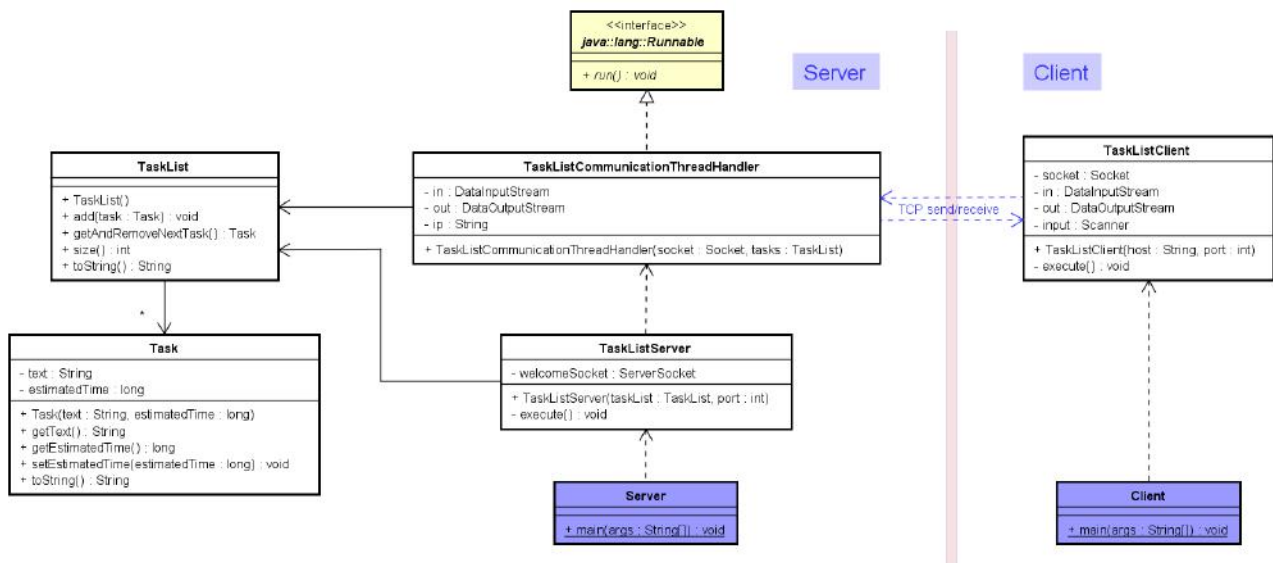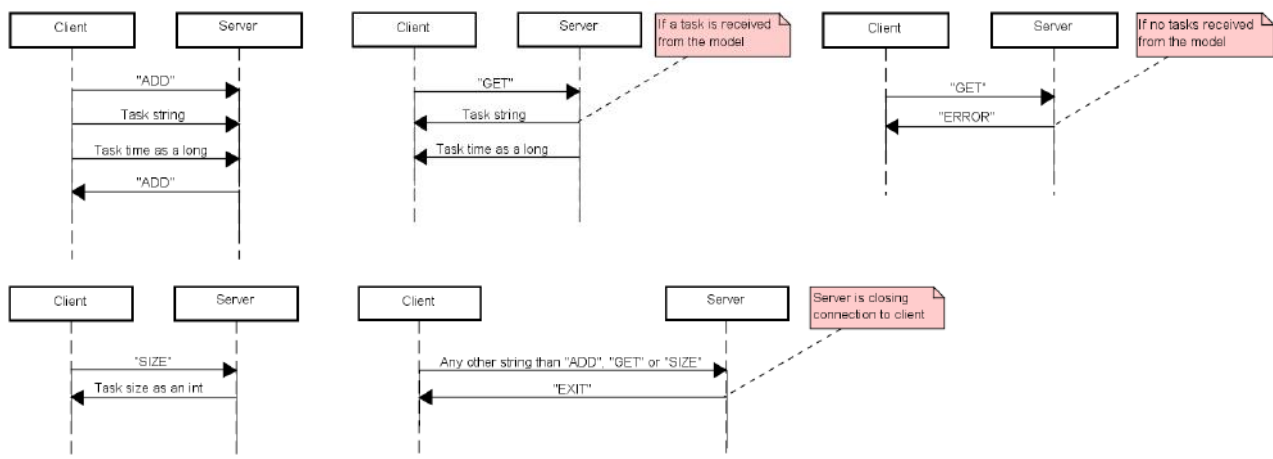## Exercise: A shared task list (TCP sockets)

The communication protocol between client and server is the following:

Step 1: Implement the Model

Implement classes `Task` and `TaskList` (or copy from Appendices A and B in this document). `TaskList` has a collection of `Task`'s and methods for adding to the end, to remove from the start and a getting the size. Note that the list may be accessed by several threads i.e. all methods should be `synchronized`.

Step 2A: Implement the Server side (Thread handler)

Implement class `TaskListCommunicationThreadHandler` implementing `Runnable` with method `run` having a loop reading a string from the client, "ADD", "GET", "SIZE" or any other string to exit. Follow the communication protocol as presented in the five diagrams above.

Note that the streams are `DataInputStream` and `DataOutputStream`, where methods `readUTF`, `readLong` and `readInt` are used when reading strings, long and integers. Methods `writeUTF`, `writeLong` and `writeInt` are used when writing strings, long and integers.

Step 2B: Implement the Server side (TaskListServer)

Implement class `TaskListServer` with method `execute` having an infinite loop in which a client socket is created (`ServerSocket` method `accept()`) and a thread (with a `TaskListCommunicationThreadHandler` object) is created and started.

Step 2C: Implement the Server side (Server main)

Implement class `Server` with a `main` method, creating a `TaskList` and a `TaskListServer` and calling `execute`.

Step 3: Implement the Client side

Step 3A: Implement the Client side (TaskListClient)

Implement class `TaskListClient`.

a) The constructor is creating a connection to server (initializing instance variables) and calling the private method `execute`.

b) Method `execute` creates a loop in which you make a simple menu and read from keyboard if you want to execute ADD, GET, SIZE or EXIT. Depending on the selected case, follow the communication protocol to communicate with the server. In case of ADD, you read from keyboard one more string representing the task and one long representing the estimated time, before sending and receiving. Make sure to include an `input.nextLine()` to clear the keyboard stream between reading a primitive type value and reading a string.

Step 3B: Implement the Client side (Client main)

Implement class `Client` with a main method, creating a `TaskListClient` and calling `execute`.

Client Example Run. **Black bold** is input from keyboard, **purple bold** is what has been received by server:

```
1) Add a task
2) Get a task
3) Get task size
Any other number) Exit
1
Enter the task: Make SDJ2 exercises
Enter the estimated time: 300
Server> Task has been added
1) Add a task
2) Get a task
3) Get task size
Any other number) Exit
1
Enter the task: Check Facebook
Enter the estimated time: 250
Server> Task has been added
1) Add a task
2) Get a task
3) Get task size
Any other number) Exit
2
Server> Make SDJ2 exercises: 300
1) Add a task
2) Get a task
3) Get task size
Any other number) Exit
```

Appendix A – Class Task

```
public class Task
{
   private String text;
   private long estimatedTime;

   public Task(String text, long estimatedTime)
   {
      this.text = text;
      this.estimatedTime = estimatedTime;
   }

   public String getText()
   {
      return text;
   }

   public long getEstimatedTime()
   {
      return estimatedTime;
   }

   public void setEstimatedTime(long estimatedTime)
   {
      this.estimatedTime = estimatedTime;
   }

   public String toString()
   {
      return text + ", (Estimated time = " + estimatedTime + ")";
   }
}
```

Appendix B – Class TaskList

```java
import java.util.ArrayList;

public class TaskList
{
   private ArrayList<Task> tasks;

   public TaskList()
   {
      tasks = new ArrayList<Task>();
   }
   public synchronized void add(Task task)
   {
      tasks.add(task);
   }
   public synchronized Task getAndRemoveNextTask()
   {
      if (tasks.size() > 0)
      {
         return tasks.remove(0);
      }
      return null;
   }
   public synchronized int size()
   {
      return tasks.size();
   }
   public synchronized String toString()
   {
      return "Tasks=" + tasks;
   }
}
```