The Assignment-2 Specification and Marking Criteria

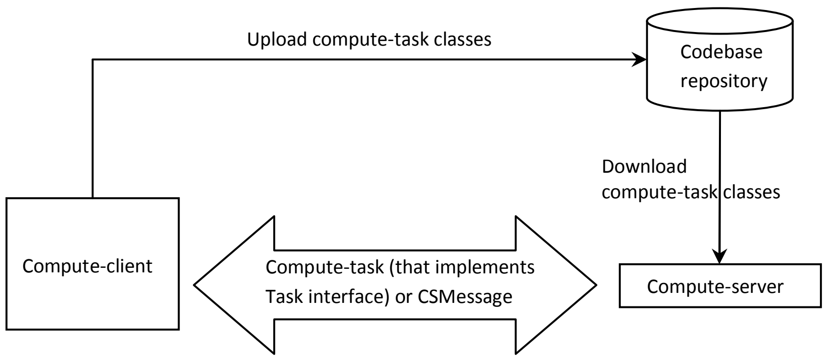
Java RMI (Remote Method Invocation, reference Chapter 5 of the textbook and Week-3 lecture) enables the local invocation and remote invocation use the same syntax to implement generic remote servers like the Compute Engine example in Week-3 lecture slides. However, Java RMI needs 2 HTTP servers to transfer Java classes between a RMI client and a RMI server at runtime. In addition, Java RMI applications need a RMI Registry for registering or looking up the remote objects.

In this assignment (assignment-2), you are to implement a remote invocation framework that is similar to Java RMI but lightweight (note: for this assignment, you don't use any Java RMI APIs). To implement the framework, you will need to use Java Interface, Java Objection Serialization, Java Thread and client/server model. Recall you have created some Java examples about these Java components and models for assignment-1; now you will need to use these Java components as a reference to build this framework. Thus if necessary, you may need to review your assignment-1.

This assignment specification is as follows.

Part 1: Java TCP Networking, Multi-threading and Object Serialization Programming

The framework consists of a compute-server, a compute-client and a codebase repository, which are depicted in the following diagram. The framework is a generic computing architecture because the compute-client and compute-server just need to know the `Task` and `CSMessage` components in advance before the framework can start to run. The specification of the components is as follows.


Upload compute-task classes
Codebase repository
Download compute-task classes
Compute-client
Compute-task (that implements Task interface) or CSMessage
Compute-server

1. The interaction contract

The interaction contract between the client and server is defined by the `Task` interface:

```
//The Task interface (interaction contract) between clients and the server
public interface Task {
    public void executeTask();
    public Object getResult();}
```

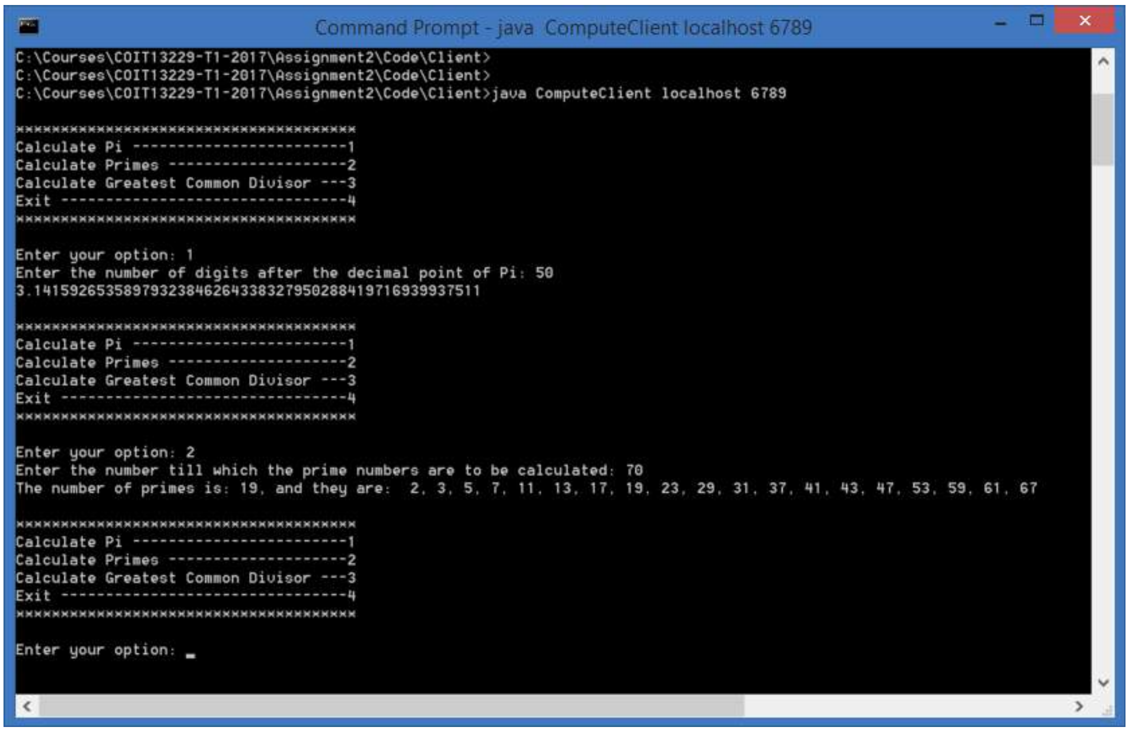
Every compute-task must implement the `Task` interface. Executing the `executeTask()` method will perform the task and set the result. Calling the `getResult()` method will return the result.

2. The compute-client and compute-server

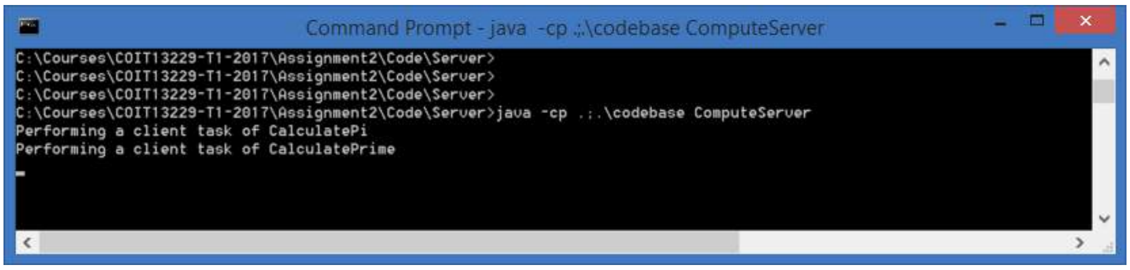
The compute-server is used as a generic compute-engine. While running, the server is continuously waiting for the compute-tasks. A compute-task is created by a compute-client and sent as a serialized object to the compute-server. Once the compute-server receives a task, it will cast (deserialize) it into the `Task` interface type and call its `executeTask()` method. After executing the task, the compute-server will send the same object back to the compute-client.

The compute-client is continuously accepting a user's requests. Every request specifies a compute-problem and its corresponding parameters. For a request, the compute-client creates a compute-task and sends it as a serialized object to the compute-server. Once receiving the compute-task object back from the server, the compute-client will call the `getResult()` method of the object to display the result.

The following screenshots show the interaction between a compute-client and the compute-server.



```
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Client>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Client>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Client>java ComputeClient localhost 6789

**************************************
Calculate Pi ------------------------1
Calculate Primes --------------------2
Calculate Greatest Common Divisor ---3
Exit --------------------------------4
**************************************

Enter your option: 1
Enter the number of digits after the decimal point of Pi: 50
3.14159265358979323846264338327950288419716939937511

**************************************
Calculate Pi ------------------------1
Calculate Primes --------------------2
Calculate Greatest Common Divisor ---3
Exit --------------------------------4
**************************************

Enter your option: 2
Enter the number till which the prime numbers are to be calculated: 70
The number of primes is: 19, and they are:  2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67

**************************************
Calculate Pi ------------------------1
Calculate Primes --------------------2
Calculate Greatest Common Divisor ---3
Exit --------------------------------4
**************************************

Enter your option: _
```

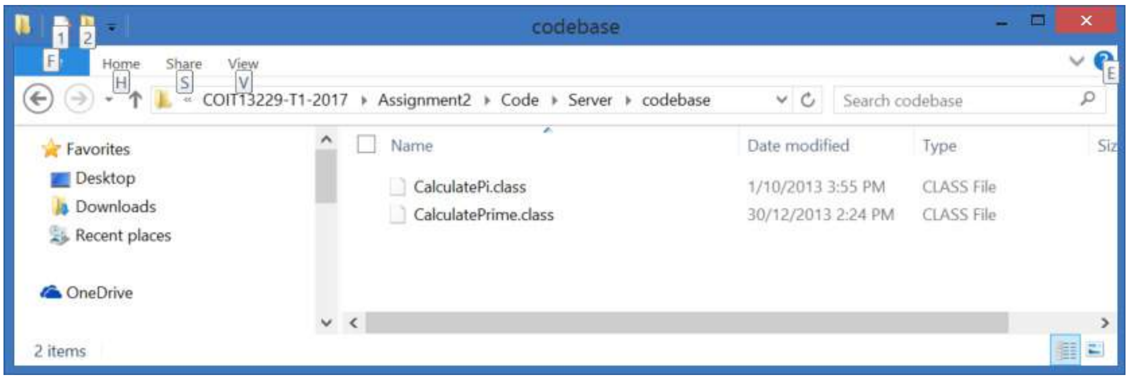


```
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>java -cp .;.\codebase ComputeServer
Performing a client task of CalculatePi
Performing a client task of CalculatePrime
```

3. The codebase repository

Such a framework makes the compute-server generic. That is, the compute-server just needs to know the `Task` interface, then it can be compiled and run. If a compute-client implements a new compute-task after the server is run up, the compute-client just needs in some way (in real world application it could be a FTP server, but in this assignment, you just need to copy the files into a directory) to upload the Java class of the compute-task into a pre-determined network location (e.g. the codebase directory), which the compute-server can access from its Java `classpath`. Then the compute-server can perform such a new compute-task. Therefore, the server never needs to be shut down, recompiled, and restarted.

The following screenshot shows that there are 2 compute-tasks that have been uploaded into the `codebase` repository.



Name	Date modified	Type
CalculatePi.class	1/10/2013 3:55 PM	CLASS File
CalculatePrime.class	30/12/2013 2:24 PM	CLASS File

4. The error message

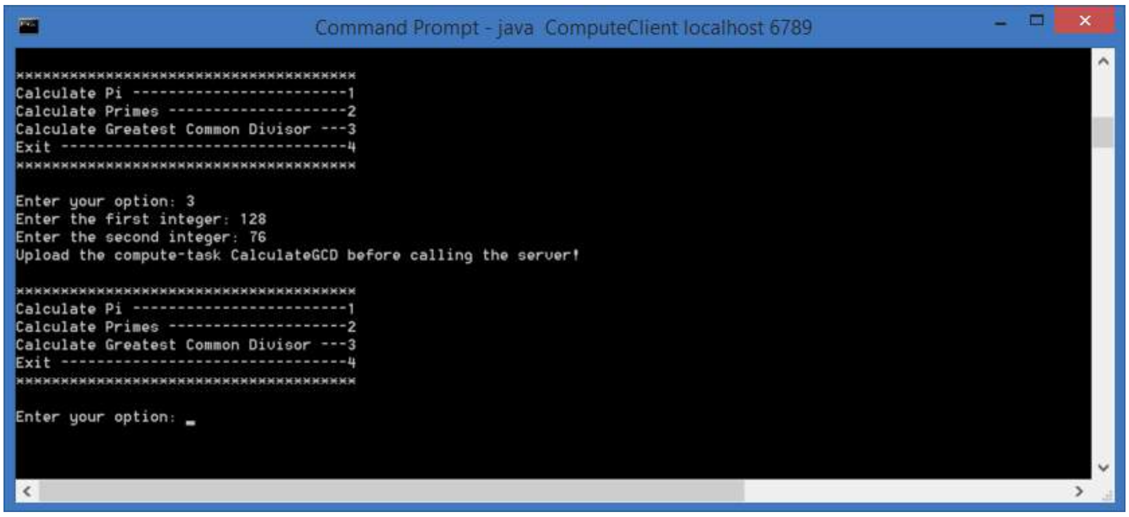
However, when there is an exception occurred (e.g. a compute-client wants the compute-server to perform a compute-task, but forgets uploading the Java class of the compute-task) onto the codebase of compute-server, the compute-server will create a `CSMessage` object and sends it back to the compute-client. Note: the `CSMessage` follows the interaction contract by implementing the `Task` interface. By calling the `getResult()` method, the compute-client will know the problem and fix it later on.

```
import java.io.*;
public class CSMessage implements Task, Serializable {
       //The variable that holds the error information
       private String finalResult;
```

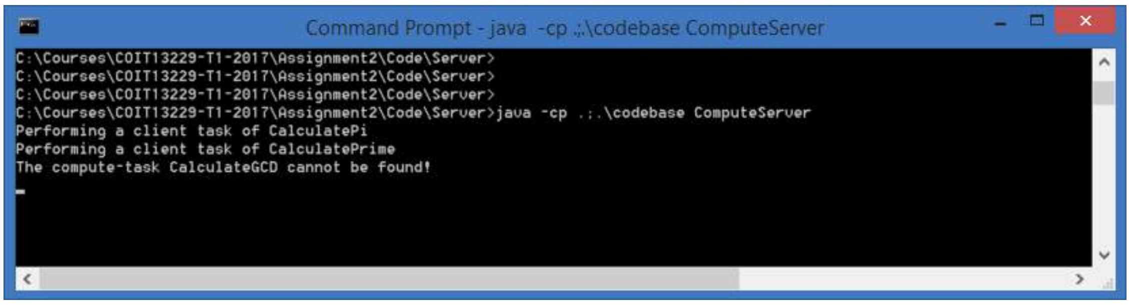
```
        public CSMessage() {
        }
        //Return the final computing result
        public Object getResult() {
                return finalResult;
        }
        //Set the error message
        public void setMessage(String msg) {
                finalResult=msg;
        }
        public void executeTask() {
    }
}
```

The following screenshots show the situation of calling the compute-server before and after the compute-task `ComputeGCD` is uploaded.

- Before the compute-task `ComputeGCD` is uploaded:

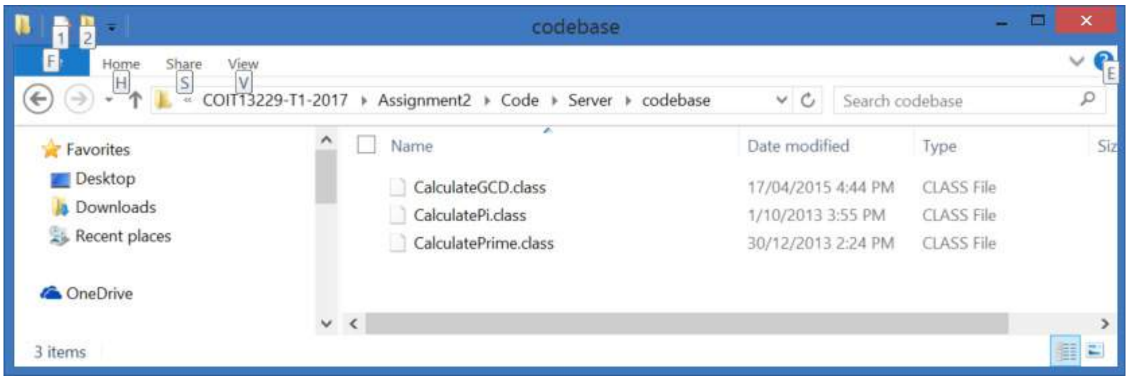


```
************************************
Calculate Pi -----------------------1
Calculate Primes -------------------2
Calculate Greatest Common Divisor ---3
Exit -------------------------------4
************************************

Enter your option: 3
Enter the first integer: 128
Enter the second integer: 76
Upload the compute-task CalculateGCD before calling the server!

************************************
Calculate Pi -----------------------1
Calculate Primes -------------------2
Calculate Greatest Common Divisor ---3
Exit -------------------------------4
************************************

Enter your option: _
```

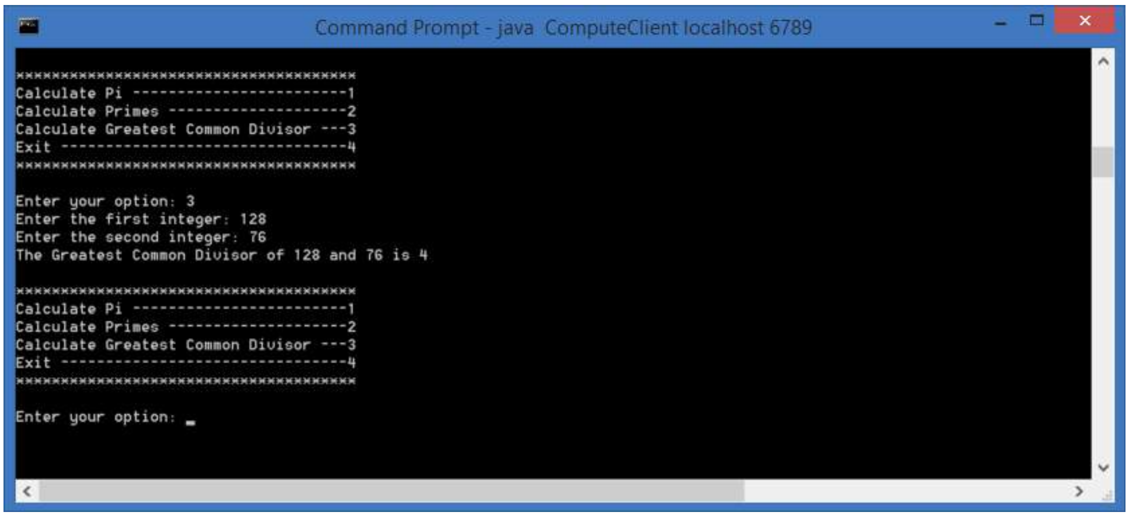


```
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>java -cp .;.\codebase ComputeServer
Performing a client task of CalculatePi
Performing a client task of CalculatePrime
The compute-task CalculateGCD cannot be found!
_
```

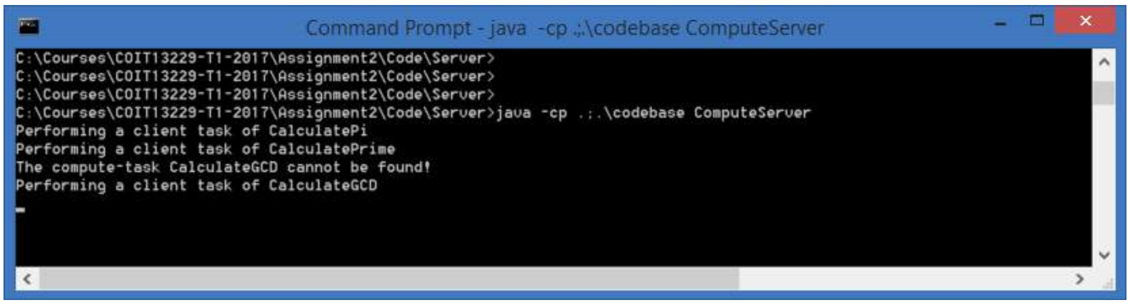
- After the compute-task `ComputeGCD` is uploaded:



Name	Date modified	Type
CalculateGCD.class	17/04/2015 4:44 PM	CLASS File
CalculatePi.class	1/10/2013 3:55 PM	CLASS File
CalculatePrime.class	30/12/2013 2:24 PM	CLASS File



```
************************************
Calculate Pi -----------------------1
Calculate Primes -------------------2
Calculate Greatest Common Divisor ---3
Exit --------------------------------4
************************************

Enter your option: 3
Enter the first integer: 128
Enter the second integer: 76
The Greatest Common Divisor of 128 and 76 is 4

************************************
Calculate Pi -----------------------1
Calculate Primes -------------------2
Calculate Greatest Common Divisor ---3
Exit --------------------------------4
************************************

Enter your option: _
```



```
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>
C:\Courses\COIT13229-T1-2017\Assignment2\Code\Server>java -cp .;.\codebase ComputeServer
Performing a client task of CalculatePi
Performing a client task of CalculatePrime
The compute-task CalculateGCD cannot be found!
Performing a client task of CalculateGCD
-
```

To complete this assignment, you need to implement such a framework and integrate the Calculate Pi, Calculate Primes and Calculate the Greatest Common Divisor tasks into this framework. The algorithms of these tasks are given on the unit web site. Your compute-server must be multi-threaded and follow the 'thread-per-connection' architecture (reference Week-4 contents). The communication between the compute-server and the compute-client must use TCP protocol through the Java TCP API `Socket` and `ServerSocket` as described in Week-2 contents of this unit and also online at, http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html, and http://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html). Please note: **use of any other protocols will incur no marks to be awarded for this part**.

To implement the framework, you need to implement the following Java classes:

1. A Java application to implement the compute-client;
2. A Java application to implement the compute-server; and
3. A Java class to implement the request processing thread.
4. A number of Java classes to implement Calculate Pi, Calculate Primes and Calculate the Greatest Common Divisor tasks.

Note: to simulate compute-client and compute-server interaction, you don't have to run them on two physical machines. Instead, they can be run on two JVMs (Java Virtual Machines) on a single physical machine. As a result, the name of the server machine can be 'localhost'.

Part 2: Program use and test instruction

After the implementation of the framework, prepare an end user instruction about how to use your software. The instruction should cover all aspects of the framework as detailed in the marking criteria below.

Submission

You need to provide the following files in your submission.

1. Files of Java source code of the compute-client, the compute-server and the processing thread and the compute-tasks. The in-line comments on the data structure and program structure in the programs are required. These source code files must be able to be compiled by the standard JDK (Java Development Kit) or NetBeans from Oracle (http://www.oracle.com/technetwork/java/index.html).
2. The compiled Java class files of the source code. These Java classes must be runnable on the standard Java Runtime Environment (JRE) or NetBeans from Oracle (http://www.oracle.com/technetwork/java/index.html).
3. A Microsoft Word document to address the issues as specified in the Part 2 above.

All the required files must be compressed into a zip file for submission. You must submit your assignment via the unit web site. **Any hardcopy or email submission will not be accepted. After the marked assignments are returned, any late submissions will not be accepted.**

The Marking Criteria

Marking Criteria	**Available Marks**
Part 1: The implementation of the framework	12
1. Whether the program can be compiled by JDK and executed by JRE or NetBeans	1
2. Whether the given Task interface is properly used as the unique communication contract between the client and server	2
3. Whether the 3 compute-tasks have been implemented as Java serializable objects, can be successfully transferred between the client and server and the server can return correct results to the client	3
4. Whether 'the task has not been uploaded' exception can be handled by using the `CSMessage`	2
5. Whether the client program functions correctly	2
6. Whether the sever is multi-threaded by using the 'thread-per-connection' model	2
Part 2: Program use and test instruction	8
1. Whether the program installation is clearly described	1
2. Whether the codebase is clearly described	1
3. Whether the test instruction covers all 3 compute-tasks	3
4. Whether the test instruction covers 'the task has not been uploaded' exception handling.	1
5. Whether the necessary screenshots have been provided and helpful for the test instruction.	2
Sub Total for Assignment-2	20

Late Penalty	-1.0 (5%) each calendar day(either full or part)
Plagiarism Penalty	
Total for Assignment-2	