# SL Java Programming Project Hand-out

## Stage I – Planning

We will create a prototype, CLI-based system/product that will help keep track of the courses being taken by students and being taught by teachers. This project has the purpose of helping you practice all the programming techniques that you have learned and been exposed to during grade 11.

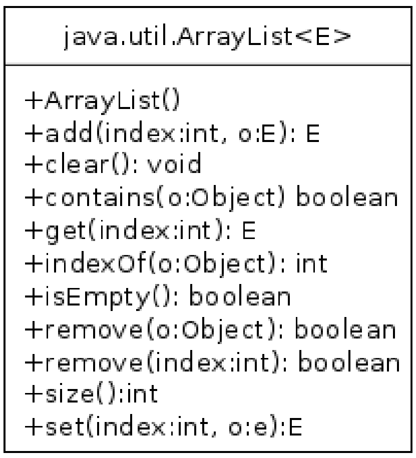
Work on this project stage by stage, taking breaks and carefully documenting your code. For tips on code documentation, check:

- http://www.devtopics.com/13-tips-to-comment-your-code/ .

The data structure to be used for this project is Java's ArrayList. You may find the following web sites useful:

- http://javahungry.blogspot.com/2015/03/difference-between-array-and-arraylist-in-java-example.html
- http://beginnersbook.com/2013/12/java-arraylist/
- http://www.tutorialspoint.com/java/java_arraylist_class.htm
- https://www.mkyong.com/java/how-to-loop-arraylist-in-java/

Figure 1 is a UML class diagram for the ArrayList Java class to assist you during the development of this project.



```
java.util.ArrayList<E>

+ArrayList()
+add(index:int, o:E): E
+clear(): void
+contains(o:Object) boolean
+get(index:int): E
+indexOf(o:Object): int
+isEmpty(): boolean
+remove(o:Object): boolean
+remove(index:int): boolean
+size():int
+set(index:int, o:e):E
```

Figure 1. UML class diagram for the ArrayList Java class.

In a nutshell, our product will do the following with ***students***, ***teachers*** and ***subjects*** (think of these as barebones success criteria):

1. Add
2. List
3. Search
4. Edit
5. Delete
6. and maybe sort

## Stage II – Design/Overview

Considering our rudimentary success criteria from the earlier planning stage, we divide our system into the hierarchical structure like the one depicted figure 2:

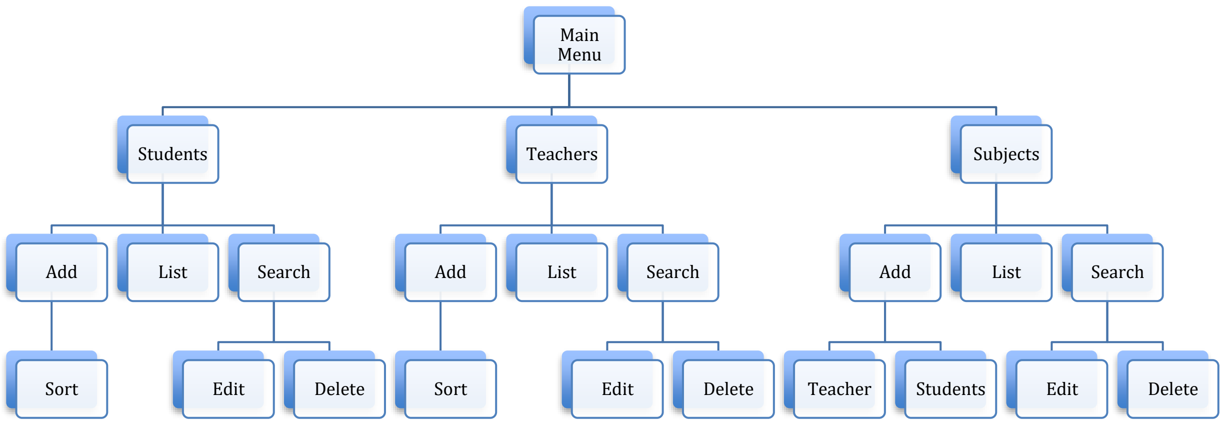


Figure 2. Structure/Top-down diagram for the project.

Study figure 2 carefully. Reflect: why is *Sort* is a sub-task of *Add*? Why are *Edit* and *Delete* sub-tasks of *Search*?

While the above diagram shows ideas for the structure and tasks of our system, we also need to understand completely the problem and its requirements. A defining diagram, which we studied and used in the first quarter of our course, is a suitable tool to organise and outline the inputs, outputs and the processing required to turn the input into the required output.

As an example, table 1 shows the defining diagram for the *Subjects* sub-task or module of our project:

Table 1. Defining diagram for the *Subject* module of the project.

| Input | Processing | Output |
|---|---|---|
| Subject name | Validate name | Subject data input |
| Higher or Standard level? | Validate HL or SL only | |
| Classroom | Validate room | |

From the defining diagram in table 1, we can specify a data dictionary illustrated in table 2:

Table 2. Data dictionary for *Subject*.

| Data item | Variable name & type | Validation |
|---|---|---|
| Subject name | subjectName: String | length > 0 |
| Higher or Standard level | subjectLevel: char | s/h accepted only |
| Classroom | classroom: String | length > 0 |
| Teacher ID | Validate ID (must exist) | Teacher data |
| Student ID(s) | Validate ID (must exist) | Student data |

Likewise a set of defining diagrams for *Student* and *Teacher* can be made, with their matching data dictionaries.

The *Subject* module or task is meant to manage (add/remove) the members of a subject (students and teachers) as well.

From the defining diagrams and data dictionaries, we can design our classes and the methods that will interact with its attributes/data. The chosen tool for this task is the UML class diagram. First, we focus on a simplified version including attributes only; we could leave the methods for a later time, to be added later, to design our project from general to more specific levels of detail.

Figure 3 below show inheritance and aggregation. Inheritance makes sense here, since there are common attributes between teachers and students. Aggregation is used to illustrate that each subject has a list of students and a teacher (lecturer).

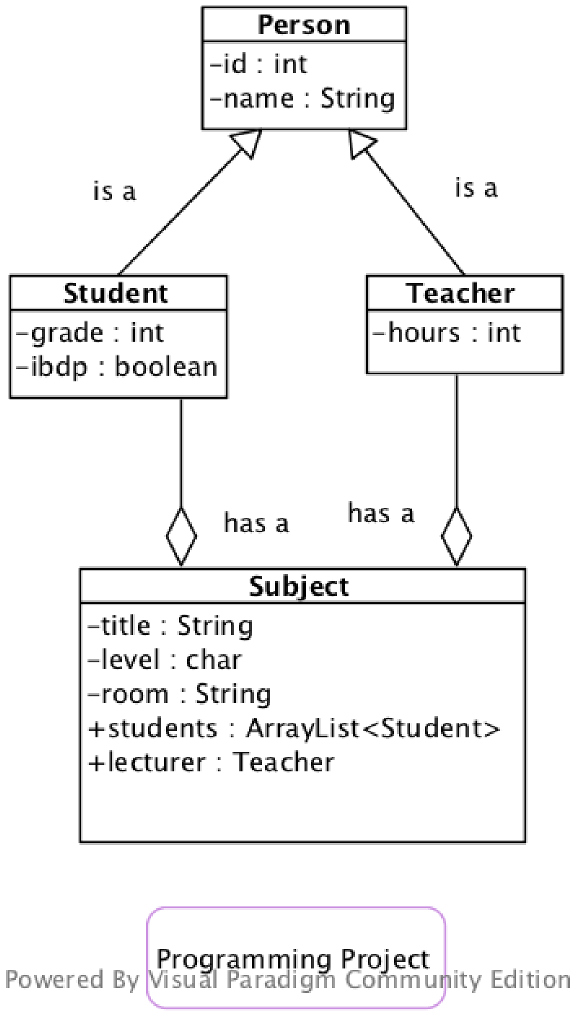


Figure 3. UML class diagram for the project's essential classes.

Now there is enough information to write the *Person*, *Student* and *Teacher* classes. Remember they are related with inheritance. Finally, the *Subject* class contains a teacher object and an arraylist of student objects.

After coding the base classes to represent our data, we have a more complete picture, with methods, as shown in figure 4 below:

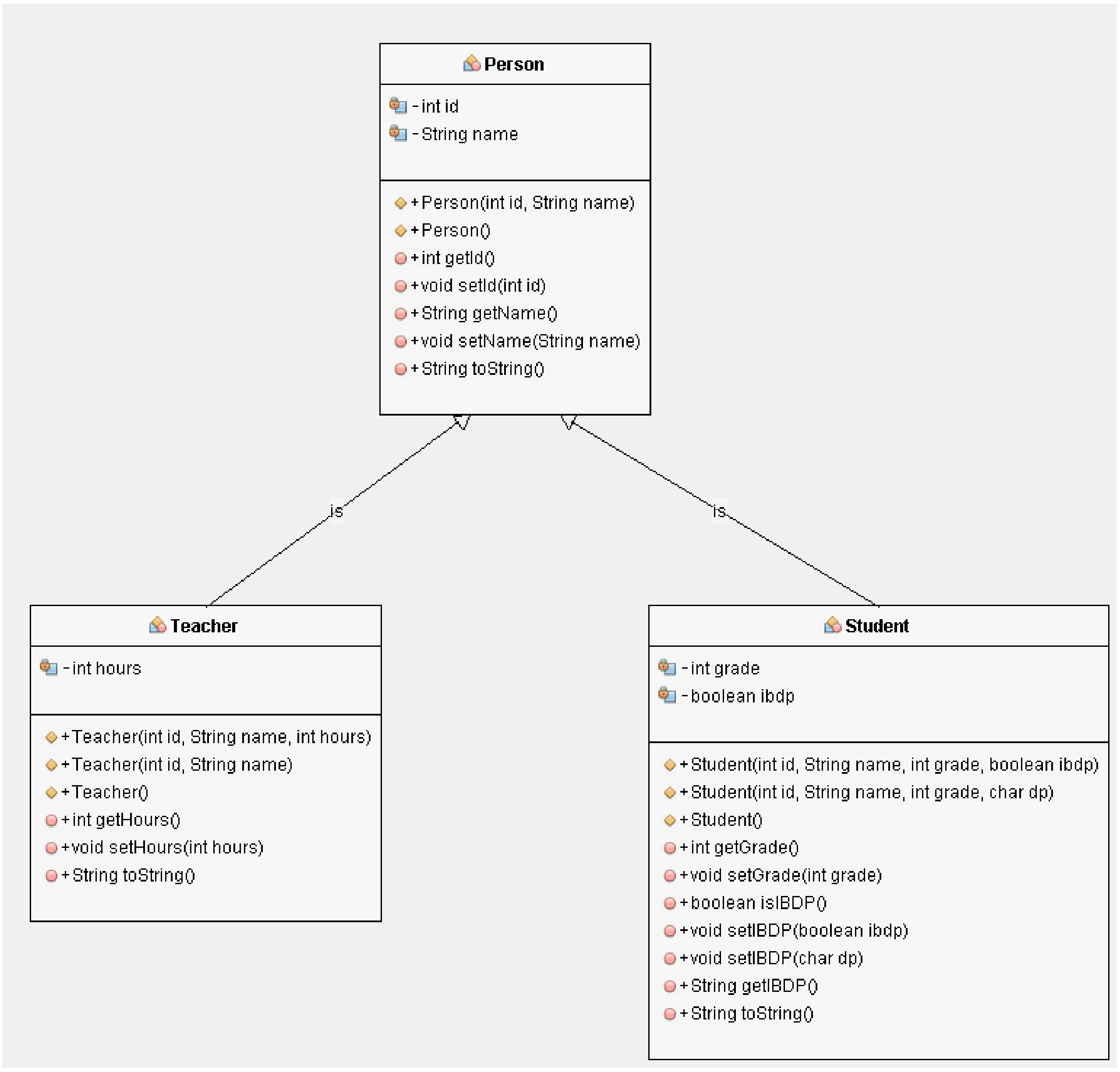
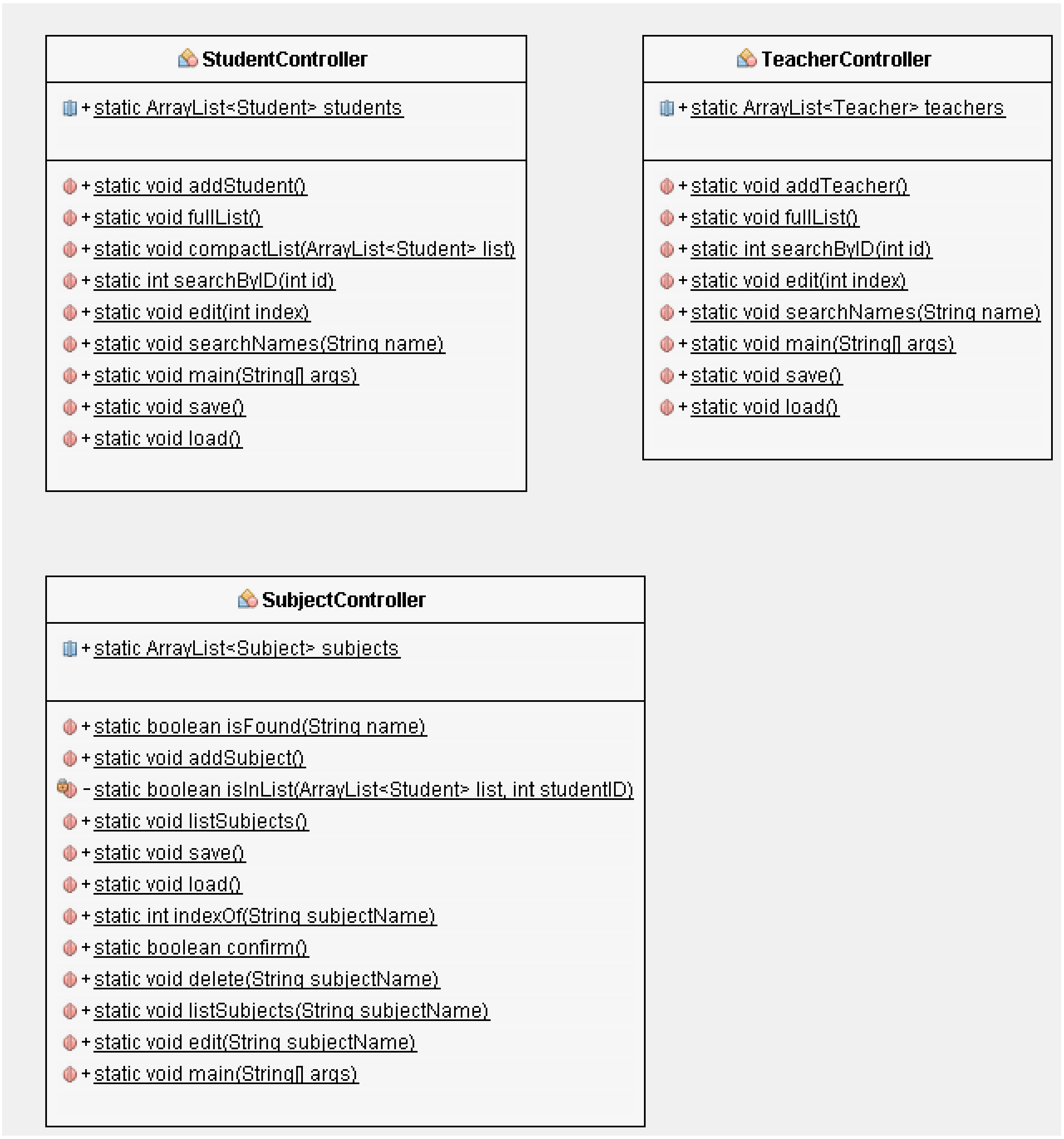

Person
- int id
- String name
+ Person(int id, String name)
+ Person()
+ int getId()
+ void setId(int id)
+ String getName()
+ void setName(String name)
+ String toString()
is
is
Teacher
- int hours
+ Teacher(int id, String name, int hours)
+ Teacher(int id, String name)
+ Teacher()
+ int getHours()
+ void setHours(int hours)
+ String toString()
Student
- int grade
- boolean ibdp
+ Student(int id, String name, int grade, boolean ibdp)
+ Student(int id, String name, int grade, char dp)
+ Student()
+ int getGrade()
+ void setGrade(int grade)
+ boolean isIBDP()
+ void setIBDP(boolean ibdp)
+ void setIBDP(char dp)
+ String getIBDP()
+ String toString()

Figure 4. Full UML class diagram for the project's base classes.

After coding these classes, we move on to the controller/manager classes, illustrated in figure 4 (next page). These classes create and manipulate objects from the base classes depicted in figure 3. The classes on figure 4 use the base classes in order to meet most of the success criteria for our solution (add, list, search, edit, etc.)



StudentController

```
+ static ArrayList<Student> students

+ static void addStudent()
+ static void fullList()
+ static void compactList(ArrayList<Student> list)
+ static int searchByID(int id)
+ static void edit(int index)
+ static void searchNames(String name)
+ static void main(String[] args)
+ static void save()
+ static void load()
```

TeacherController

```
+ static ArrayList<Teacher> teachers

+ static void addTeacher()
+ static void fullList()
+ static int searchByID(int id)
+ static void edit(int index)
+ static void searchNames(String name)
+ static void main(String[] args)
+ static void save()
+ static void load()
```

SubjectController

```
+ static ArrayList<Subject> subjects

+ static boolean isFound(String name)
+ static void addSubject()
- static boolean isInList(ArrayList<Student> list, int studentID)
+ static void listSubjects()
+ static void save()
+ static void load()
+ static int indexOf(String subjectName)
+ static boolean confirm()
+ static void delete(String subjectName)
+ static void listSubjects(String subjectName)
+ static void edit(String subjectName)
+ static void main(String[] args)
```

Analyse the code and the documentation. Hopefully it will help you develop your own product for the IB Computer Science IA.