COM1027 Programming Fundamentals

Lab 10

Object-oriented Programming Principles

Purpose

The purpose of this lab is to develop your skills and ability to:

- Convert a UML class diagram into Java code
- Define concrete classes and complex objects
- Define an implementation class based on an Interface
- Override methods.

Lab Structure

Labs are a mixture of step by step instructions that enable you to learn new skills, and exercises so that you can define your own examples.

We will use **Concept Reminders** in front of text when we are reminding you of concepts you have already learnt previously in labs or lectures.

In using this document it is possible to paste text directly into Eclipse. That means when defining pieces of code contained in lab sheets in Eclipse their value can be copied directly from here, which will remove the possibility of mistyping values. If you are reading a PDF version of the document, then clicking on the 'Select' icon activates text selection. Before running the code, ensure that the code has been copied correctly, and make the relevant changes to make the code fully-functional.

These lab exercises will be **assessed**. They are marked, and contribute to your final grade. This lab exercise has 10 marks to earn which will contribute towards your final grade. Every place you have to add code is indicated as an 'Exercise' with instructions.

A total of 10 labs will be assessed, that will correspond to the first 20% of your final grade. Please submit your completed work using GitLab before 4pm on Friday Xth Month 2021.

You must comply with the University's academic misconduct procedures: `https://www.surrey.ac.uk/office-student-complaints-appeals-and-regulation/academic-misconduct`

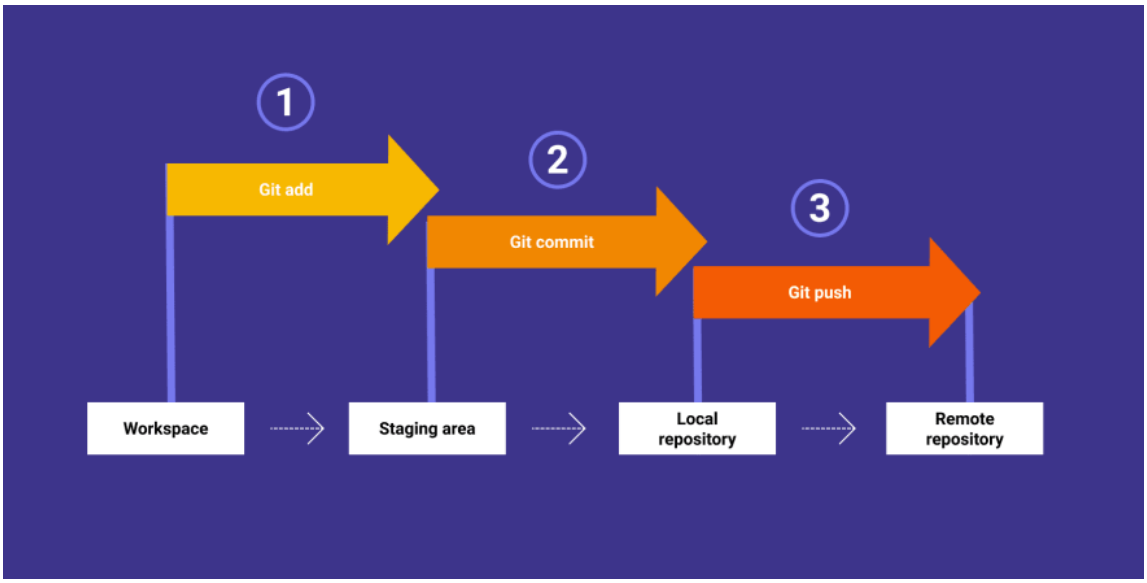
Please ask us lots of questions during the labs, or use the discussion forums on SurreyLearn. All the demonstrators are here to help you get a better understanding of the Java programming concepts.

Instructions

For this module, we will be using automated assessment tools for all lab activities and coursework. To achieve this, we will be using the Faculty of Engineering and Physical Sciences (FEPS) GitLab platform to upload completed code.

In last week's lab activity, you completed the following:

- Downloaded a project from SurreyLearn and imported it into your workspace
- Made changes to the project by creating new classes
- Used the UML diagrams to convert the structured English language to Java code
- Commited the changes to your local repository and
- Pushed those changes onto your remote repository on GitLab



Screenshot taken from `https://docs.gitlab.com/`

In this lab, you will download a new project from SurreyLearn, make substantial changes to the project, commit and push those changes to your remote repository, just like in previous weeks.

Important Note: Do not push any buggy code to the repository unless you have to. Use the `'Run As > Java Application'` or `'Run As > JUnit Test'` as indicated in the instructions below.

Reminder: You can access your GitLab repository via `https://gitlab.eps.surrey.ac.uk/`, using your username and password.

Example

In this example, you will be expected to watch one of the pre-recorded coding demonstrations in order to support the implementation of the exercises.

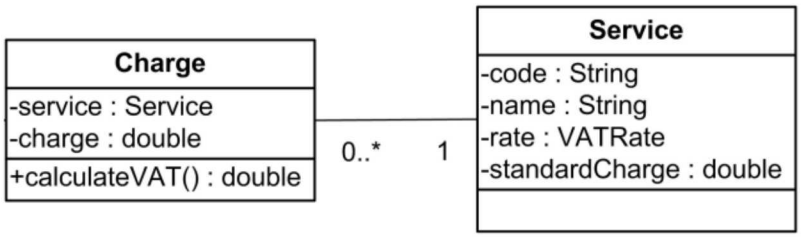
1. Import an existing Eclipse project and set your workspace
From SurreyLearn, download the project called `COM1027_Lab10.zip`. Extract the contents of the project in the Downloads folder. Copy the uncompressed (unzipped) `Lab010` folder to your local repository. **Note:** Do not copy the `COM1027_Lab010` folder. Navigate in the folder, and only copy `Lab010` folder.

If you are using the computer labs, then your local git directory would look like this:
`/user/HS223/[username]/git/com1027[username]/`
For example:
`/user/HS223/sk0041/git/com1027sk0041/`
where sk0041 shows a sample username.

Start Eclipse and open the same lab workspace as last time. Import the project in Eclipse, by clicking on `File > Import > General > Project from Folder or Archive`. Select `Directory` to locate `Lab010` from your local git repository. Once the files are loaded, select the Eclipse project and click `Finish`.

Exercise 1 (1 Marks)

This exercise requires you to convert the following UML class diagram into Java code:



The diagram shows an example of an association between the `Charge` and `Service` classes. A *Charge* links a *Guest (which will be introduced and defined in the next exercise)* to a *Service* and records the charge incurred by the guest for the service. The charge is normally the same as the standard charge defined for the service.

A *Service* is defined by its code, name, VAT rate and standard charge. The code is a string which consists of a four letter service type (for example, "ROOM") followed by a 4 digit number. Validation of this is required. This can be added in the constructor or in a separate method and then called in the constructor.

For this exercise, you will be using the `lab10_exercise1` package and you are expected to demonstrate your understanding of constructors, getters and complex objects.

- Define a `VATRate`. Here, a VATRate is dened as an enumeration with the values `ZERO`, `LOW` and `STANDARD`. User-defined methods may need to be introduced in the enumerated type to support the implementation of the the `Service` and/or `Charge` classes.
- Define the `Service` class in Java. Include in your class:
  - The required fields using an appropriate visibility, type and name.
  - A parameterised constructor for the class, with the required visibility, signature, name and body.
  - Appropriate getters, each with the required visibility, signature, name and body.
- Define the `Charge` class in Java. Include in your class:
  - The required fields using an appropriate visibility, type, name and comment.
  - A parameterised constructor for the class, with the required visibility, signature, name, body and comments.
  - Appropriate getters, each with the required visibility, signature, name, body and comments.
  - The `calculateVAT` method, with the required visibility, signature, name, body and comments.

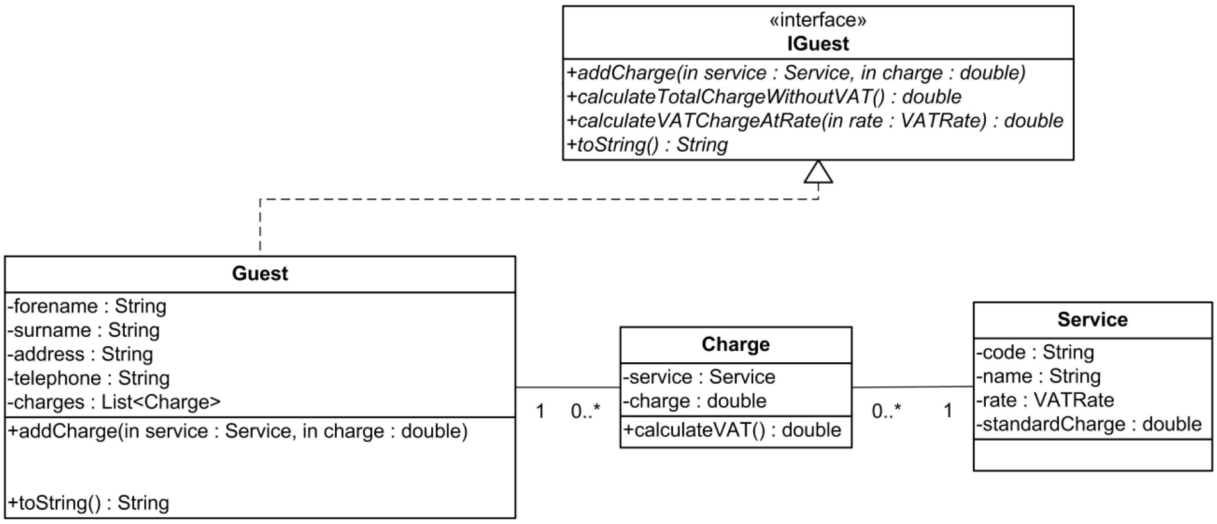
  VAT Charge at a Specified VAT Rate - The VAT to be charged is calculated as the sum of all VAT percentages for all the charges that the guest has incurred at the specified VAT rate. VAT is charged on items at different rates as set down in VAT law. Most items are charged at the standard rate of 17.5%; some items are charged at the lower rate of 5%; while some items are zero-rated and do not incur VAT.

To test the functionality of the code, right-click on the project name and select `Run As` > `JUnit Test`.

Exercise 2 (3 Marks)

This exercise requires you to create two units of code using your understanding of interfaces, their methods and sub-classes. This exercise also requires you to add comments in your code.

For this exercise, you will be using the `lab10_exercise2` package. Copy the `Charge` and `Service` classes from the previous exercise, as well as the `VATRate` enumerated type. Then create the required classes as follows:



An *IGuest* is an interface specifying the required behaviour of a *Guest* object. A string representation of the *IGuest* can be obtained from the *toString* method. This will return the guest's forename, surname, address and telephone number as a string, for example "Sid James, 12 North Lane, Guildford, Surrey, 01483 654321". To indicate that a guest has incurred a charge, a *Service* and its associated charge may added to the guest's record.

A *Guest* is an implementation of an *IGuest*. This implementation has an forename, surname, address and telephone number. Each *Guest* also has a *Charge* for each service they have taken. No validation is performed on the forename, surname, address or telephone number.

- Define the *IGuest* interface in Java. Include in your interface:
  - The interface methods, each with the required visibility, signature and name.
- Define the *Guest* class in Java so that it implements the *IGuest* interface. Include in your class:
  - The required fields using an appropriate visibility, type, name and comment.
  - A parameterised constructor for the class, with the required visibility, signature, name, body and comments.
  - Appropriate getters, each with the required visibility, signature, name, body and comments. You can check the JUnit files to identify which getters are required.
  - The *addCharge* method, with the required visibility, signature, name, body and comments.
  - The *calculateTotalChargeWithoutVAT* method, with the required visibility, signature, name, but without an implementation. The method should return by default zero (0).

To achieve this, add the following for the *calculateTotalChargeWithoutVAT* method. The functionality for this method will be added in Exercise 3.

```
@Override
public double calculateTotalChargeWithoutVAT() {
      return 0;
}
```

– The *calculateVATChargeAtRate* method, with the required visibility, signature, name, but without an implementation. The method should return by default zero (0). This is similar with the one above.

```
@Override
public double calculateVATChargeAtRate(VATRate rate) {
      return 0;
}
```

– The *toString* method, with the required visibility, signature, name, body and comments.

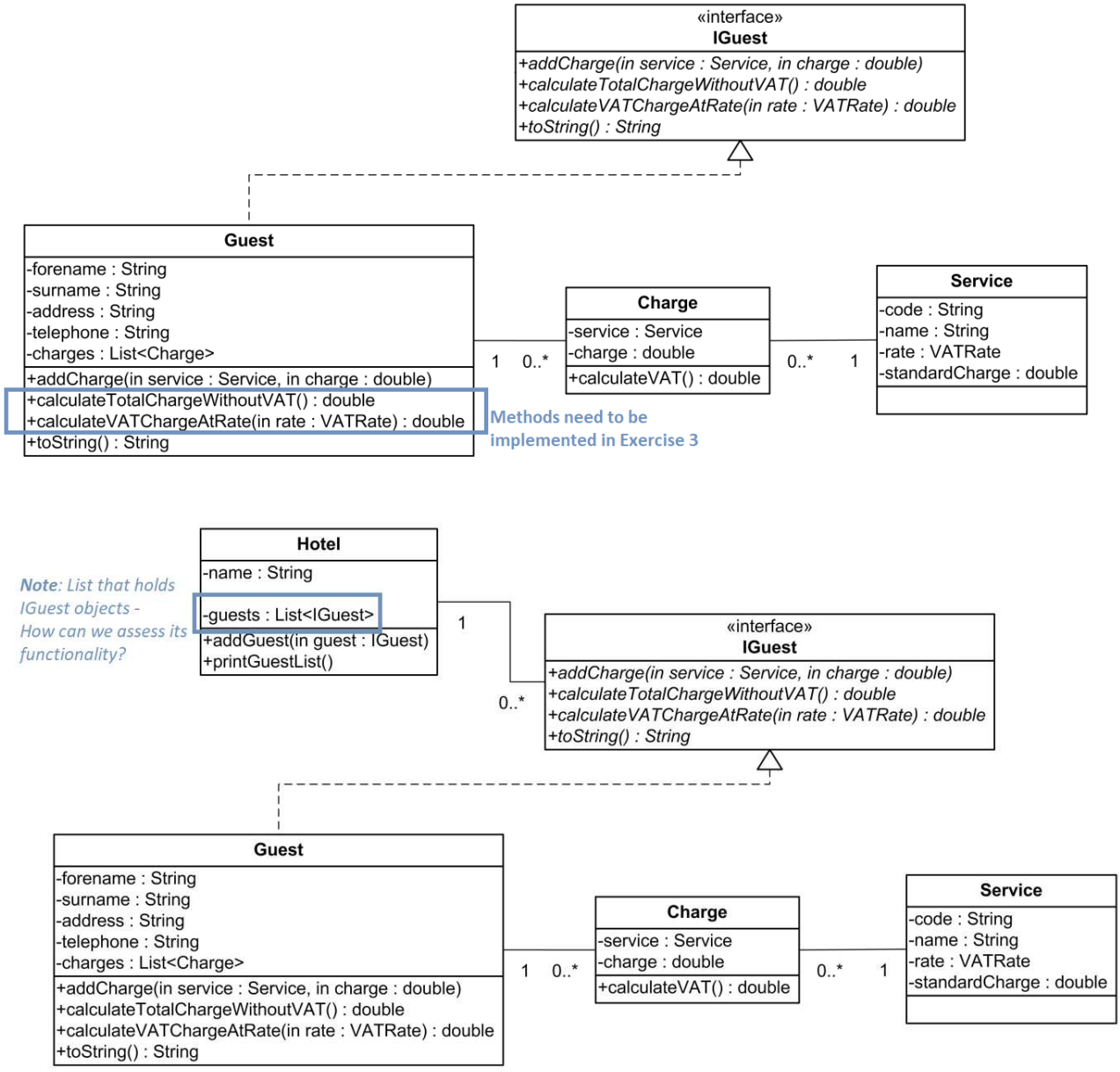
Before moving to Exercise 3, test your code by right-clicking the project name and then `Run As` > `JUnit Test`. Once all the tests for the `TV` class pass, you can start working on Exercise 3.

Go to next page

Stretch & Challenge Exercise (6 Marks)

This exercise requires you to use the classes defined in exercises 1 and 2 (as well as the pre-defined `Hotel` class), and complete the functionality of the hotel charge system.

For this exercise, you will be using the `lab10_exercise3` package. Copy all the relevant classes from `lab10_exercise2` and `lab10_exercise1` to the `lab10_exercise3` package.



For an *IGuest*, the following can be calculated:

- **Total Charge without VAT -** The total charge for the guest without VAT is calculated as the total total charge for each service the guest has used.
- **VAT Charge at a Specified VAT Rate -** The VAT to be charged is calculated as the sum of all VAT percentages for all the charges that the guest has incurred at the specified VAT rate. VAT is charged on items at different rates as set down in VAT law. Most items are charged at the standard rate of 17.5%; some items are charged at the lower rate of 5%; while some items are zero-rated and do not incur VAT. For example, if a guest has incurred two charges at the standard rate of VAT, with a value of £60 for the first charge, and £90 for the second charge, then the standard rate VAT is:

$$\begin{aligned} \text{Charge 1} &= £60 \\ \text{Charge 2} &= £90 \\ \text{Total Charge without VAT} &= £150 \end{aligned}$$

$$\begin{aligned} \text{VAT at the Standard Rate} &= \frac{£60 \times 17.5}{100} + \frac{£90 \times 17.5}{100} \\ &= £10.50 + £15.75 \\ &= £26.25 \end{aligned}$$

  Here, a *VATRate* is defined as an enumeration with the values "ZERO", "LOW" and "STANDARD".

- Inspect the `Hotel` class that has been defined for you. Some of the lines of code, may not function until the complete behaviour of the `Guest` class is defined.
- Complete the functionality of the *Guest* class in Java so that it implements the *IGuest* interface. Include in your class:
  - The *calculateTotalChargeWithoutVAT* method, with the required visibility, signature, name, and body. The method should return the appropriate value as per the notes above.
  - The *calculateVATChargeAtRate* method, with the required visibility, signature, name, and body. The method should return the appropriate value as per the notes above.

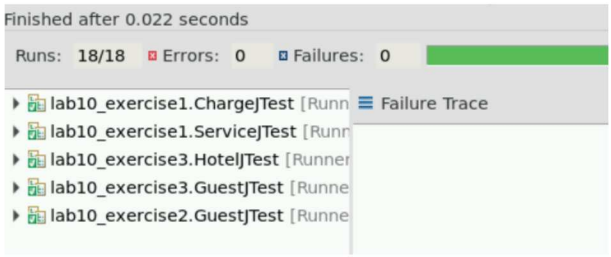
You can test your code, by creating a `Test` class with multiple objects or run the predefined JUnit tests. Comments are also added in the JUnit methods to further help you understand how the methods should function.

Final Steps

The final step of the lab activity is to check on whether the code is fully functional. To do this, right-click on the project name and select `Run As > JUnit Test`.

If any errors occur, select the relevant file and check the error.

By selecting `Run As > JUnit Test` all the pre-defined JUnit tests will be called. You should get the following in order to proceed:



It is now time to test the structure of your Maven project. To do this, right-click on the project name and select `Run As > Maven Test`.

This should return the following message on the console:

```
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Ti

Results :

Tests run: 18, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------
[INFO] Total time: 4.977 s
```

The test passes - now what?
You can now commit and push all the changes made to the remote repository. To do this, select the `Git Staging` view via the menu `Window > Show View > Other...` and then `Git > Git Staging`. Add all the unstaged changes to your local repository, and commit and push the changes to the remote repository.