# COM1003 Problem Sheet

---

***This is an individual piece of coursework; you are not allowed to work in groups and your code must be all your own work. You may not use other peoples' code or ideas, e.g., taken from the Internet. Any form of unfair means (plagiarism or collusion) will be treated as a serious academic offence and will be processed under our University Discipline Regulations. These might result in a loss of marks, a possible failure of the module or even expulsion from our University. Read further details in the UG Student Handbook.***

***Before you begin, you should have completed the Online Quiz 1 and the lab sheets from weeks 1-4. To complete this worksheet you will also need to cover the material for week 5.***

***You may ask demonstrators for help in understanding the problem, but this is intended to be an individual assignment, so you should not ask for help with the programming task.***

***This is a live problem sheet: you are welcome to add questions or suggestions to it as comments, which will be addressed by your instructors.***

---

## Introduction (video brief; alternative link)

Similarly to how you downloaded the code for the second semester (see Week1_Lab1_EclipseSetUp_2023.docx), you are now provided with a Gradle project that implements a system to manage orders in a small coffee house. The code of the project is organised in a main package named **`uk.ac.sheffield.com1003.cafe`** which is split across the following directories:

- **`src/main/java`** contains the source code of the project, including the classes that you should be implementing or completing.
- **`src/test/java`** contains JUnit tests which you can execute to assess if your solution is working as expected. You should **not** modify any of the classes in this directory. If you do, it will not be possible to assess your submission and you will obtain a **zero mark**.

To accept the assignment, visit this GitHub Classroom Assignment. **It is very important that you choose your own student username to accept the assignment (it will be linked to your personal GitHub account).** If you have correctly linked your student username (e.g., aca1jrs) with your Github account for the labs repo, then you may not need to link them again. If you make a mistake or cannot find your own student username, email j.rojas@sheffield.ac.uk. **Do not use anyone else's username**. If your Github account is linked to another student's username, please email me, as this may cause marking issues.

In the source directory (**src/main/java**) of the provided project, you will find the following classes:

- **Cafe**: Represents the main class of the system. A cafe has a `name` of type String, an array of recipes called `menu`, an array of orders called `orders` and two order indexes, one pointing to the index of the next order to be placed and one pointing to the index of the next order to be served. The `Cafe` class has the following public methods: `printMenu()`, `addRecipe()`, `removeRecipe()`, `placeOrder(...)` and `serveOrder()`.
- **Recipe**: A recipe is represented by the `name` of the recipe, e.g., "Large latte", its `price`, `size` and the `ingredients` contained in the recipe.
- **Ingredient**: This is an abstract class, extended by classes Water, Coffee, Milk, and Syrup.
- **Order**: Represents an order placed in the cafe. The information contained in an order includes the date and time in which the order was placed, the name of the customer, a reference to the recipe that has been ordered, the amount of money paid (in pounds), and possibly the customer's special request (e.g., "extra hot").

The **uk.ac.sheffield.com1003.cafe.TestCafe** class in **src/test/java** contains some tests that you may find helpful to develop a correct solution.

***Note**: You may notice that some of the classes still present some level of redundancy or inefficiency that we would ideally like to avoid (e.g. by using public abstract classes and/or interfaces). This is intended to avoid unnecessary complexity and keep the problem sheet achievable. **Refrain from making changes to the provided code (in particular class and method signatures) and focus on implementing the required functionality as requested in this problem sheet**.*

## Tasks

By browsing the project's source code, you will notice that there are several methods in the different classes mentioned above that are either missing or incomplete. Your task is to implement these methods according to the instructions provided in the methods' Javadoc. These are your tasks in this problem sheet:

1. Create a new class named **App** in package **uk.ac.sheffield.com1003.cafe** with a **main** method in which a **Cafe** instance is created with the following three drink **Recipe** instances. You will need to implement and use the method **Cafe.addRecipe(Recipe newRecipe)**.
    - An **"Espresso"**, which is a **small** beverage, costs **£1.5**, and is made with two ingredients: Coffee (default amount 8gr), and Water (default amount 30ml).
    - A **"Double Espresso"** which is a **regular** beverage, costs **£2.2** and is made with the same two ingredients as a single espresso, coffee and water, but in different quantities: 15 grams and 50 millilitres, respectively. For this, you will need to overload the constructors of classes **Coffee** and **Water** to implement constructors **Coffee(int amount)** and **Water(int amount)**.
    - A **"Large Soy Latte"** which is a **large** beverage that costs **£2.5** and uses **soy** milk (you will need to add **SOY** as a milk type in class **Milk**; the rest of the recipe for a nice **Large Soy Latte** is your personal choice).

2. Implement method **`Cafe.printMenu()`** and call it from method **`App.main`**. See the method Javadoc in **`Cafe.java`** and the JUnit test **`TestCafe.testPrintMenu()`** for guidance. You will have to implement **`Cafe.greeting()`** and call it from **`Cafe.printMenu()`**.
3. The cafe is now ready to start serving orders, but the code does not allow them to do so yet. You are asked to implement methods **`Cafe.placeOrder`** and **`Cafe.serveOrder`**.
    - Notice that the method placeOrder should throw two types of exception: **`RecipeNotFoundException`** (if the recipe name received as argument does not exist in the menu) and **`CafeOutOfCapacityException`** (if the maximum number of orders for the cafe has been reached). You must create these exception classes and use them appropriately.
4. The cafe is now up and running and receiving orders. The staff would like you to implement a method that will print the list of pending orders for them: **`Cafe.printPendingOrders`**.
5. Some clients are asking what the difference is between certain drinks. To help staff on this, you must Implement method **`Recipe.equals`** to check if two recipes are the same. Two recipes are *equal* if they are the same size, contain the same ingredients (including units and amounts), and cost the same, *even if their names are different*.
6. The cafe wants to start serving more exciting drinks. Create a class named **`Syrup`** to package **`uk.ac.sheffield.com1003.cafe.ingredients`**. This class must extend class **`Ingredient`**, must contain a private field named **`flavour`**, which is taken as input argument to a constructor (i.e., **`public Syrup(String flavour)`**) and must override the **`toString()`** method to return a string with this format: **`"Syrup [unit={}, amount={}, flavour={}]"`**.
7. After a while, the cafe owners would like to revamp their menu. Implement method **`Cafe.removeRecipe`** to allow them to delete a recipe given its recipe name. If the provided recipe name does not exist, the method should throw a new instance of **`RecipeNotFoundException`** (which you should have implemented in package **`uk.ac.sheffield.com1003.cafe.exceptions`** for task 3).

You are expected to **document** your code and display **meaningful** messages to the console. The quality of the documentation and reporting of results will be assessed.

## Testing your classes

The test directory in your project includes some JUnit tests which are meant to help you assess that your solution is 'correct' and that you are working in the right direction. Your solution code will need to successfully pass the tests provided in these classes. However, note that the test suite provided does not fully test your solution, therefore just because your solution passes all these tests, it does not mean it is fully correct - you must make sure of this yourself. During marking, additional, more thorough tests (not included in the code provided) will be run on your code.
Make sure you watch the supporting video showing how to verify that your code has passed all the JUnit tests provided.

## General Rules and Recommended approach

- Study the code provided, and then complete the methods in the indicated classes (you need to create some classes). You have already been provided with the template methods and should not modify the signatures of these methods (i.e., you are not allowed to change any method's name, return type or arguments).
- Read carefully the Javadoc comments provided in the template code, as they will specify the desired functionality that you are expected to implement.
- Watch the supporting video about how to run the tests in the provided test classes and confirm that your code passes all those tests. If not, please, check your code as you might have a bug there.
- You are encouraged to make good use of git by committing frequently and using descriptive commit messages. Your commit history should give an idea of the work you have done in your solution. That said, you will not be marked down for not using git effectively.
- Once you have passed all the tests provided, keep checking your code as the provided tests do not cover completely your code. Feel free to add your own JUnit tests if you wish to (not marked), but do so in a separate test class, e.g., in MyTests.java (alongside TestCafe.java).
- Once you are happy with your solution, you will need to do two things
    - Make sure that you commit and push your code to your GitHub classroom repository before the deadline.
    - Go to Blackboard to submit your assignment. **You do not need to upload any code to BlackBoard**, just add the link to your latest commit in the comment section of your submission, e.g.,: **https://github.com/tuos-dcs-COM1003/com1003-problem-sheet-[YOUR_GITHUB_USERNAME]/commit/xxxxxxxxxxx**
- After the deadline has passed, standard late submission penalties and extenuating circumstances rules apply.

## Coding tips

Your code should be written using a good coding style, and you should note that **readability** is very important. Following the Google Java guidelines is recommended. In particular, you should:

- Use comments, but only when required (i.e. not excessively, but you should have a JavaDoc comment block at the head of each class).
- Use one code statement per line.
- Code your methods with as few lines as possible, and as many lines as needed. Ideally a complete method should be visible in a code editor without scrolling (see *Clean Cod*e for a longer discussion).
- Adhere to naming conventions; class names in `UpperCamelCase`, method and variable names in `lowerCamelCase`, and constant names in `CONSTANT_CASE`.
- Choose sensible, self-documenting, and descriptive names for all variables, methods, and classes.
- Use indentation consistently with either 2 or 4 whitespaces per indent (use whitespace because tabs can be interpreted differently on different operating systems).

- Before your code is checked, ask yourself whether the code is understandable by someone marking it.

## Assessment and hand-in procedure

This problem sheet is worth 20% of your mark for the Spring semester. Your marks will depend on developing a solution that:

- Compiles without needing modifications. This includes situations in which a character or characters in another language are used in the code[1], or situations where an import to a non-existing package or class causes a compilation error. Solutions that do not compile, will automatically obtain a **zero mark**.
- Implements all the tasks listed in this handout.
- Passes all the tests provided in the test classes, which **must not** be modified.
- The code is appropriately documented.
- The messages displayed in the console are meaningful.

***You must submit your solution by 15:00 on Fri, 17 Mar 2023.***

***Follow the guidance about how to upload your solution in Blackboard. Watch the supporting video for a demo.***

***Start working on this problem sheet as soon as possible so that you can get help from the demonstrators and/or lecturers during the labs.***

***Use the discussion board dedicated to 'Problem sheet' to ask your questions, without sharing your solution.***

***Late submissions will incur standard late penalties.***

[1] For those working on a computer with a different language charset, consider switching to an English charset to avoid potential problems, or confirm before submitting your code that your solution compiles and runs properly in a computer with an English charset (e.g. University computers).