COM1027 Programming Fundamentals

Lab 111

Revision: Object-oriented Programming Principles

Purpose

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

- Revise and put into practice some of the concepts that have been covered as part of this module
- 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`

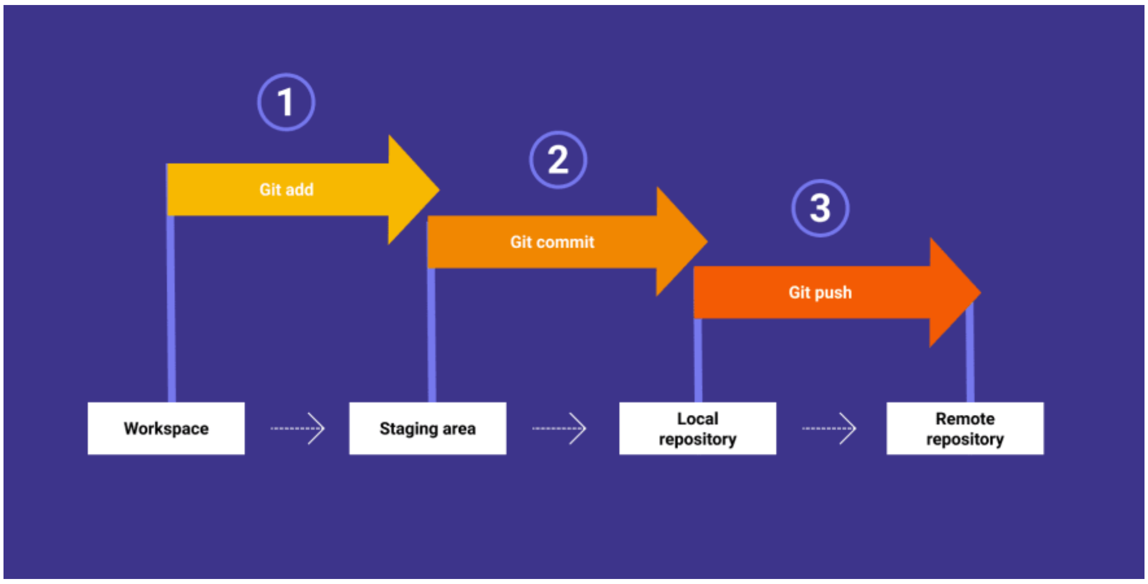
Please ask us lots of questions during the labs, or use the discussion forums on SurreyLearn. All the demonstrators are here to help you get a better understanding of the Java programming concepts.

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_Lab11.zip`. Extract the contents of the project in the Downloads folder. Copy the uncompressed (unzipped) `Lab11` folder to your local repository. **Note:** Do not copy the `COM1027_Lab11` folder. Navigate in the folder, and only copy `Lab11` 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 `Lab11` from your local git repository. Once the files are loaded, select the Eclipse project and click `Finish`.

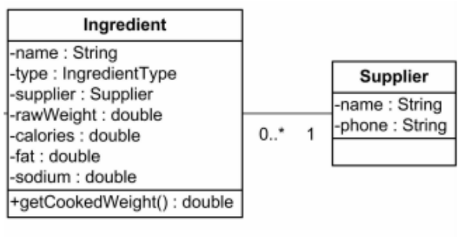
2. Coding Demonstration
Once imported, set the Eclipse project to the side and visit SurreyLearn. Go to the following links on SurreyLearn and make sure that you have a good understanding of what a UML class diagram represents and how it can be converted into Java code.

- Visit https://surreylearn.surrey.ac.uk/d2l/le/lessons/227067/topics/2302728 (Week 10 material)
- Make notes on what a UML class diagram represents (i.e. classes in the OOP system, fields, methods and relationships)
- Visit https://surreylearn.surrey.ac.uk/d2l/le/lessons/227067/topics/2302729 (Week 10 material)
- Make notes on how each unit of code can be represented in code, i.e. concrete classes, enumerated types, interfaces (super classes) and sub classes.

The `lab11_example` can be used for any experimentation and will not get assessed. This could include re-creating the same UML class diagrams or testing some of the concepts covered in those videos.

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 `Supplier` and `Ingredient` classes. Note: Two enumerated types have been provided in the exercise and can be used to support the implementation of those two classes.

A *Supplier* object stores the name and phone number for a particular supplier of an ingredient. No *Supplier* object can be created without a name or phone number. No specific validation is performed on the name, but the phone number must be a string consisting of a 0 followed by numbers (6 to 10 digits with no spaces). If the phone number is invalid a *Supplier* object cannot be created and an exception is thrown.

An *Ingredient* object represents an ingredient that has been added to an item. This consists of a name, type, supplier, raw (uncooked) weight (g), number of calories (kcal), amount of fat (g) and sodium (g). No *Ingredient* object can be created without a name, type, supplier, and amounts for the raw weight, calories, fat or sodium. No validation is performed on these fields.

The *IngredientType* is defined to be either VEGETABLE or MEAT.

Note: The *Ingredient* class provides a method to calculate the cooked weight of the ingredient. The cooked weight of any ingredient is simply assumed to be 80% of the raw weight (g).

For this exercise, you will be using the `lab11_exercise1` package and you are expected to demonstrate your understanding of constructors, getters, complex objects as well as the use of exceptions.

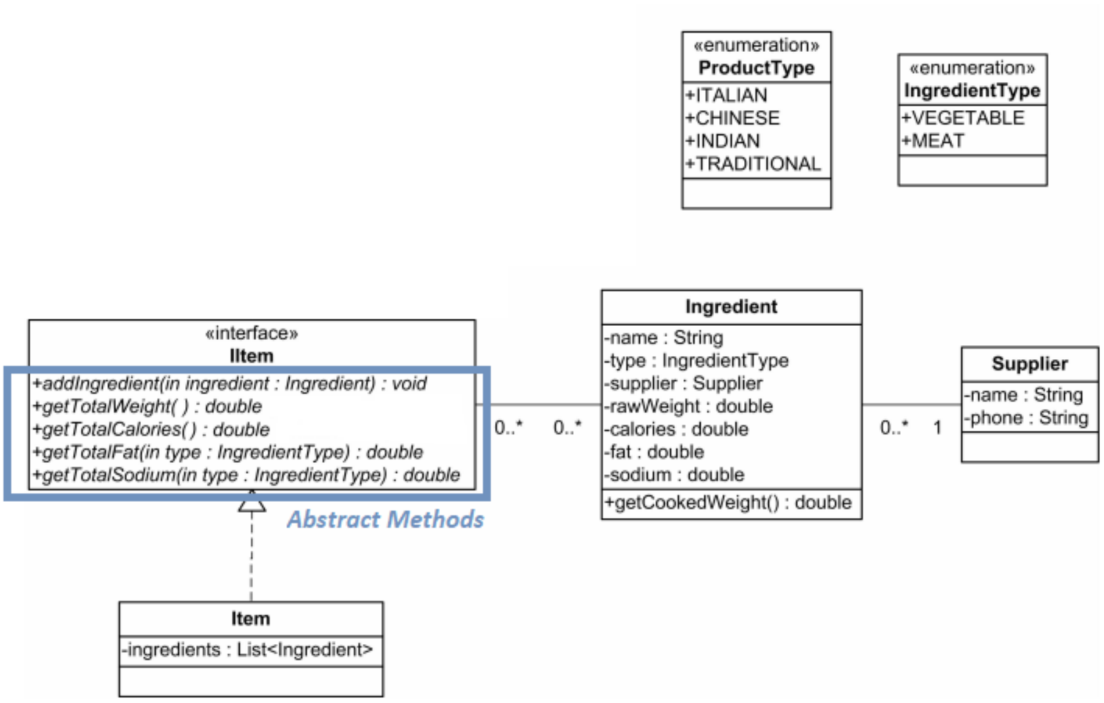
- Define the *Supplier* class in Java. Include in your class:
  - The required fields using an appropriate visibility, type and name.
  - A parameterised constructor for the class, with the required validation.
  - Appropriate getters (if needed)
- Define the *Ingredient* class in Java. Include in your class:
  - The required fields
  - A parameterised constructor for the class - Look at the JUnit test case. Are any exceptions required?
  - Appropriate getters and/or setters. `Note: Only define the accesssor and mutator methods that are required. Do not auto-generate all the getters and setters by default.`
  - The *getCookedWeight* method, with the required implementation.

To test the functionality of the code, and whether it meets the expectations of this exercise, 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.

For this exercise, you will be using the `lab11_exercise2` package. Copy the `Supplier` and `Ingredient` classes from the previous exercise, as well as the two enumerated types. Then create the required classes as follows:



An *IItem* is an interface which specifies all of the required methods of an *Item* object. Methods are defined to add an *Ingredient* object to the item, to get the total cooked weight (g), total calories (kcal), fat (g) and sodium (g) for either the meat or the vegetables in the item.

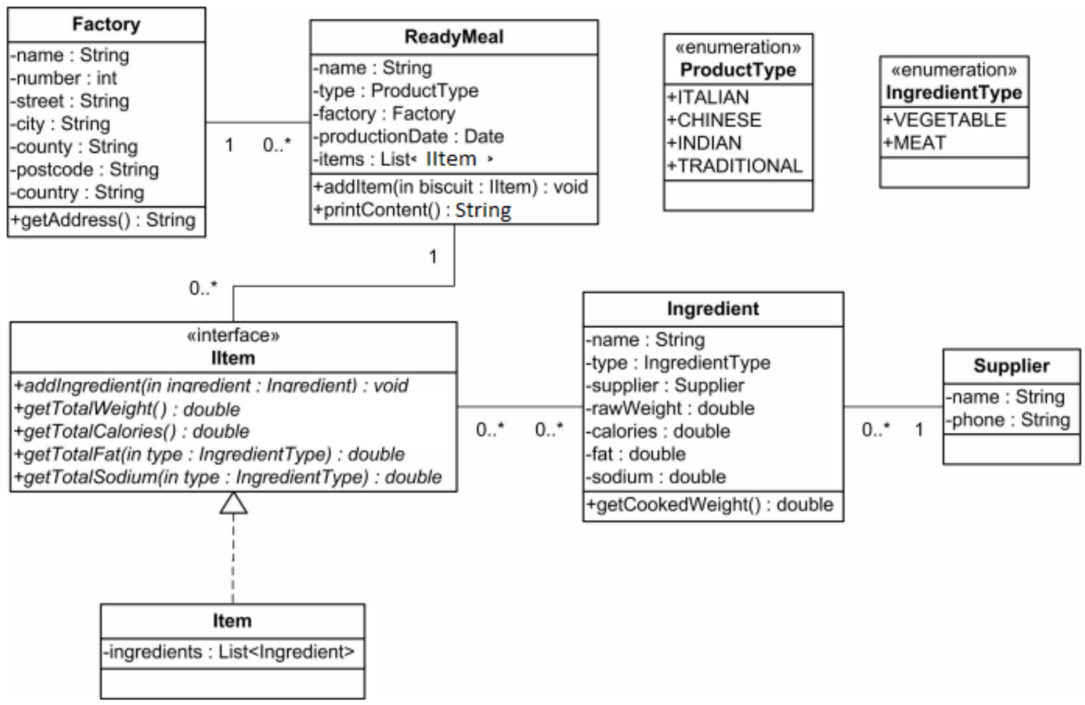
- Define the *IItem* interface in Java with the interface methods.
- Define the *Item* class in Java so that it implements the *IItem* interface. Include in your class:
  - The required fields
  - A constructor for the class, with the required visibility, signature, name, and body. Does this have to be a parameterised or default constructor?
  - The *addIngredient* method, with the required visibility, signature, name, body and comments.
  - The *getTotalCalories* method by adding the relevant calories and returning them
  - The *getTotalFat*, *getTotalSodium*, and *getTotalWeight* methods, with their required visibility, signature, name and body. Note that some of these have a parameter which needs to be checked! For example the getTotalFat method should return the total fat (g) in a numeric form for the selected ingredient type.

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

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 `Factory` class), and complete the functionality of the Ready Meal Production system.

For this exercise, you will be using the `lab11_exercise3` package. Copy all the relevant classes from `lab11_exercise2` to the `lab11_exercise3` package.



A *Factory* object stores the registered address at which ready meals are made. This includes the full address of the factory. No *Factory* object can be created without a name, number, street, city, county, postcode or country. No validation is performed on these fields. *Factory* provides a method *getAddress* which returns the formatted address as follows:

```
Findus
97 Horse Avenue
Oxford
Oxfordshire
OX1 1NA
UK
```

A *ReadyMeal* object represents a particular ready meal that has been made at the factory. This includes the name of the ready meal, the type of product, the factory at which it was made and the date of production. The object also contains a reference to all of the *IItem* objects which the ready meal contains. No *ReadyMeal* object can be created without a name, type or factory. No validation is performed on these fields.

The *ProductType* is defined to be either ITALIAN, CHINESE, INDIAN or TRADITIONAL.

Note: The *ReadyMeal* provides a method to add an *IItem* to the ready meal. When the first *IItem* object is added to the ready meal, the production date is set to the current date. A method is also provided to print the ready meal's contents (to the Java console), formatted as follows:

```
Italian Lasagne Ready Meal 150g

Contents
Calories: 565kcal
Meat (fat): 10.0g
Vegetable (sodium): 20.0g

Findus
97 Horse Avenue
Oxford
Oxfordshire
OX1 1NA
UK
```

The contents consists of the type of the ready meal ("Italian") followed by the name ("Lasagne"), then ("Ready Meal") and finally the total cooked weight of the ready meal ("150g"). For ITALIAN items, the description is "Italian". For CHINESE items the description is "Chinese". For INDIAN items the description is "Indian". For TRADITIONAL items the description is "Traditional". The contents for the ready meal are listed with the sum of the meal's calories, the total fat for meat ingredients and total sodium for vegetable *IItem* objects in the ready meal. The factory address is printed last on the contents.

- Inspect the `Factory` class that has been defined for you. Some of the lines of code, may not function until the complete behaviour of the `ReadyMeal` class is defined.
- Complete the functionality of the *ReadyMeal* class in Java. Include in your class:
  - The required fields as per the UML class diagram
  - The `productionDate` can be declared as a field using the following syntax `private Date productionDate = null;` and can then be defined within the constructor `this.productionDate = new Date();`
  - A parameterised constructor for the class
  - The *addItem* method, with the required visibility and body, and
  - The *printContent* method which should return a String value.

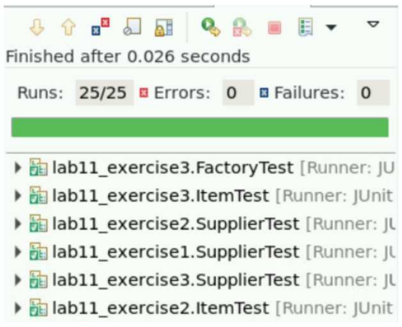
You can test your code, by creating a `Test` class with multiple objects or run the predefined JUnit tests.

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.

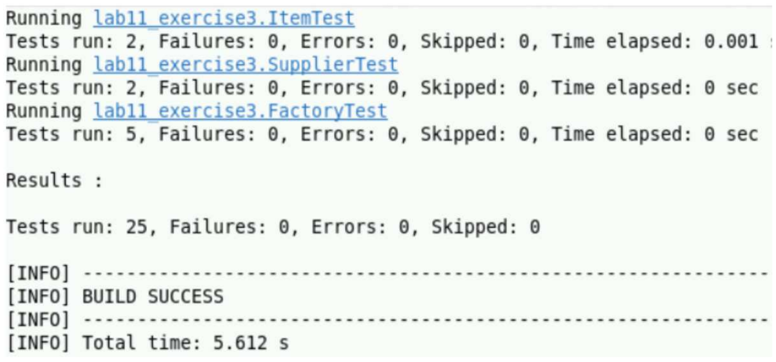
By selecting `Run As > JUnit Test` all the pre-defined JUnit tests will be called. You should get the following in order to proceed:



```
Finished after 0.026 seconds
Runs: 25/25  Errors: 0  Failures: 0
lab11_exercise3.FactoryTest [Runner: JU
lab11_exercise3.ItemTest [Runner: JUnit
lab11_exercise2.SupplierTest [Runner: JU
lab11_exercise1.SupplierTest [Runner: JU
lab11_exercise3.SupplierTest [Runner: JU
lab11_exercise2.ItemTest [Runner: JUnit
```

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:



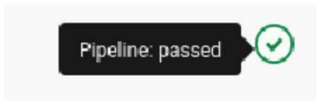
```
Running lab11_exercise3.ItemTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001
Running lab11_exercise3.SupplierTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec
Running lab11_exercise3.FactoryTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

Results :

Tests run: 25, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 5.612 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.


Pipeline: passed