Exercises, Sockets SDJ2

Exercise: A remote Student list: Server side

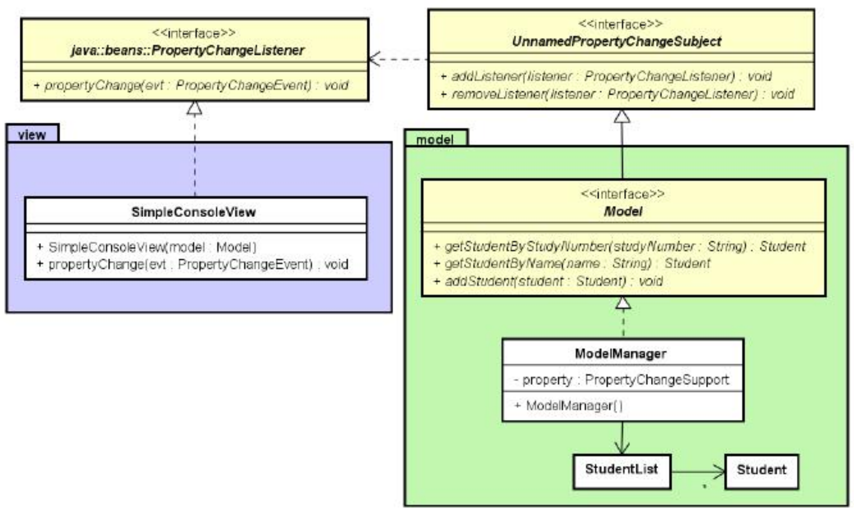
The purpose for this exercise is to

1) Implement a StudentList application (as a simple console version)
2) Add functionality to provide a remote access, i.e. as a server (using TCP sockets and JSon strings)

Step 1: **Done!** Uploaded as `StudentList-Application-SingleUser.zip`

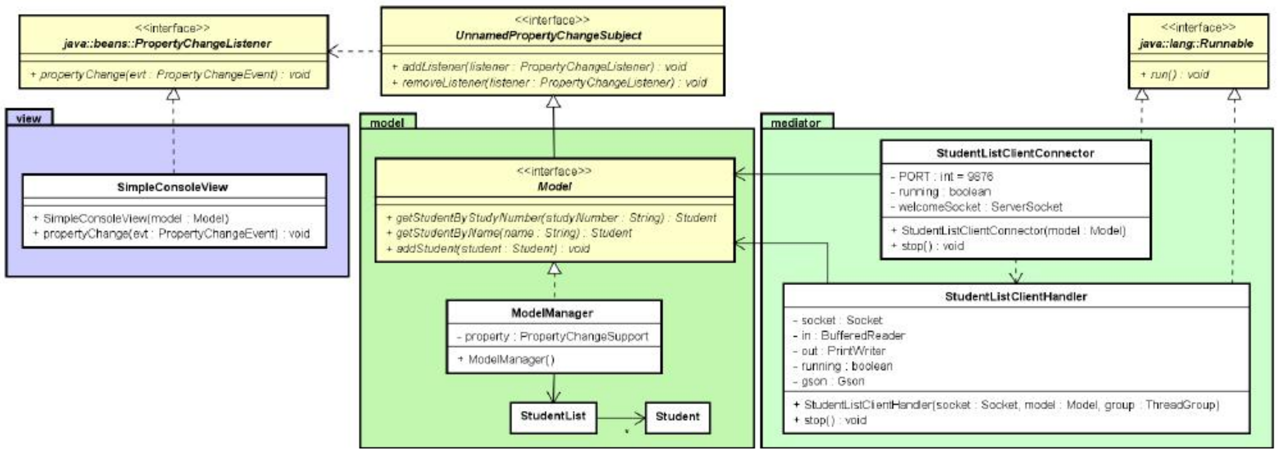
The uploaded zip file contains an application of a single user Student list application as shown below.

Note: `SimpleConsoleView` is listening for events fired from the model and do not actively call any methods. Because no one calls model methods (yet), no output will be shown when running the application.



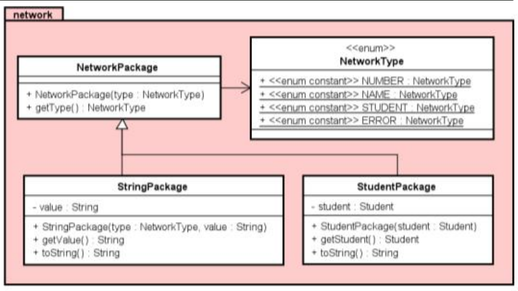
<<interface>> java::beans::PropertyChangeListener
+ propertyChange(evt : PropertyChangeEvent) : void
<<interface>> UnnamedPropertyChangeSubject
+ addListener(listener : PropertyChangeListener) : void
+ removeListener(listener : PropertyChangeListener) : void
view
SimpleConsoleView
+ SimpleConsoleView(model : Model)
+ propertyChange(evt : PropertyChangeEvent) : void
model
<<interface>> Model
+ getStudentByStudyNumber(studyNumber : String) : Student
+ getStudentByName(name : String) : Student
+ addStudent(student : Student) : void
ModelManager
- property : PropertyChangeSupport
+ ModelManager()
StudentList
Student

Step 2: Modify the single user system to include a Server



<<interface>> java::beans::PropertyChangeListener
+ propertyChange(evt : PropertyChangeEvent) : void
<<interface>> UnnamedPropertyChangeSubject
+ addListener(listener : PropertyChangeListener) : void
+ removeListener(listener : PropertyChangeListener) : void
<<interface>> java::lang::Runnable
+ run() : void
view
SimpleConsoleView
+ SimpleConsoleView(model : Model)
+ propertyChange(evt : PropertyChangeEvent) : void
model
<<interface>> Model
+ getStudentByStudyNumber(studyNumber : String) : Student
+ getStudentByName(name : String) : Student
+ addStudent(student : Student) : void
ModelManager
- property : PropertyChangeSupport
+ ModelManager()
StudentList
Student
mediator
StudentListClientConnector
- PORT : int = 9876
- running : boolean
- welcomeSocket : ServerSocket
+ StudentListClientConnector(model : Model)
+ stop() : void
StudentListClientHandler
- socket : Socket
- in : BufferedReader
- out : PrintWriter
- running : boolean
- gson : Gson
+ StudentListClientHandler(socket : Socket, model : Model, group : ThreadGroup)
+ stop() : void

All classes in package `network` are given in the uploaded file and may be used when sending and receiving messages.



network
NetworkPackage
+ NetworkPackage(type : NetworkType)
+ getType() : NetworkType
<<enum>> NetworkType
+ <<enum constant>> NUMBER : NetworkType
+ <<enum constant>> NAME : NetworkType
+ <<enum constant>> STUDENT : NetworkType
+ <<enum constant>> ERROR : NetworkType
StringPackage
- value : String
+ StringPackage(type : NetworkType, value : String)
+ getValue() : String
+ toString() : String
StudentPackage
- student : Student
+ StudentPackage(student : Student)
+ getStudent() : Student
+ toString() : String

Your job is to implement the two classes in the `mediator` package, `StudentListClientConnector` and `StudentListClientHandler`.

- Class `StudentListClientConnector` implements `Runnable`. In the `run` method you
  - Create a ServerSocket
  - Make an infinite loop repeatedly accepting incoming connection from a client, creating a StudentListClientHandler object, adding it to a Thread, setting the thread to a daemon thread and starting the thread.

- Class `StudentListClientHandler` implements `Runnable`. In the `run` method you
    - Read a line from in stream,
    - Convert the line from JSon to a `NetWorkPackage`
    - Get the type and make a switch for the types (NAME, NUMBER, STUDENT, ERROR)
        - If NAME or NUMBER then re-convert from JSon to a `StringPackage`, call the related get method from the model, and create a `StudentPacket` with the returned student.
        - If STUDENT the re-convert for JSon to a `StudentPackage`, call the add method in the model, and create a `StringPacket` of type NAME with the name of the student. If an exception occur then use type ERROR with a string containing the exception message.
        - If ERROR the just reply with the same package
    - Send the reply packet in the out stream