Well done on getting this far in this module OOP basics. This lab combines the joy of coffee with the power of OOP. By the end of this lab, you'll be able to create classes that represent various coffee beverages using inheritance. You'll gain an understanding of how attributes and methods are shared across a class hierarchy, allowing for efficient code reuse. Additionally, you'll learn to utilize common methods in a superclass, enhancing the reusability and organization of your code.

Imagine you are a software developer hired by a company to work on a virtual coffee machine project. As a team member, your task is to develop a system that enables the application to brew various types of coffee. The machine needs to simulate the preparation of different coffee beverages, such as espresso and latte, each with unique characteristics. The machine should recognize different coffee beverages as objects inherited from a Coffee superclass.

Goal

Design and implement a coffee hierarchy using inheritance to represent different coffee beverages (Latte, Espresso) while capturing shared attributes and functionalities.

So, grab your favorite mug, put on your coding hat, and get brewing!

🖥️ Note: When you encounter this icon, it's time to get into your IDE and start coding!

In your lab environment, open IntelliJ by double-clicking on the icon.

You'll build classes for various coffee drinks using inheritance. Here's the plan:

- Starter classes: You have two classes in the starter code:
    - **`Coffee`**, with some basic methods for common coffee attributes.
    - The **`CoffeeMachine`** class contains the **`main()`** method, which provides a menu for users to choose their desired beverage. Based on the selected beverage (Espresso or Latte), the user provides more information about their coffee order. This method can be used to create the objects of your classes and call respective methods to see if your code is working as expected.
- Subclass creation: Create subclasses, like **`Espresso`** and **`Latte`**, that will inherit from the **`Coffee`** class. These subclasses define specific functionalities and details unique to each beverage type.
- Exploring reusability: Explore how the inherited methods from the **`Coffee`** class can be used consistently for different coffee objects.

In this task, you'll examine the class to understand how it represents coffee using attributes like name, roast, caffeine level, and price. You'll also explore how methods within the class set and utilize these attributes, providing insight into object-oriented programming in Java.

To begin, expand the *src* folder and open the provided **`Coffee`** class.

- Observe that four attributes define **`Coffee`**:
  - **`name`**: a String representing, for example, “Espresso.”
  - **`roast`**: a String representing how strong the coffee must be, for example, “medium.”
  - **`caffeineLevelInMg`**: an int storing the caffeine level in milligrams, but its value is calculated based on the roast. For example, if a medium roast is selected, the caffeine level would be 50 mg.
  - **`price`**: a double representing the price of the coffee in dollars (for example, 3.45).
- There is a parameterized constructor with three arguments. It assigns these values to the corresponding attributes and then calls the **`setCaffeineLevel()`** method. This ensures the caffeine level is set based on the chosen roast.

Remember

- Methods can be called inside other methods. After all, you’ve been calling all your methods inside the **`main()`** method!
- A constructor is also a method, although it is slightly special! (to recall, you can refer to *Constructors: Instantiating Objects with Different States*)
- The method **`setCaffeineLevel()`** checks the roast attribute value using an if-else if statement and assigns the appropriate value to the **`caffeineLevel`** attribute.
- The **`grindBeans()`** and **`brewCoffee()`** methods simulate the coffee preparation, and the **`printInfo()`** method is defined to print the **`name`**, **`roast`**, and **`caffeineLevel`** of the coffee in a formatted way.

Now that you understand the basics of the `Coffee` class, you can create subclasses for specific coffee types. These subclasses should inherit the attributes and methods from the `Coffee` class and have their own unique details.

Step 1: Create the Espresso subclass

Espresso is a concentrated coffee shot brewed with hot water forced through finely ground beans under high pressure. And it is served in individual servings called shots. You must represent this as an attribute in your `Espresso` class.

🖥️ **It's time to get coding!**

- 🖥️ **TODO 1: Create a new class named `Espresso`.**
- 🖥️ **TODO 2: Make `Espresso` inherit from the `Coffee` class using the `extends` keyword.**
- 🖥️ **TODO 3: Add an `int` attribute named `numberOfShots` to store the number of servings for this espresso.**
- 🖥️ **TODO 4: Create a parameterized constructor with four arguments, `name`, `roast`, `price`, and `numberOfShots`.**
- 🖥️ **TODO 5: Use `super` to call the constructor of the `Coffee` class and set the `name`, `roast`, and `price`.**
- 🖥️ **TODO 6: Initialize `numberOfShots` using `this`.**
- 🖥️ **TODO 7: Define a new method named `printEspressoDetails()` inside the `Espresso` class. This method should print a message about the number of servings the user orders, the cost per serving, and their total bill.**
    - **For example, "You asked for 3 servings! Every serving of Espresso costs 2.5$. Your total bill is 7.5$".**

Here's the code skeleton for the class with comments to guide you:

```
// TODO 2: inherit from Coffee using the extends keyword

public class Espresso {



    // TODO 3: declare an attribute to store the number of shots (int)



    // TODO 4: constructor to initialize all attributes

    public Espresso(String name, String roast, double price, int numberOfShots) {

        // TODO 5: use super to call the Coffee constructor



        // TODO 6: initialize numberOfShots using this

    }
```

```
    // TODO 7: define a method to print a message about the number of servings,
cost per serving, and total bill



}
```

Step 2: Use the CoffeeMachine class to brew a virtual espresso

Open the provided `CoffeeMachine` class, and complete the following tasks mentioned as TODO comments inside the `main()` method.

🖥️ **It's time to get coding!**

- 🖥️ **TODO 8: Create an object of the `Espresso` class using the parameterized constructor. Use the already declared variables as arguments (in the correct order).**
    - **For example, `Espresso myEspresso = new Espresso(espressoName, espressoRoast, espressoPrice, numberOfShots);`**
- 🖥️ **TODO 9: Call the `grindBeans()` method on your `Espresso` object using the dot operator.**
- 🖥️ **TODO 10: Call the `brewCoffee()` method on your `Espresso` object using the dot operator.**
- 🖥️ **TODO 11: Call the `printInfo()` method on your `Espresso` object using the dot operator.**
- 🖥️ **TODO 12: Call the `printEspressoDetails()` method on your `Espresso` object using the dot operator.**

Pause and check

Well done on achieving your first milestone! Before moving on to the next task, ensure your code compiles correctly.

To do so, run your code using the IDE. So far, you've only created the Espresso class and its object. You can test your `Espresso` class by running the `CoffeeMachine` and selecting the Espresso option.

- **Can you create an `Espresso` object with different roasts and numbers of shots?**
- **Does the output of `printEspressoDetails()` accurately reflect the cost per serving and total bill?**

Make sure you try all the espresso options, such as choosing different roasts and the number of servings.

Step 1: Create the Latte subclass

A latte is a coffee drink made with steamed milk, a layer of milk foam, and optionally flavored with syrup (for example, vanilla or caramel). So, milk type and syrup flavor are the specific attributes that must be added to your **`Latte`** class.

🖥️ It's time to get coding!

- 🖥️ TODO 13: Create a new class named **`Latte`**.
- 🖥️ TODO 14: Make **`Latte`** inherit from the **`Coffee`** class using the **`extends`** keyword.
- 🖥️ TODO 15: Add two String attributes to this class:
  - **`milkType`** to store the type of milk used in the latte. For example, "Whole milk", "Skimmed milk", "Oat milk", or "Almond milk".
  - **`syrupFlavor`** to represent the syrup flavor to be added to the latte. For example, "Vanilla", "Caramel", "Hazelnut", or "None" if no syrup is used.
- 🖥️ TODO 16: Create a parameterized constructor with five arguments, **`name`**, **`roast`**, **`price`**, **`milkType`**, and **`syrupFlavor`**.
- 🖥️ TODO 17: Use **`super`** to call the constructor of the **`Coffee`** class and set the **`name`**, **`roast`**, and **`price`**.
- 🖥️ TODO 18: Initialize **`milkType`** and **`syrupFlavor`** using this.

🖥️ TODO 19: Define a new method named **`printLatteDetails()`** inside the **`Latte`** class. This method prints the milk type and syrup flavor used to prepare the latte and the total bill.

- For example, "Your latte has whole milk and hazelnut flavor. Your total bill is 3.5$".

Here's the code skeleton for the class with comments to guide you:

```
// TODO 14: inherit from Coffee using the extends keyword

public class Latte {



    // TODO 15: declare two attributes to store the milk type and syrup flavor
(String)



    // TODO 16: constructor to initialize all attributes

    public Latte(String name, String roast, double price, String milkType, String
syrupFlavor) {
```

```
        // TODO 17: use super to call the Coffee constructor



        // TODO 18: initialize milkType and syrupFlavor using this

    }



    // TODO 19: define a method to print the details of milk type and syrup flavor
along with the total bill

}
```

Step 2: Use the CoffeeMachine class to brew a virtual latte

In the provided **`CoffeeMachine`** class, complete the following tasks mentioned as TODO comments inside the **`main()`** method.

🖥️ It's time to get coding!

- 🖥️ TODO 20: Create an object of the **`Latte`** class using the parameterized constructor and pass the already declared variables as arguments (in the precise order).
    - For example, **`Latte myLatte = new Latte(latteName, latteRoast, lattePrice, milkType, syrupFlavor);`**
- 🖥️ TODO 21: Call the **`grindBeans()`** method on your **`Latte`** object using the dot operator.
- 🖥️ TODO 22: Call the **`brewCoffee()`** method on your **`Latte`** object using the dot operator.
- 🖥️ TODO 23: Call the **`printInfo()`** method on your **`Latte`** object using the dot operator.
- 🖥️ TODO 24: Call the **`printLatteDetails()`** method on your **`Espresso`** object using the dot operator.

Pause and check

Congratulations on achieving your second milestone! Before proceeding to the next, please make sure your code compiles correctly.

To do so, run your code using the IDE. Since you have both **`Espresso`** and **`Latte`** classes and their objects ready, test your Latte class by running the **`CoffeeMachine`** and selecting the **`Latte`** option.

- Can you create a **`Latte`** object with different roasts, milk types, and syrup flavors?
- Does the output of **`printLatteDetails()`** accurately reflect the billed amount?

Make sure you try all the options, such as choosing different roasts, milk types, and syrup flavors. Also, try saying “no” to the syrup flavor.

1. Run your code using the IDE. If your program is successful, it should function as follows:
    a. A menu should display from which you can select the type of coffee you want to brew (**`Espresso`** or **`Latte`**) or exit the coffee machine.
    b. If you don’t choose to exit, the coffee machine should prompt you to make further choices based on whether you select espresso or latte. For example, selecting espresso should prompt you for the roast and number of servings.
    c. Once you’ve entered all the information about your coffee selection, you receive your virtual cuppa and your bill.
    d. Observe that the menu reappears until you exit the machine.
2. You must try all the options, including the exit.
3. Be very careful with the test values you input. For example:

Prompts that expect a number as input, such as “Enter your choice (1, 2, or 3):” or “How many servings would you like? (a number please):”.

    a. If you provide them with a String or any other value type, you should get an error, and the program should stop running because Java expects a number as input, and you provided something else.

Prompts that expect a String as input, such as "What Roast would you like? (light, medium, dark)":
If you provide them with an integer or any other value type, your program should run smoothly (because you can store any value inside the double quotes of String), but the output should be unexpected and incorrect.

    b. Suppose while ordering an espresso with “light” roast and 2 servings, you’d expect the output to be:

```
You ordered a Espresso with a light roast.

The caffeine level in your coffee is 50 mg.

You asked for 2 servings!
```

```
Every serving of Espresso costs 2.5$. Your total bill is 5.0$.
```

Here, instead of giving light as input, if you input 1, the output would be:

```
You ordered a Espresso with a 1 roast.

The caffeine level in your coffee is 0 mg.

You asked for 2 servings!

Every serving of Espresso costs 2.5$. Your total bill is 5.0$.
```

Observe that the roast is 1 instead of “light”, and the caffeine level is incorrect. It is “0 mg” when it must have been “50 mg”. This happened because the caffeine level depends on the roast, and 1 doesn’t match any of the roast values specified in the **`setCaffeineLevel()`** method.

4. Compare the output of your program to the expected results. A few expected results are given below:

- Selecting **`Espresso`** with **`medium`** roast and 3 servings:

```
Welcome to the Coffee Machine!

Select an option to continue:

1. Espresso

2. Latte

3. Exit

Enter your choice (1, 2, or 3): 1

What Roast would you like? (light, medium, dark): medium

How many servings would you like? (a number please): 3



Grinding beans for Espresso...



Brewing your favorite Espresso...
```

```
You ordered a Espresso with a medium roast.

The caffeine level in your coffee is 100 mg.

You asked for 3 servings!

Every serving of Espresso costs 2.5$. Your total bill is 7.5$.
```

- Selecting **`Latte`** with **`dark roast`**, **`skim milk`**, and **`hazelnut syrup`**:

```
Welcome to the Coffee Machine!

Select an option to continue:

1. Espresso

2. Latte

3. Exit

Enter your choice (1, 2, or 3): 2

What Roast would you like? (light, medium, dark): dark

What milk type would you like? (whole, skim, almond, oat): skim

Would you like syrup? (yes/ no): yes

Which flavor would you like? (vanilla, caramel, hazelnut): hazelnut



Grinding beans for Latte...



Brewing your favorite Latte...



You ordered a Latte with a dark roast.

The caffeine level in your coffee is 150 mg.

Your latte has skim milk and hazelnut flavor.
```

```
Your total bill is 3.5$.
```

- Selecting **`Latte`** with **`medium roast`**, **`oat milk`**, and **`no syrup`**:

```
Welcome to the Coffee Machine!

Select an option to continue:

1. Espresso

2. Latte

3. Exit

Enter your choice (1, 2, or 3): 2

What Roast would you like? (light, medium, dark): medium

What milk type would you like? (whole, skim, almond, oat): oat

Would you like syrup? (yes/ no): no



Grinding beans for Latte...



Brewing your favorite Latte...



You ordered a Latte with a medium roast.

The caffeine level in your coffee is 100 mg.

Your latte has oat milk and no flavor.

Your total bill is 3.5$.
```

- **When you select the `roast` for your coffee, you could enter anything other than the options provided (even “darker” or “lighter” would work), and the program still executes, although the output is unexpected. Try to imply conditions to control the input to the desired values only.**

- **Similarly, when you select the `milkType` and `syrupFlavor` for your `Latte`, you could enter any value other than the ones provided in the prompt's parentheses (for example, "toned" for `milkType`, and "banana" for `syrupFlavor`). Try to apply conditions that would allow only the expected values as input.**

In this lab, you explored the concept of inheritance in Java in a coffee machine scenario. You designed and implemented a coffee hierarchy in which the Coffee superclass captured common attributes such as `name`, `roast`, `caffeineLevel`, and `price`, along with functionalities like `grindBeans`, `brewCoffee`, and `printInfo`. Subclasses such as `Espresso` and `Latte` inherited these attributes and methods, while adding their unique details like the number of shots for Espresso and the milk type and syrup flavor for Latte.

By leveraging inheritance, you achieve code reusability and code maintainability. In this instance, the `Coffee` class served as a blueprint, reducing redundancy when you created subclasses for specific coffee types.