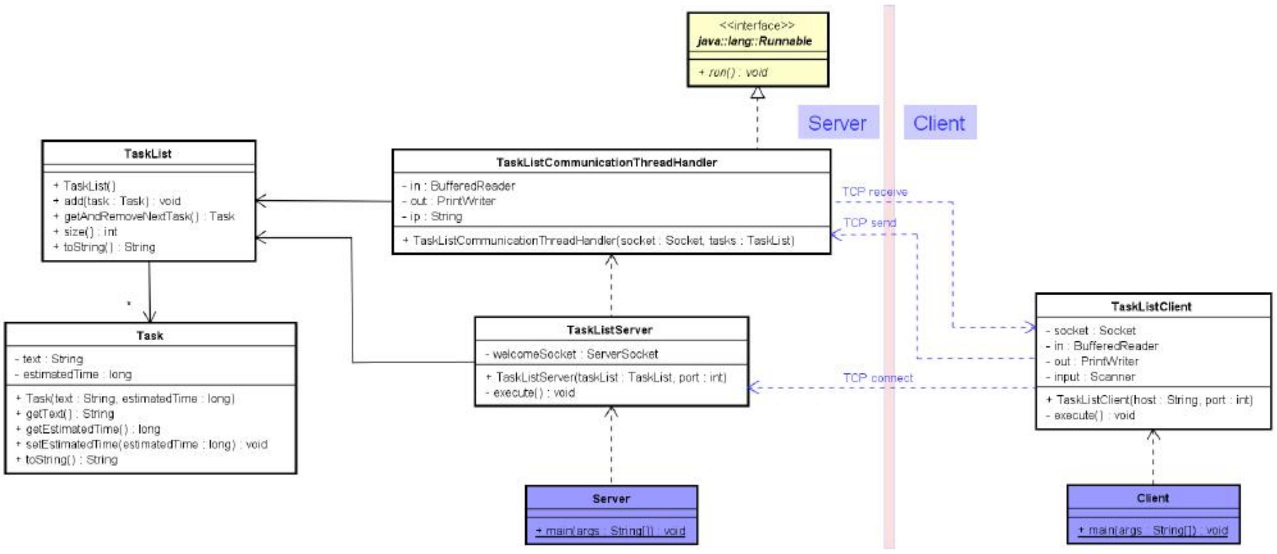
Exercises, sockets SDJ2

Exercise: A shared task list (TCP sockets – using JSon)

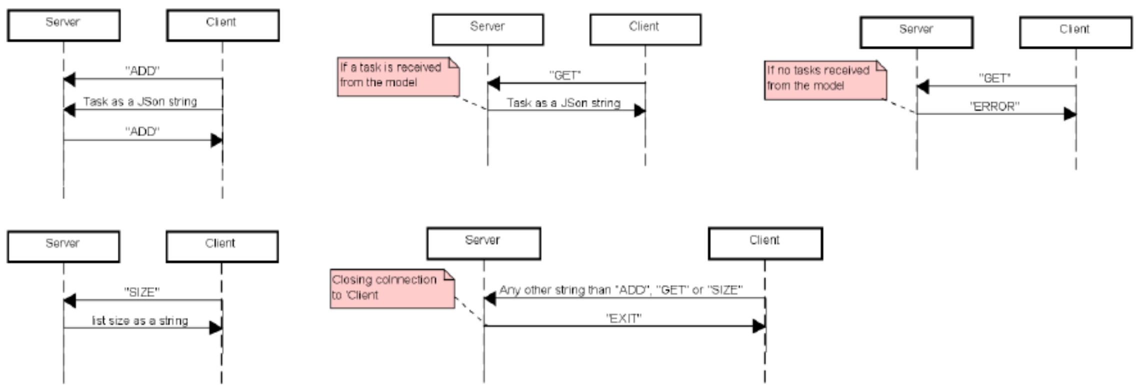
Note: If you made a TaskList exercise without sending JSon strings, then there are only minor changes to complete this one. This time, when sending a Task (in "ADD" and "GET"), you convert the Task to a JSon string, send it and on the receiving side you convert it back to a Task object. If you did not make this exercise, then no worries – just follow this exercise step by step.

Create one IntelliJ module for the server and another module for the client. Add the gson library/jar to both modules. The class diagram is the following (Note that streams are `BufferedReader` and `Printwriter`. Alternatively, use `DataInputStream` and `DataOutputStream` on both sides):



<<interface>> java::lang::Runnable
+ run() : void
Server
Client
TaskList
+ TaskList()
+ add(task : Task) : void
+ getAndRemoveNextTask() : Task
+ size() : int
+ toString() : String
TaskListCommunicationThreadHandler
- in : BufferedReader
- out : PrintWriter
- ip : String
+ TaskListCommunicationThreadHandler(socket : Socket, tasks : TaskList)
TCP receive
TCP send
Task
- text : String
- estimatedTime : long
+ Task(text : String, estimatedTime : long)
+ getText() : String
+ getEstimatedTime() : long
+ setEstimatedTime(estimatedTime : long) : void
+ toString() : String
TaskListServer
- welcomeSocket : ServerSocket
+ TaskListServer(taskList : TaskList, port : int)
- execute() : void
TCP connect
TaskListClient
- socket : Socket
- in : BufferedReader
- out : PrintWriter
- input : Scanner
+ TaskListClient(host : String, port : int)
- execute() : void
Server
+ main(args : String[]) : void
Client
+ main(args : String[]) : void

The full communication protocol between client and server is the following:



Server
Client
"ADD"
Task as a JSon string
"ADD"
If a task is received from the model
"GET"
Task as a JSon string
If no tasks received from the model
"GET"
"ERROR"
"SIZE"
list size as a string
Closing connection to Client
Any other string than "ADD", "GET" or "SIZE"
"EXIT"

In the "ADD" mode, client is sending two strings, "ADD" and a JSon string of the Task to add, and the server replies with the string "ADD". In all other cases the protocol is a simple send/receive of srings, the client sends a string "GET", "SIZE" or any other string (not "ADD") to Exit. The server replies with a string, either a JSon string of the Task to get, "ERROR" if there are no more Task to get, the size as a string, or "EXIT".

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 getting the size. Note that the list may be accessed by several threads i.e. all methods are `synchronized`.

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. In "GET" the server receives a JSon string and you have to deserialize to get the Task and in "ADD" if there are more tasks on the list, you convert the task into a JSon string before sending.

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 converting to a JSon string and sending and receiving the "ADD" string from server. 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> 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> ADD
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

```
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;
   }
}
```