The Assignment 1 Specification and Marking Criteria

A Simplified Master/Worker Framework

The application background

Volunteer computing is to harness the donated computing cycles and temporary storage from millions of the Internet volunteers for a computation task, which is too large to be processed by a single computer in a reasonable amount of time. The following web page outlines volunteer computing.

https://boinc.berkeley.edu/trac/wiki/VolunteerComputing, accessed on 15th Jan 2020

The most successful volunteer computing framework is BOINC (Berkeley Open Infrastructure for Network Computing). BOINC is client/server architecture. The server (termed as master) creates a list of compute-tasks. Volunteer clients (termed as worker in BOINC) access the master to download available compute-tasks. Workers perform the compute-tasks and upload the results to the master. Optionally workers may be awarded credits by the master for their contribution of computing cycles and temporary storage. Volunteer computing by using BOINC framework has been applied to many scientific projects such as SETI@home, Einstein@Home, and IBM World Community Grid. The following web pages introduce volunteer computing and application projects.

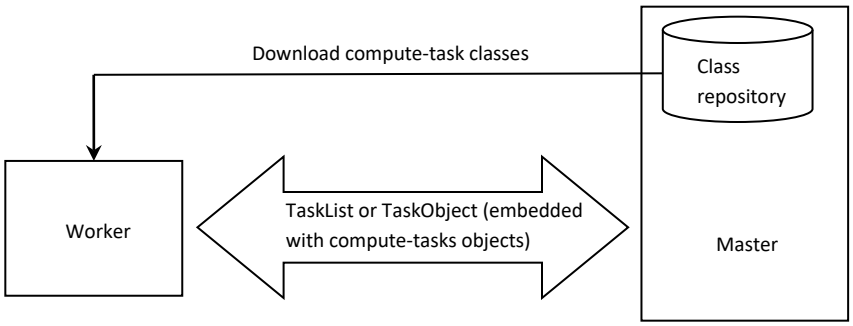
https://boinc.berkeley.edu/, accessed on 15th Jan 2020
https://setiathome.berkeley.edu/, accessed on 15th Jan 2020
https://einsteinathome.org/, accessed on 15th Jan 2020
https://www.worldcommunitygrid.org/discover.action#introduction, accessed on 15th Jan 2020

In this assignment, you are to develop a simplified master/worker framework. The Java networking models and components that you have practised form Week 1 to Week 4 of this unit are enough to develop such a simplified framework. These models and components are client/server model, Java TCP streaming, multi-threading and object serialization. You will need to review these models and components and practise relevant lab projects of these weeks before you start this assignment.

The assignment specification is as follows.

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

The framework consists of a Master (i.e. server), a number of Workers (i.e. clients) and a Class Repository in the Master. The framework is depicted in the following diagram. The framework is a generic computing architecture because the Master and Workers just need to know the interaction contract in advance so that they can interact with each other via the framework. The specification of the interaction contract is as follows.



1. The interaction contract

The interaction contract between a Worker and the Master consists of:

1. The `Task` interface defines two standard methods that every compute-task must implements.

```
package Contract;
public interface Task {
    public void executeTask();
    public Object getResult();
}
```

2. The `TaskList` class is a container that holds the titles and the class names of available compute-tasks.

```
package Contract;
import java.io.Serializable;
public class TaskList implements Serializable{
    private String AvailableTasks[];
    private String TaskClassName[];
    public String[] getAvailableTasks() {
        return AvailableTasks;
    }
    public void setAvailableTasks(String[] AvailableTasks) {
        this.AvailableTasks = AvailableTasks;
    }
    public String[] getTaskClassName() {
        return TaskClassName;
    }
    public void setTaskClassName(String[] TaskClassName) {
        this.TaskClassName = TaskClassName;
    }
}
```

3. The `TaskObject` class is a container that holds a particular compute-class object, its ID and credit.

```
package Contract;
import java.io.Serializable;
public class TaskObject implements Serializable{
    private Integer TaskID=0;
    private Integer Credit=0;
    private Task TObject=null;
    public TaskObject() {
    }
```

```
        public Integer getTaskID() {
            return TaskID;
        }
        public void setTaskID(Integer TaskID) {
            this.TaskID = TaskID;
        }
        public Integer getCredit() {
            return Credit;
        }
        public void setCredit(Integer Credit) {
            this.Credit = Credit;
        }
        public Task getTObject() {
            return TObject;
        }
        public void setTObject(Task TObject) {
            this.TObject = TObject;
        }
    }
```

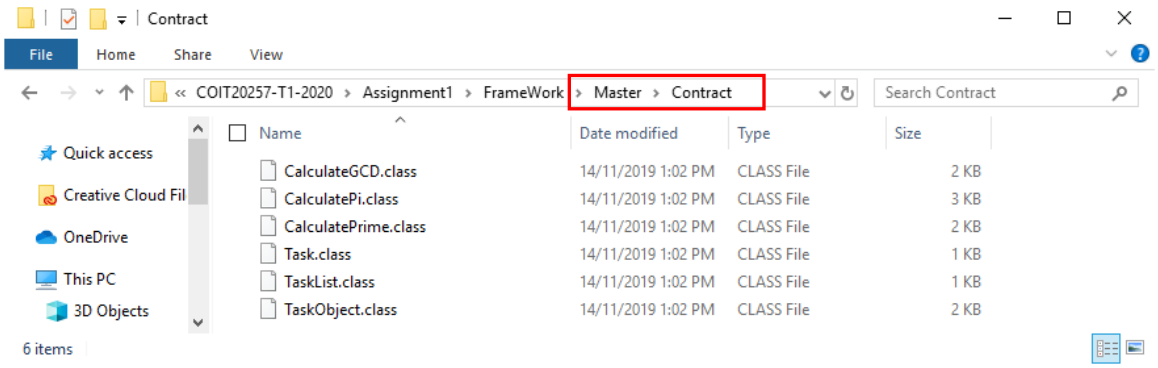
The above interface and classes form a complete interaction contract between the Master and Workers.

2. The compute-task

The Master has a class repository that saves the Java classes of available compute-tasks. A 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. A compute-task must implement `java.io.Serializable` interface as well so that the compute-task can be transferred between the Master and a worker over the network. The structure of a compute-class is as follows.

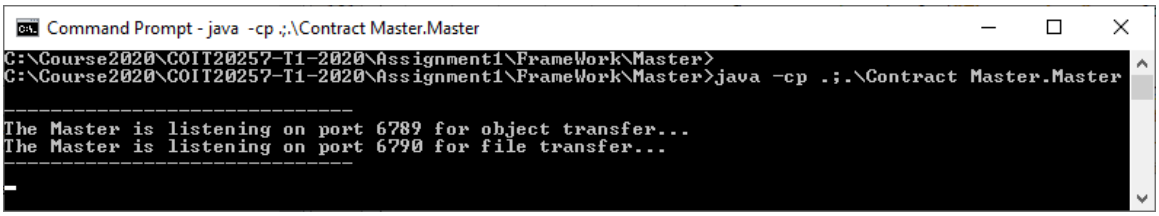
```
public class CalculatePi implements Task, Serializable{
        ……
        @Override
        public void executeTask() {
        //The implementation of method
        ……
        }
        @Override
        public Object getResult() {
        //The implementation of method
        ……
        }
        //may have other methods
        ……
}
```

The following screenshot shows that 3 compute-tasks `CalculatePi.class`, `CalculatePrime.class` and `CalculateGCD.class` (Greatest Common Divisor) along with the aforementioned `Task` interface, the `TaskList` class and `TaskObject` class are saved in the class repository.

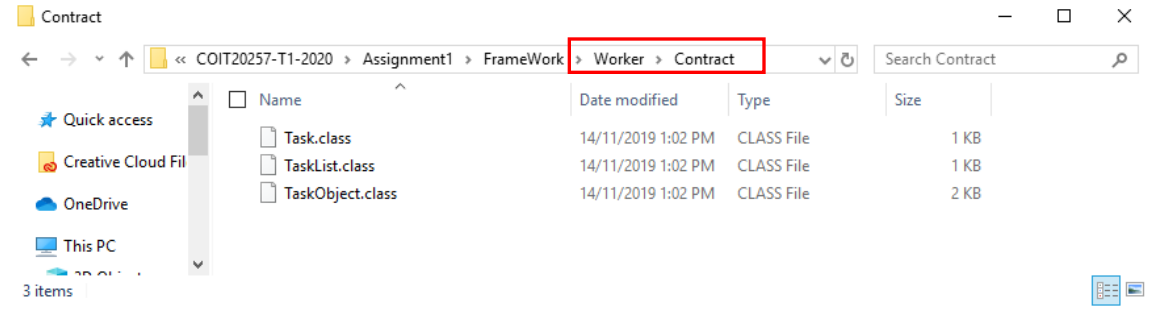


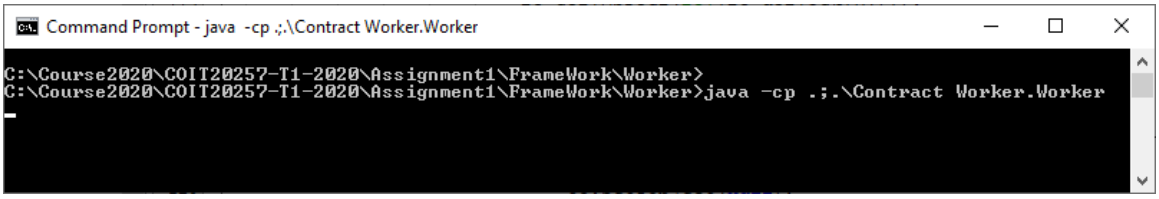
3. The interaction workflow of the framework

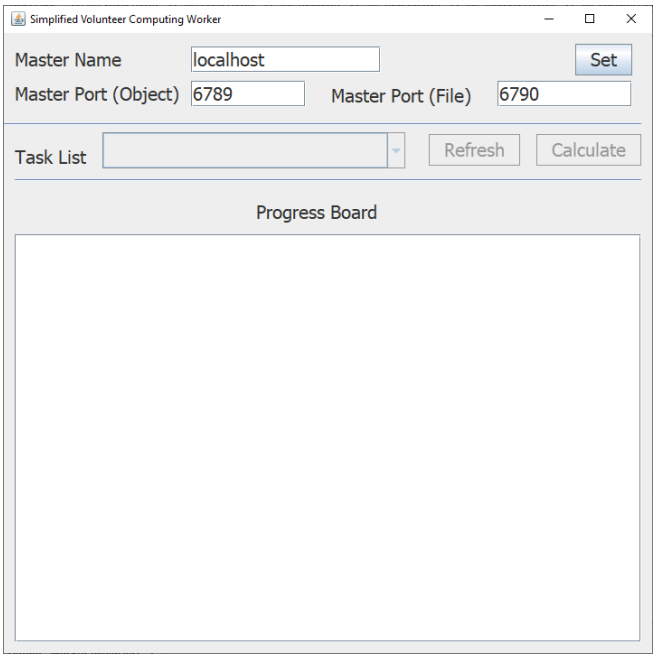
The following screenshot show that when the Master starts, it is listening on two ports for object or file transfer.



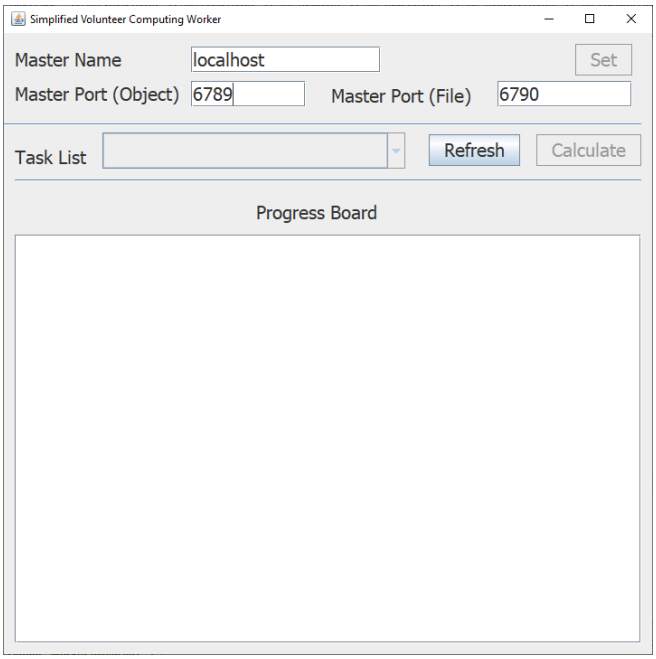
A Worker provides a user frontend to access the remote Master. A Worker just needs to know the interaction contract only before starting volunteer computing. The following screenshots show the contract directory in a Worker and the start of a Worker.



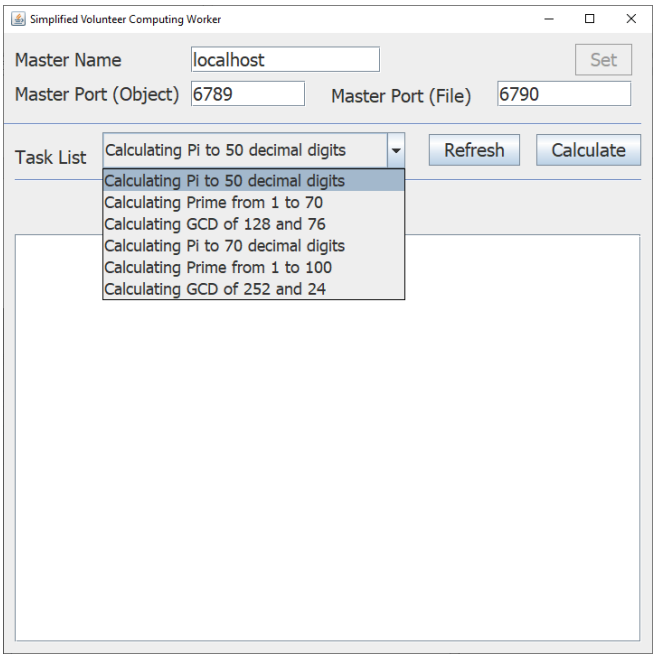




On the Worker frontend as shown in the following screenshot, clicking the Set button will establish two TCP connections to the Master on the given ports, one for object transfer and the other for file transfer, and enable the Refresh button.



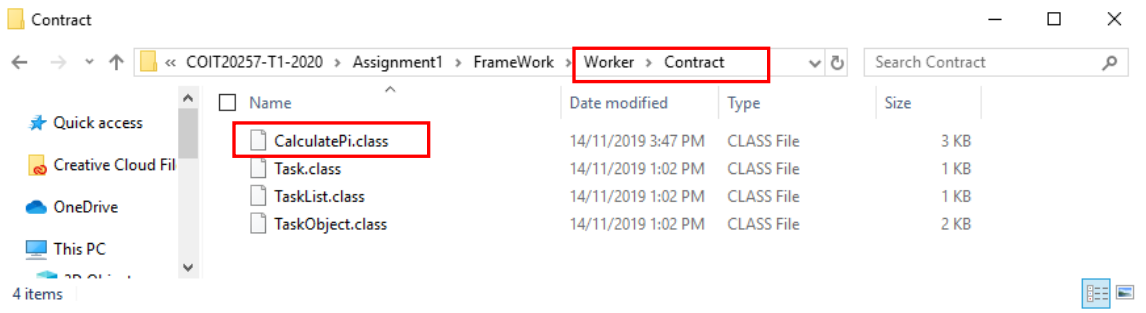
Clicking the Refresh button, the worker will send an empty TaskList object to the Master and receive the same TaskList object back with the available compute-tasks that is filled by the Master. After that the Calculate button is enabled and the Worker is ready to ask for a compute-task to calculate.

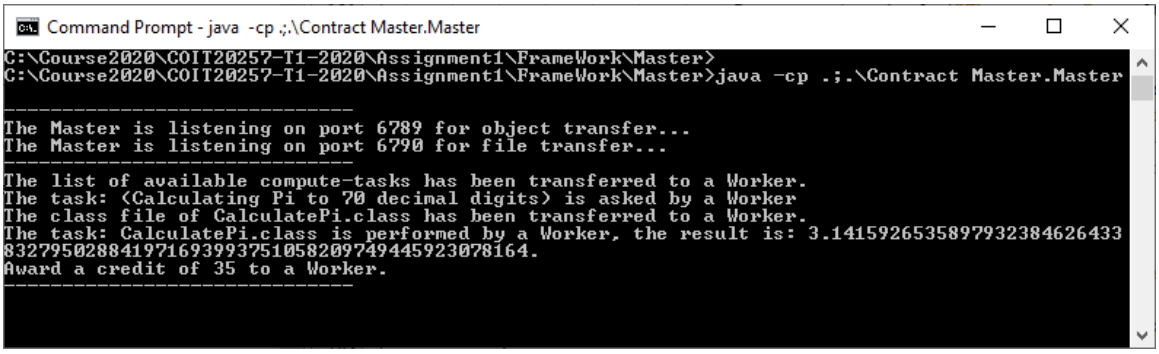


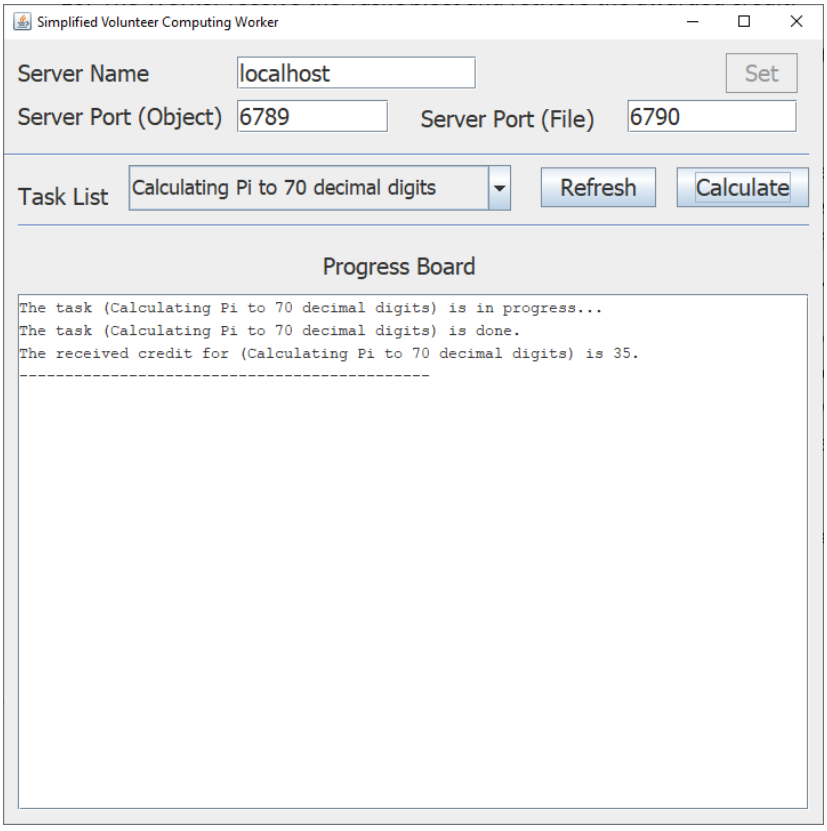
Selecting a task e.g. 'Calculating Pi to 70 decimal digits' and clicking the Calculate button, the following interaction will happen between the Worker and the Master.

1. The Worker downloads the class file of the selected compute-task (e.g. `CalculatePi.class`) from the Master
   Note: the downloading is automated by the Worker, not manually by a user.
2. The worker creates a TaskObject and sets the selected task ID on the TaskObject and then send the TaskObject to the Master.
3. The Master receives the TaskObject, creates the compute-task object (e.g. `CalculatePi`) according to the compute-task ID.
4. The Master sets the compute-task object on the TaskObject and sends the TaskObject to the Worker.
5. The Worker receives the TaskObject and gets the compute-task (e.g. `CalculatePi`) from the TaskObject.
6. The compute-task object (e.g. `CalculatePi`) is cast (be deserialized) into the `Task` interface type and its `executeTask()` is called.
7. The Worker sends the same TaskObject to the Master, which includes the computing results now.
8. The Master receives the TaskObject and retrieves the results.
9. The Master sets a credit on the TaskObject and sends it to the Worker.
10. The Worker receives the TaskObject and retrieves the awarded credit.

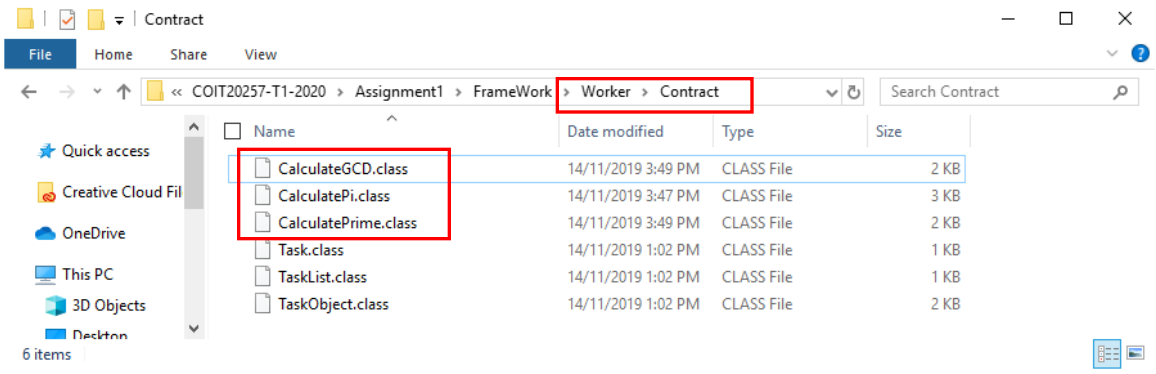
The output of the whole interaction above is shown in the following screenshots.

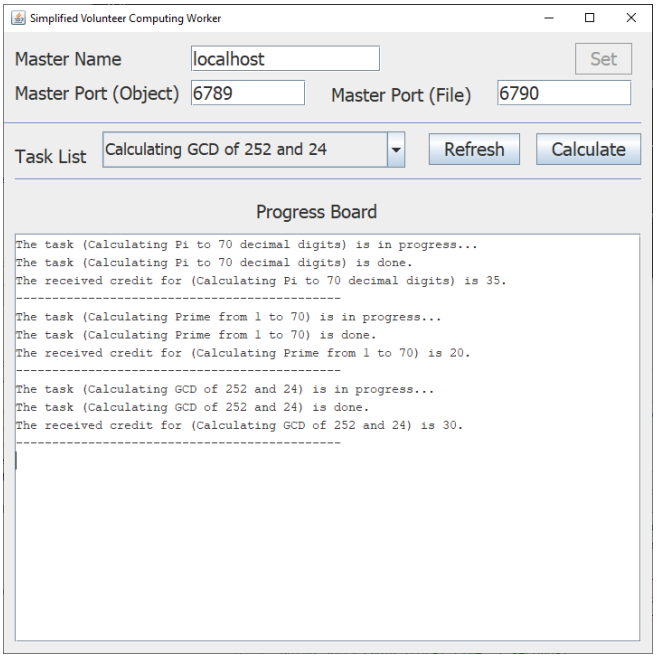


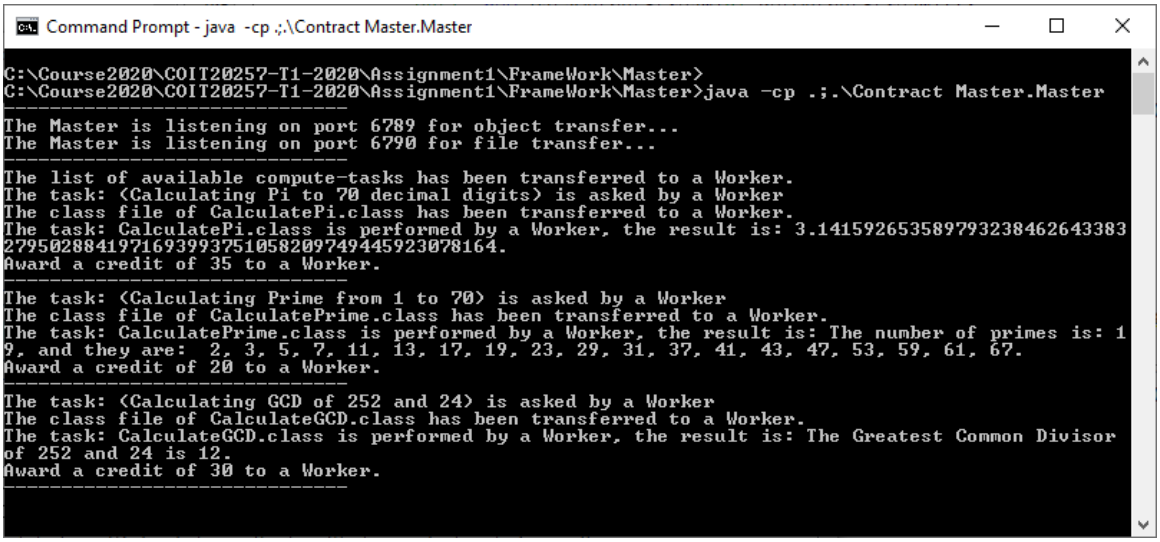




In the same way the Worker can ask for other tasks to perform as shown in the following screenshots.







4. The implementation

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. The Master must be multi-threaded and follow the 'thread-per-connection' architecture (reference Week-4 contents). The communication between the Master and the Worker must use TCP streaming by using Java TCP API `Socket` and `ServerSocket` as described in Week-2 contents of this unit and also online at, https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html, and https://docs.oracle.com/javase/8/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 Worker; graphic user interface is required;
2. A Java application to implement the Master; and
3. A number of Java class to implement the processing threads when necessary.

   Note: to demonstrate your competence of concurrency programming, you will need to make an analysis on when concurrency is needed. Marks will be deducted if concurrency is not identified or necessary multithreading is not used.

4. A number of Java classes to implement *Calculate Pi*, *Calculate Primes* and *Calculate the Greatest Common Divisor* tasks.

Note: to simulate Master and Worker 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 Master 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 Part 2 of marking criteria below.

Submission

You need to provide the following files in your submission.

1. Files of Java source code of the Master, the Worker and the processing threads 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 IDE from Oracle.

2. The compiled Java class files of the source code. These Java classes must be runnable on the standard Java Runtime Environment (JRE) from Oracle (http://www.oracle.com/technetwork/java/index.html).

Note: an easy way to provide the source code and executables is to submit them in a NetBeans project.

3. A Microsoft Word document to address the issues as specified in 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: Java TCP streaming, Multi-threading and Object Serialization Programming	21
1. Whether the project can be compiled by JDK or NetBeans IDE and executed by JRE	2
2. Whether the given `Task` interface, `TaskList` and `TaskObject` classes are properly used as the unique communication contract between Workers and the Master	3
3. Whether the 3 compute-tasks can be successfully transferred between Workers and the Master	3
4. Whether the Master structure is sound as a TCP server	2
5. Whether multithreading is used for the Master when concurrency exists	2
6. Whether the Worker's user interface is fully implemented	2
7. Whether the 3 compute-tasks can be successfully executed by a Worker and can return correct results to the Master	3
8. Whether TCP streaming is correctly used for file transfer	2

9. Whether TCP streaming is correctly used for object transfer	2
Part 2: Program use and test instruction	9
1. Whether the program compiling and installation is clearly described	2
2. Whether the class repository in the Master is clearly described	1
3. Whether the test instruction covers all 3 compute-tasks	3
4. Whether the necessary screenshots have been provided and helpful for the test.	2
5. Whether the source code is readable and includes enough inline comments.	1
Sub Total for Assignment 1	30
Late Penalty	-1.5 (5%) each calendar day (either full or partial)
Plagiarism Related Penalty	
Total for Assignment 1	