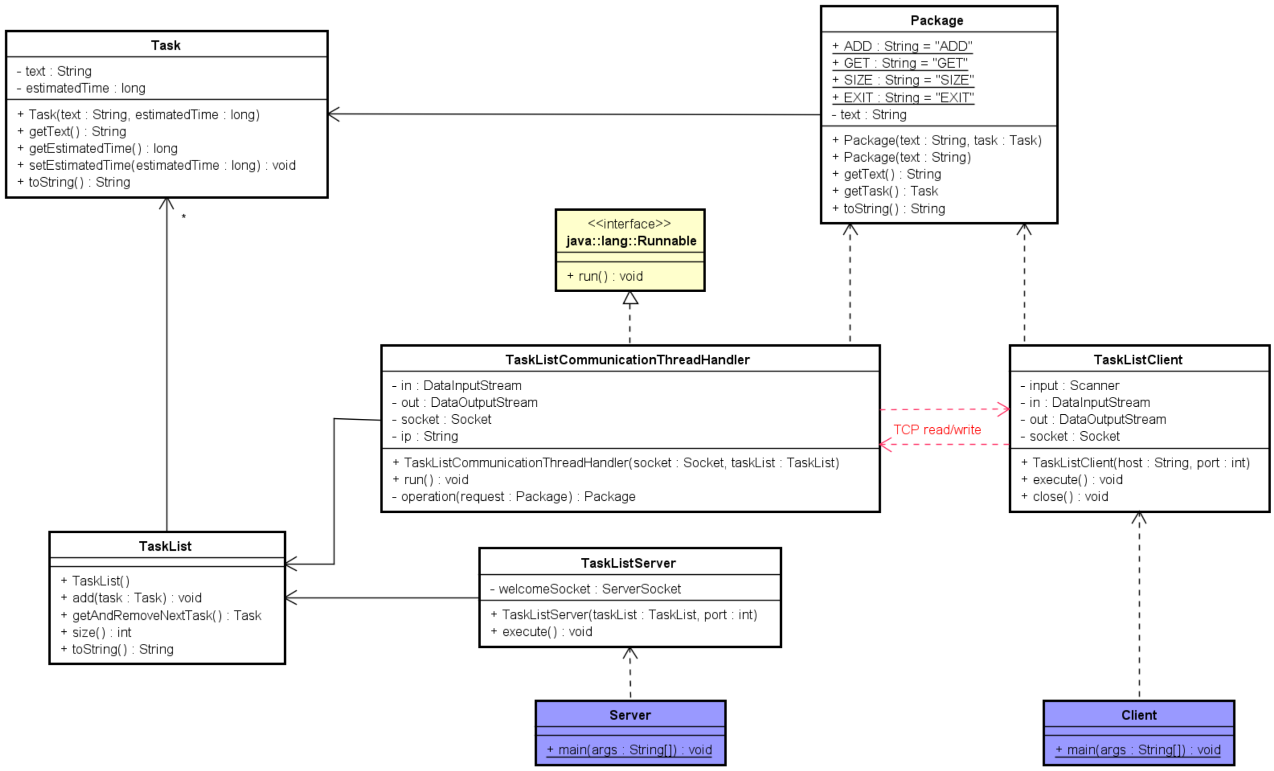
Exercise 06.03 – A shared task list (TCP sockets)



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. In the class diagram (and implementation in appendix) the collection is shown as an `ArrayList` but it is better / more efficient (and legal) to use a queue instead (in which `getAndRemoveNextTask` is calling `dequeue` and `add` is calling `enqueue`).

Note that the task list may be accessed by several threads i.e. all methods should be `synchronized`.

Step 2: Implement the Package to use when reading and writing via TCP

Implement class `Package` (or copy from Appendix C). The package contain a text and a task. The one-argument constructor sets the task to `null`.

The way to use it (from the client):

```
Package request = new Package(Package.ADD, task);
Package request = new Package(Package.GET);
Package request = new Package(Package.SIZE);
Package request = new Package(Package.EXIT);
```

The way to use it (from the server):

```
Package reply = new Package(Package.ADD);
Package reply = new Package(Package.GET, task);
Package reply = new Package("NO TASKS - EMPTY TASK LIST");
Package reply = new Package(Package.SIZE + "=" + size);
Package reply = new Package(Package.EXIT);
Package reply = new Package("WRONG FORMAT");
```

Step 3: Implement the Server side

Step 3A: Implement the Server side (Thread handler)

Implement class `TaskListCommunicationThreadHandler`.

a) implementing `Runnable`

b) The constructor is initializing instance variables

c) Method `run` with a loop 1) reading a `Json` string from the client, 2) converting this to a `Package` object, 3) Depending on the request the `TaskList` object is accessed and a reply is generated (this part is convenient made in the private method `operation`). 4) Converting the package to Json and 5) sending it to the client. End the loop if the text part of the message is "EXIT".

d) Method `operation` may be implemented with a switch reading the text part of the `Package`. This is either "ADD", "GET", "SIZE" or "EXIT". If it is "ADD" then the remaining of the package is the task to add in the task list. The first three calls methods in the `TaskList` object while EXIT just return EXIT and end the loop in the `run` method.

Step 3B: Implement the Server side (TaskListServer)

Implement class `TaskListServer`.

a) The constructor is initializing instance variables

b) Method `execute` creates 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 3C: Implement the Server side (Server main)

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

Step 4: Implement the Client side

Step 4A: Implement the Client side (TaskListClient)

Implement class `TaskListClient`.

a) The constructor is initializing instance variables

b) Method `execute` creates a loop in which you 1) make a menu to distinguish between ADD, GET, SIZE and EXIT. If ADD has been selected then type in the task text and the estimated time 2) create a package 3) convert this to Json, 4) send this to the server, 3) reads the reply from server 5) convert from Json to Package and 6) print it out. If you get the string "EXIT" from server then end the loop.

c) Method `close` closes the socket and the keyboard stream (Scanner object)

Step 4B: Implement the Client side (Client main)

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

Example Run (client side):

```
Connected to server: localhost at port 6789
1) Type 1 for "ADD"
2) Type 2 to "GET"
3) Type 3 to get the "SIZE"
0) Type 0 to "EXIT"
Enter choice: 1
Enter task: Check Facebook
Enter estimated time: 200
Client> {"text":"ADD","task":{"text":"Check
facebook","estimatedTime":200}}
Server> {"text":"ADD"}
Task: ADD
```

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()
   {
      Task task = null;
      if (tasks.size() > 0)
      {
         task = tasks.get(0);
         tasks.remove(0);
      }
      return task;
   }
   public synchronized int size()
   {
      return tasks.size();
   }
   public String toString()
   {
      return "Tasks=" + tasks;
   }
}
```

Appendix C – Class Package

```
public class Package
{
   public static final String ADD = "ADD";
   public static final String GET = "GET";
   public static final String SIZE = "SIZE";
   public static final String EXIT = "EXIT";

   private String text;
   private Task task;

   public Package(String text, Task task)
   {
      this.text = text;
      this.task = task;
   }

   public Package(String operation)
   {
      this(operation, null);
   }

   public String getText()
   {
      return text;
   }

   public Task getTask()
   {
      return task;
   }

   public String toString()
   {
      if (task == null)
         return "" + text;
      else
         return text + " " + task;
   }
}
```