INTPROG TB2: OBJECT ORIENTED PROGRAMMING

Practical Worksheet 6: Using ArrayLists

1. Introduction

This week, the focus is on using ArrayLists within Java. The ArrayList class is contained within the java.util package. I would recommend that you keep the ArrayList documentation open as you work – it will help you to know what methods are available to you. The documentation is here: https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html.

During the example, you won't be given all the code that you will need for it to be considered complete. By now, you should be able to write accessor and mutator methods yourself, so you will only be told which ones are required, and not their implementation. You will need to complete these for yourself.

As usual, work through this practical session at your own speed. Make sure you understand everything included within the example before you move onto the questions at the end of the worksheet. If you have questions, either ask your practical tutor, or ask a question using the anonymous question form on Moodle.

2. Example – Lab Classes

In this practical session, we will complete the lab classes that was started during week 4. Make sure that you have completed the example which is included in week 4 and the first question. You will need to transfer the contents of your BlueJ project into NetBeans before you can continue with this example. I would also test that it is functioning as anticipated before you add to the files. It is from this point that we will start with this example.

We are going to create a separate class called "LabClass". The lab class will have the following structure:

Lab Class	
unit: String instructor: String room: String timetabledSession: String	maxStudents: int groupNumber: int studentList: ArrayList<Student>
getUnit() getInstructor() setInstructor() getRoom() setRoom() getTimetabledSession() setTimetabledSession() getMaximumStudents()	setMaximumStudents() getGroupNumber() addStudent() getNumberOfEnrolledStudents() removeStudentByIndex() removeStudentByStudentName() findStudentByName() printLabClassInformation()

Create the class, and define the instance variables for the class. Don't forget that they all need to be defined as private variables.

We can then write the constructor for the LabClass. It will be as follows:

```
/**
 * Constructor to create the lab class
 * @param cUnit the unit the practical class is part of
 * @param cInstructor the instructor for the practical class
 * @param cRoom the room the practical class is in
 * @param cTimetabledSession the time and day of week the class is held on
 * @param cMaxStudents the maximum number of students which can be in the class
 * @param cGroupNo the group number for the practical class
 */
public LabClass(String cUnit, String cInstructor, String cRoom, String cTimetabledSession,
int cMaxStudents, int cGroupNo)
{
    unit = cUnit;
    instructor = cInstructor;
    room = cRoom;
    timetabledSession = cTimetabledSession;
    maxStudents = cMaxStudents;
    groupNumber = cGroupNo;

    studentList = new ArrayList<Student>();
}
```

Don't forget to import java.util to give you access to the ArrayList class.

You will need to write the following accessor and mutator methods:

- getUnit()
- getInstructor()
- setInstructor()
- getRoom()
- setRoom()
- getTimetabledSession()
- setTimetabledSession()
- getMaximumStudents()
- setMaximumStudents()
- getGroupNumber()

You will notice that the unit and group number instance variables have no mutator methods. This is because they will not change during the runtime of the program – once the associated unit and group number have been set, they will not change.

We are going to start with the addStudent() method. This will be used to add students to our class list. We are going to use the Student class which was created in practical 4 to model our students throughout this practical.

Add the following method after your getGroupNumber() method:

```
/**
* A method to add a student to the student list.  If the maximum number of
* students has been reached, the student will not be added to the list.
* An output in the terminal will display whether the student has been added
* to the list or not.
* @param newStudent the student to add to the class list
*/
public void addStudent(Student newStudent)
{
    if (studentList.size() == maxStudents)
    {
        // the maximum number of students has been reached
        // the student will not be added
        System.out.println("Sorry, the class is already full");
    }
    else
    {
        // the maximum number of students has not been reached
        // the student will be added
        studentList.add(newStudent);
        System.out.println("The student has been added");
    }
}
```

This method first checks to see whether the maximum number of students has been reached. If it has, the student cannot be added and the user will be warned. Otherwise, it will add the student, and tell the user that the student has been added to the lab class.

The next method we need to create will return the number of students which have been enrolled on the lab class. This is will count the number of students who have been added to the studentList ArrayList. To achieve this, we are going to use the size() method which is part of the ArrayList class.

```
/**
* Method to get the number of students who are currently enrolled on the
* unit.
* @return the number of students who have been enrolled
*/
public int getNumberOfEnrolledStudents()
{
    return studentList.size();
}
```

The next method will use the index of a student to remove them from the list. This will use a method which is contained within the ArrayList class, which will use the index number to remove a student from the list.

```
/**
* A method to remove a student using their location in the ArrayList
* @param index
*/
public void removeStudentByIndex(int index)
{
    studentList.remove(index);
}
```

If we know the index number for the student, the above method will be useful. However, we may not always know which position the student is held in. Therefore, we can write a method which will remove a student just by using their name. As we want to remove an item from the list whilst using a loop, we need to use an iterator, and its remove() method.

```
/**
 * Removes a student from the class list using their name rather than the
 * index number.
 * @param studentToRemove the name of the student to remove
 */
public void removeStudentByName(String studentToRemove)
{
    Iterator<Student> it = studentList.iterator();
    Student currentStudent;
    String currentStudentName;
    boolean found = false;

    while(it.hasNext())
    {
        currentStudent = it.next();
        currentStudentName = currentStudent.getName();

        if(currentStudentName.equals(studentToRemove))
        {
            it.remove();
            System.out.println(studentToRemove + " has been removed");
            found = true;
        }
    }

    if (!found)
    {
        // student hasn't been found in the list, so warn the user
        System.out.println("Student is not in the class list");
    }

}
```

The next thing we want to be able to do is print the information held about a particular student within the class list. The below method will search through the ArrayList until it finds the required student, then print the information. If the student is not included within the class list, the user will receive an error message stating that they have not been found.

```
/**
 * A method to print the information about a student when only given their
 * name.  If the student is not part of the lab class, the user will be told.
 * @param searchName the name of the student to print the information about
 */
public void findStudentByName(String searchName)
{
    boolean found = false;
    int index = 0;
    String studentName;
    Student currentStudent;

    // this while loop will run until the student is found, or it reaches
    // the end of the studentList
    while (index < studentList.size() && !found)
    {
        // get the student at the current position in the list (based on
        // the value of index
        currentStudent = studentList.get(index);
        // get the student name, from the current student
        studentName = currentStudent.getName();

        if(studentName.equals(searchName))
        {
            // the name to be found matches the name of the current student
            found = true;
            currentStudent.printInformation();
        }
        else
        {
            // the name to be found doesn't match the name of the current student
            index++;  // increment the index by 1
        }
    }

    if(!found)
    {
        // the student hasn't been found
        System.out.println("Student not found");
    }
}
```

The final method which is required for this class is the printLabClassInformation(). This will print all the information held about the lab class, and print the names of all the students enrolled on practical session.

```
/**
 * A method to print all the information relating to the practical class,
 * including the student list.  The student list consists of name only.
 */
public void printLabClassInformation()
{
    System.out.println("Unit: " + unit);
    System.out.println("Instructor: " + instructor);
    System.out.println("Room: " + room);
    System.out.println("Timetabled Session: " + timetabledSession);
    System.out.println("Group Number: " + groupNumber);
    System.out.println("Maximum Number of Students: " + maxStudents);
    System.out.println("Number of Enrolled Students: " + studentList.size());

    System.out.println("Student List");

    for (Student student : studentList)
    {
        System.out.println(student.getName());
    }

}
```

In the main method, delete the previous testing commands, and start by creating a ClassList object. Then, you will need to write tests to ensure that the accessor and mutator methods you wrote are working correctly.

```
package labclasses;

public class LabClasses
{
    public static void main(String[] args)
    {

        LabClass intprog = new LabClass("INTPROG", "Claire", "LG0.14", "Mon 2-4pm", 4, 1);
        // by choosing a low maximum number of students we can quickly test
        //that the add function works as intended

        /** tests for the accessor and mutator methods you wrote */

    }

}
```

Enter the below underneath your testing of the accessor and mutator methods. This will test to ensure that all the methods we have written are functioning as expected.

```
intprog.addStudent(new Student("Alice", "123456"));
intprog.printLabClassInformation(); // this will test whether the
                                    // student has been added, the
                                    // printLabClassInformation() method
                                    // and the getNumberOfEnrolledStudents()
                                    // method

intprog.addStudent(new Student("Bob", "234567"));
intprog.addStudent(new Student("Charlie", "345678"));
intprog.addStudent(new Student("David", "456789"));
intprog.addStudent(new Student("Eve", "567890")); // this should produce an error

intprog.printLabClassInformation();

intprog.findStudentByName("Charlie");
intprog.removeStudentByIndex(1);
intprog.printLabClassInformation();
intprog.removeStudentByName("David");
intprog.findStudentByName("David"); // this should produce an error
intprog.removeStudentByName("David"); // this should produce an error
```

3. Practice Questions

The following questions are designed to give you some practice at using ArrayLists within Java. If you have any questions, please make sure you ask the members of staff within the practical sessions.

1. On Moodle, you will find the MusicOrganiser starter project. Download and unzip the project. It contains 3 classes which have been started. You need to complete all the methods contained within the project. Make sure you read the documentation comments above the methods – they will help you understand what is required.
   Note: You only need to change the MusicManager class, you should not need to touch the other files.

2. You will be creating a StockManagementSystem project. This will contain the following 2 classes:

Product
productID: int name: String quantity: int
getID() getName() getQuantity() printInfo() increaseQuantity() sellItem()

StockManager
stock: ArrayList<Product>
addProduct() findProduct() delivery() sellProduct() numberInStock() printProductDetails()

a) Write the Product class using the UML diagram above. Make sure that the increaseQuantity() and sell item methods have conditionals, to ensure that invalid operations cannot be carried out. The printInfo() method should print all the information about the item on a single line.

b) Create the StockManager class, declare the instance variables, and define the constructor.

c) Write the addProduct() method. This will add a product to the stock ArrayList.

d) Write the findProduct() method. This will take the product ID as a parameter and return the associated Product when found. If the product is not found, it returns null.

e) Write the delivery() and sellProduct() methods. Both these methods will take the product ID and quantity as parameters. The findProduct() method should be used to find the item based on the ID number. The stock quantity for that item should then be increased or reduced by the quantity in the parameter.

f) Write the numberInStock() method. This will take the product ID as a parameter, then print the quantity of that item which is in stock.

g) Write the printProductDetails() method. This will print the product information for everything which is in the stock ArrayList.

h) Write a main method which can be used to test the functionality of the StockManagementSystem project.

3. On Moodle, you will find the MailSystem project. This contains a MailClient class. You will need to add to this to complete the project. The project has the following classes:

MailItem
to: String from: String subject: String urgent: boolean message: String
getTo() getFrom() getSubject() getUrgent() getMessage() printMail()

MailServer
items: ArrayList<MailItem>
howManyMailItems() getNextMailItem() postMail()

MailClient
server: MailServer user: String
sendMailItem() printNextMailItem()

a) Write the MailItem class, using the above UML diagram

b) Write the MailServer class, using the above UML diagram. The howManyItems() method will count how many mail items there are in the ArrayList for a particular user (the user name should be entered as a parameter for the method). The getNextMailItem() method will return the next mail item for a particular user (which again is entered as a parameter). Before it returns the next mail item it must remove it from the ArrayList. When the method gets to the end of the ArrayList without finding any mail for the user it will need to return null.
Hint: use an iterator / while loop combination for this exercise. As soon as the mail item for the requested user has been found remove the item from the iterator, then return the item.

c) Create a main method file which can be used to test your classes. Initially you will need to create a MailServer, then 2 MailClients. You should then be able to send mail between the clients, and read them too.

4. Create a new project called "ChildrensToys". Within this project create the following two classes:

Toy
name: String owner: String type: String colour: String
getName() getOwner() setOwner() getType() getColour() getToyInformation()

ToyCollection
toys: ArrayList<Toy>
addToy() removeToy() removeToyByName() changeToyOwner() getNumberOfToysByOwner() listToysByOwner() listAllToys()

The getToyInformation() should return a string which has all the information on the toy on a single line, in the following format:

```
Teddy is a Brown Bear which is owned by Alice
```

The getNumberOfToysByOwner() should return an integer, showing the number of toys owned by the person entered in the parameters. It should return 0 if there are no toys for that owner.

The listToysByOwner() method should output information in the following format:

```
Bob owns 2 toys:
Ken
Barbie
```

or

```
Eve does not have any toys
```

The listAllToys() should output the information in the following format:

```
There are 5 toys in the collection
Teddy is a Brown Bear which is owned by Alice
Pudsey is a Yellow Bear which is owned by Bob
Lightning McQueen is a Red Car which is owned by Alice
Ken is a Pink Action Man which is owned by Bob
Barbie is a Pink Barbie which is owned by Bob
```

Make sure that you create a main method which can be used to test your ToyCollection class functionality.