest. We can use a for loop to do this (Figure 16).

```
public  Double    getColdestTemperature()
{
    Double min = 0.0;

    for(int index = 0; index < temperatures.size(); index++)
        if(min > temperatures.get(index))
            min = temperatures.get(index);

    return min;
}
```

Figure 16, First attempt at finding the coldest temperature.

9) Let's invoke this method from the client and let's add some printout code Figure 17) to see if the results are correct.

```
private void executeTest()
{
    SimpleTemperatureRecorderServer server  =   new SimpleTemperatureRecorderServer();
    if(server.isServerConnected())
    {
        System.out.println("[CLIENT] - The Server is connected. The test can proceed");
        server.addTemperature(10.0);
        System.out.println("[CLIENT] - The minimum recorded temperature is " + server.getColdestTemperature());
        server.addTemperature(1.0);
        System.out.println("[CLIENT] - The minimum recorded temperature is " + server.getColdestTemperature());
        server.addTemperature(12.0);
        System.out.println("[CLIENT] - The minimum recorded temperature is " + server.getColdestTemperature());
        server.addTemperature(-0.1);
        System.out.println("[CLIENT] - The minimum recorded temperature is " + server.getColdestTemperature());
        server.addTemperature(-18.0);
        System.out.println("[CLIENT] - The minimum recorded temperature is " + server.getColdestTemperature());
        server.addTemperature(18.0);
        System.out.println("[CLIENT] - The minimum recorded temperature is " + server.getColdestTemperature());
     }
    else
    {
        System.out.println("[CLIENT] - The Server is NOT connected, terminating test");
    }
}
```

Figure 17, Invoking the getColdestTemperature from the Client

10) Let's test this running the client (Figure 18). You can note that the method IS NOT correct. The first 3 coldest temperatures are wrong as they are 0, they should be 10, 1, 1, -0.1, etc….. Why is this ? Observe that after the first temperature below zero the results are correct. Try to solve the problem on your own (by using the debugger, if needed) before looking at the solution in the following tasks.

```
run:
[SERVER] - Testing if the server is connected
[CLIENT] - The Server is connected. The test can proceed
[SERVER] - Adding sample: 10.0
[SERVER] - All samples now are: [10.0]
[CLIENT] - The minimum recorded temperature is 0.0
[SERVER] - Adding sample: 1.0
[SERVER] - All samples now are: [10.0, 1.0]
[CLIENT] - The minimum recorded temperature is 0.0
[SERVER] - Adding sample: 12.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0]
[CLIENT] - The minimum recorded temperature is 0.0
[SERVER] - Adding sample: -0.1
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1]
[CLIENT] - The minimum recorded temperature is -0.1
[SERVER] - Adding sample: -18.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1, -18.0]
[CLIENT] - The minimum recorded temperature is -18.0
[SERVER] - Adding sample: 18.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1, -18.0, 18.0]
[CLIENT] - The minimum recorded temperature is -18.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 18, Testing the getColdestTemperature from the Client

11) The reason for the erroneous behaviour of the method is the initialization of the variable min. By setting its initial value to zero (which may seem a reasonable choice) we surreptitiously introduced a new first sample which value is zero and this alters the behaviour of the code which will return false results until the first temperature below zero is recorded. This kind of bugs are particularly hard to find as they depend on the data set. If you have only temperatures below zero it will always behave correctly, on the other end, if you have only temperatures above zero it will always behave incorrectly.

12) If I modify the method getColdestTemperature like in Figure 19, it now behaves correctly. Why is that ? Observe that this time we initialize the min to the first recorded temperature and not an arbitrary value as before, also observe how the lower index of the loop has been modified from 0 to 1 as the first sample has been already been used to initialize min.

```
public  Double    getColdestTemperature()
{
    Double min = temperatures.get(0);

    for(int index = 1; index < temperatures.size(); index++)
        if(min > temperatures.get(index))
            min = temperatures.get(index);

    return min;
}
```

Figure 19, Second attempt (correct) at finding the coldest temperature.

13) Let's test it again by running the client as in Figure 20. You can observe that the minimum temperatures are now correct.

```
[SERVER] - Testing if the server is connected
[CLIENT] - The Server is connected. The test can proceed
[SERVER] - Adding sample: 10.0
[SERVER] - All samples now are: [10.0]
[CLIENT] - The minimum recorded temperature is 10.0
[SERVER] - Adding sample: 1.0
[SERVER] - All samples now are: [10.0, 1.0]
[CLIENT] - The minimum recorded temperature is 1.0
[SERVER] - Adding sample: 12.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0]
[CLIENT] - The minimum recorded temperature is 1.0
[SERVER] - Adding sample: -0.1
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1]
[CLIENT] - The minimum recorded temperature is -0.1
[SERVER] - Adding sample: -18.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1, -18.0]
[CLIENT] - The minimum recorded temperature is -18.0
[SERVER] - Adding sample: 18.0
[SERVER] - All samples now are: [10.0, 1.0, 12.0, -0.1, -18.0, 18.0]
[CLIENT] - The minimum recorded temperature is -18.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 20, Testing the getColdestTemperature (correct implementation) from the Client

TASKS to BE PERFORMED Independently be the student (from Task 14 to Task 20) (Formative Assessment)

14) Add a getHottestTemperature that returns the highest temperature and test it.
15) Add a getAverageTemperature that returns the average temperature and test it.
16) Add a getNumberOfSamples that returns the number of temperatures being recorded so far.
17) What if the client invoke the getColdestTemperature (or Get HottestTemperature or getAverageTemperature) before any temperature has been added with addTemperature (in this case the ArrayList is empty). Add an exception on the server side to cope with this situation in all the relevant methods and handle it correctly on the client side.
18) Everytime we execute the program, we loose the temperatures which we have added during the previous execution. How can we prevent this from happening ? You do not have to develop code at the moment to solve this problem, just reflect on it and think about possible solutions.
19) There are various scales for ttemperatures (Celsius, Fahrenheit, Kelvin), modify the code so that it is possible to perform all actions specifying the scale as a parameter to the methods and convert to Celsius in the server if it comes in a different scale
20) Are all temperatures acceptable ? Is there a lower or upper limit to a temperature ? Is a temperature of plus one million degrees acceptable ? Is a temperature of minus one million degrees acceptable? If you think that there are boundaries of acceptable temperatures, how would you deal with it? Would raising an exception in the addTemperature a correct approach ?