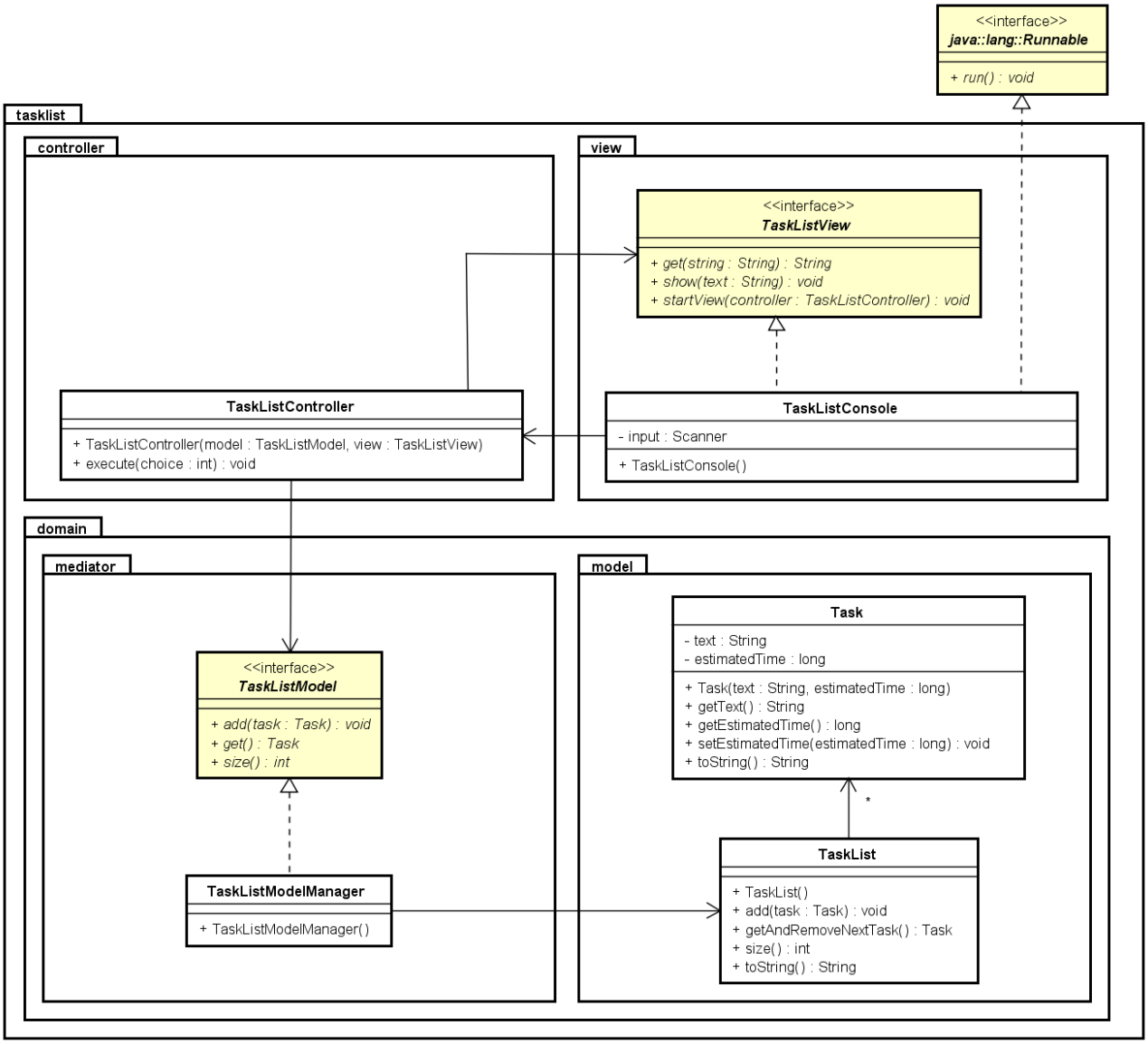
Exercise 10.01

Create a project with the three files `Task`, `TaskList` and `Main` from appendix and run the class `Main`.

The output should look like this:

```
1) Type 1 for "ADD"
2) Type 2 to "GET"
3) Type 3 to get the "SIZE"
0) Type 0 to "EXIT"
Enter choice:
```

The exercise is to convert this in to MVC such that you end up with the classes shown below (exercise given in 5 steps on the following pages):



Step 1 – Model

a) Move model classes (`Task` and `TaskList`) into the package `tasklist.domain.model`

b) Define interface `TaskListModel` in package `tasklist.domain.mediator` as the following

```
package tasklist.domain.mediator;

import tasklist.domain.model.Task;

public interface TaskListModel
{
   void add(Task task);
   Task get();
   int size();
}
```

c) Implement the interface `TaskListModel` in a class `TaskListModelManager` also in package `tasklist.domain.mediator`. This class contain one instance variable of type `TaskList` and each method simply delegates to this object. Make all method `synchronized` such that this class later could be uses in a multithreaded program.

Step 2 – Controller (first part)

a) Define an "empty" controller this way (this makes it easier to take the next step)

```
package tasklist.controller;
public class TaskListController
{
   public void execute(int choice)
   {
      // TODO Auto-generated method stub
   }
}
```

Step 3 – View

a) Define interface `TaskListView` in package `tasklist.view` as the following

```
package tasklist.view;

import tasklist.controller.TaskListController;

public interface TaskListView
{
   String get(String text);
   void show(String text);
   void startView(TaskListController controller);
}
```

b) Implement a class `TaskListConsole` implementing the two interfaces `TaskListView` and `Runnable` - also in package `tasklist.view`. This class contain all the printout and keyboard input from the main method and a `run` method with a loop showing the menu and waiting for keyboard input.

- Two instance variables, a `TaskListController` and a `Scanner`
- Constructor crating the `Scanner` object and setting `TaskListController` to `null`.
- Method `startView` sets the controller, creates a thread and start the thread.
- The `run` method has a loop, the menu prompt/printout, and calling the constructor with the menu choice. Terminate the loop if the choice is 0.
- The `get` method prints the text/argument and return the result from a keyboard input.
- The `show` method prints the text to the console.

Step 4 – Controller (again)

Modify the controller, class `TaskListController`

- Two instance variables, a `TaskListModel` and a `TaskListView`
- Constructor setting both instance variables.
- Method `execute` with a switch for choice.
    - Case 1 / add: call `get` from the view twice to get task text and time. Convert the time into a long and create a `Task` object. Add this to the model and print out using the `show` method in the view.
    - Case 2 / get: call `get` from the model and print out using the `show` method in the view.
    - Case 3 / size: call `size` from the model and print out using the `show` method in the view.
    - Case 4 / exit: call `System.exit`

Step 5 – Main method

Create a class with a `main` method (or modify class `Main`). In the `main` method do the following:

- Create the model (variable of type `TaskListModel` created as an instance of `TaskListModelManager`)
- Create the view (variable of type `TaskListView` created as an instance of `TaskListConsole`)
- Create the controller (a `TaskListController`) giving the model and the view as argument.
- Start the view calling method `startView` giving the controller as argument.

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

Appendix C – Class Main

```
import java.util.Scanner;

public class Main
{
   public static void main(String args[])
   {
      TaskList taskList = new TaskList();
      Scanner input = new Scanner(System.in);
      boolean continueWorking = true;
      while (continueWorking)
      {
         System.out.print("1) Type 1 for \"ADD\"\n"
               + "2) Type 2 to \"GET\""
               + "\n3) Type 3 to get the \"SIZE\"\n"
               + "0) Type 0 to \"EXIT\"\nEnter choice: ");
         int choice = input.nextInt();
         input.nextLine();

         switch (choice)
         {
            case 1:
               String what = "task";
               System.out.print("Enter " + what +": ");
               String taskText = input.nextLine();
               what = "estimated task time";
               System.out.print("Enter " + what +": ");
               String taskTime = input.nextLine();
               long time = -1;
               try
               {
                  time = Long.parseLong(taskTime);
               }
               catch (NumberFormatException e)
               {
               }
               Task task = new Task(taskText, time);
               taskList.add(task);
               System.out.println("ADDED: " + task);
               break;
            case 2:
               task = taskList.getAndRemoveNextTask();
               System.out.println("Task: "+task);
               break;
            case 3:
               int size = taskList.size();
               System.out.println("Size=" + size);
               break;
            case 0:
               System.exit(1);
               break;
         }
         if (choice == 0)
         {
            continueWorking = false;
         }
      }
      input.close();
   }
}
```