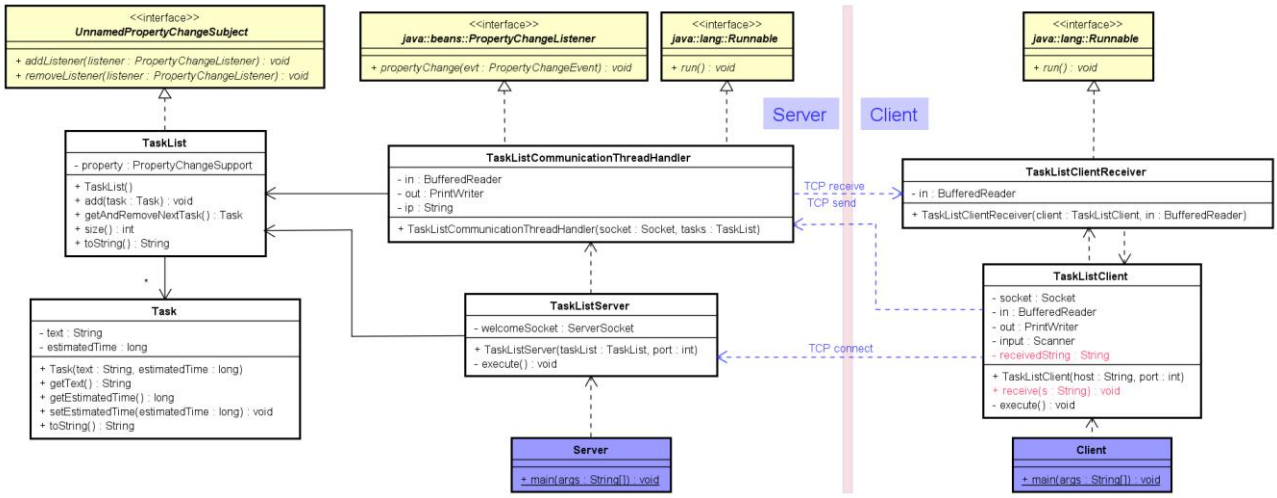
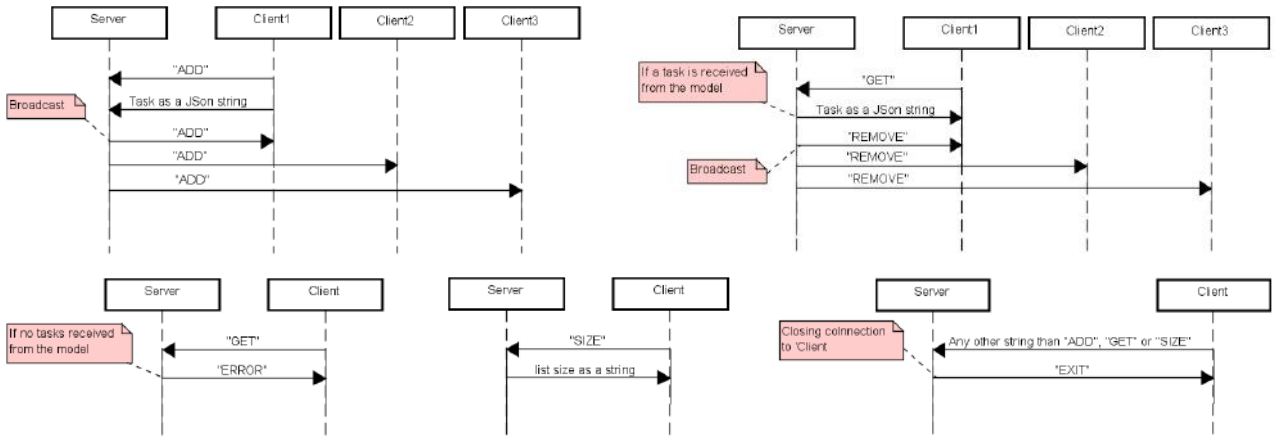
Exercise: A shared task list (TCP sockets – request/reply and broadcasts)



The communication protocol between client and server is the following:



A client starts the communication by sending a string ("ADD", "GET", "SIZE" or any other string to Exit). In case of "ADD" the client sends one more string, a Task object converted to a JSon string. In case of "ADD" and of "GET" where a task has been replied (as a JSon string) to the calling client, all clients including the caller receives a string "ADD" or "REMOVE" for "ADD" and "GET", respectively. In all other cases ("GET" when the list is empty, "SIZE" and any other string to Exit), the server reply the calling client with a string ("ERROR", The size as a string, or "EXIT").

Step 1: Implement the Model

Implement classes `Task` and `TaskList` (or modify the model classes from Appendices A and B in this document). Class `Task` can be taken directly, but class `TasksList` is not a Subject in the Observer pattern yet, and you have to include this part. Fire events in the add and get methods (use property names "ADD" and "REMOVE" for these events)

Step 2: Implement the Server side

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 `BufferedReader` and `Printwriter` but it is ok to use `DataInputStream` and `DataOutputStream` as long as you use the same on both Server and Client.

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.
Let it be a listener for the `TaskList` with a `propertyChange` method sending the broadcast messages.

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 (TaskListClientReceiver)

Implement class `TaskListClientReceiver` implementing `Runnable`. In the `run` method, make an infinite loop reading a line from server and calling the receive method in the `TaskListClient` class.

Step 3B: Implement the Client side (TaskListClient)

Implement class `TaskListClient`.

a) The constructor is creating a connection to server and calling the private method `execute`. Further, creates and start a thread with a `TaskListClientReceiver` argument.

b) Method `receive` (to be called from the receiver thread) is checking the string to see if it is "ADD" or "REMOVE" representing broadcasts in which case you just print it out. Any other string would be a reply from a request in which case you update the `receivedString` instance variable and call `notify()` to get the thread waiting for the reply out of its wait state.

c) 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 creating a Task, converting it to a JSon string and sending. Make sure to include an `input.nextLine()` to clear the keyboard stream between reading a primitive type value and reading a string.
Note: In this method you do not read from server, but some of the cases you have to let the thread wait until you have received the reply from the receiver thread. Let it wait while `receivedString == null`. After the wait-loop, set `receivedString` back to `null`.

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

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

Example Run. **Black bold** is input from keyboard, **purple bold** is what has been received as a reply by server, and **green bold** are broadcasts from server (send to all clients). Note that Client 1 first adds two tasks and then client 2 get a task:

```
CLIENT 1:
---------
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 broadcast> ADD
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 broadcast> ADD
1) Add a task
2) Get a task
3) Get task size
Any other number) Exit
Server broadcast> REMOVE
```

```
CLIENT 2
---------
1) Add a task
2) Get a task
3) Get task size
Any other number) Exit
Server broadcast> ADD
Server broadcast> ADD
2
Server> Make SDJ2 exercises: 300
Server broadcast> REMOVE
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 (in a non-Observer version)

```
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;
   }
}
```