

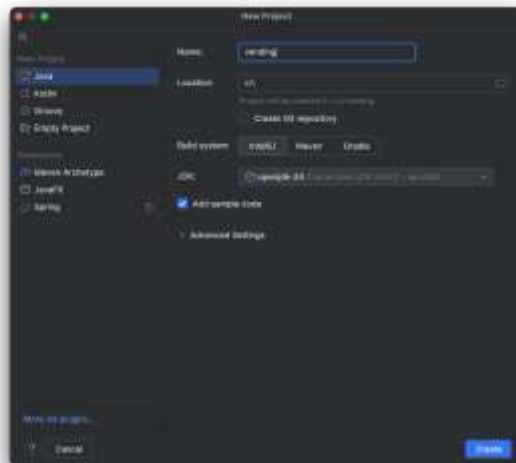
WEEK 3 LAB: CREATING A JAVA CLASS

In this lab you will create a Java class that will model a vending machine. An object created from this class will initially model the way a real vending machine might store the information about the products it vends and their prices, as well as the current balance (the amount of money that has been inserted and not yet used to purchase products). This simple vending machine will vend only coffee, which can be with or without milk. In the next lab you will continue to develop the vending machine so that it is possible to purchase products.

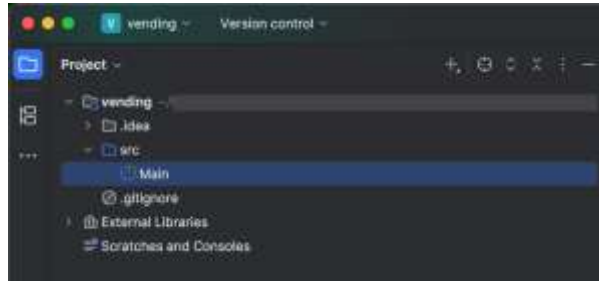
You should attempt the steps for each task and try to answer the **questions** that follow some of the steps.

Task 1 : Creating the VendingMachine class

1. Start IntelliJ and close any open projects. In the **Welcome** dialogue choose the **New Project** option. In the **New Project** dialogue, enter *vending* for the project name, and choose the location where you want to store the project. Choose *Language: Java* and *Build system: IntelliJ*, which are likely to be selected by default. Ensure that a JDK is selected, and check the **Add sample code** box. Click **Create**.



2. IntelliJ should open your new project, which should have one class, *Main*, in the Project window.



- Right-click on the `src` folder in the Project window and choose **New > Java Class** from the pop-up menu. In the **New Java Class** dialogue, enter *VendingMachine* for the name. Make sure you accept the default to create a **Class** rather than any of the other options which may be shown.

VendingMachine should appear as an additional class in the Project window, and should be opened in an editor tab (double-click on the class to open if it isn't open). The code for your class should now simply be:

```
public class VendingMachine
{

}
```

Task 2 : Fields and constructor

- A *VendingMachine* object needs **fields** to keep track of some important information, such as the price of the products and the number in stock in the machine. The complete list of fields needed is as follows:

| Field name | Data type | Purpose |
|-------------------------|-----------|---|
| stockWithMilk | int | The number of coffees with milk available |
| stockWithoutMilk | int | The number of coffees without milk available |
| priceWithMilk | double | The price of coffee with milk |
| priceWithoutMilk | double | The price of coffee without milk |
| balance | double | The amount of money inserted which has not yet been applied to a purchase |

Add the code to define these fields into your *VendingMachine* class. Compile the class and try to fix any errors. Your code must compile successfully before you go on to the next step. Build the project (**Build > Build project** option on the top menu) and fix errors as necessary. After that, IntelliJ will automatically compile classes when their code is changed.

Q2.1 Explain the different choice of data types for the fields representing stock and prices. Are these good choices?

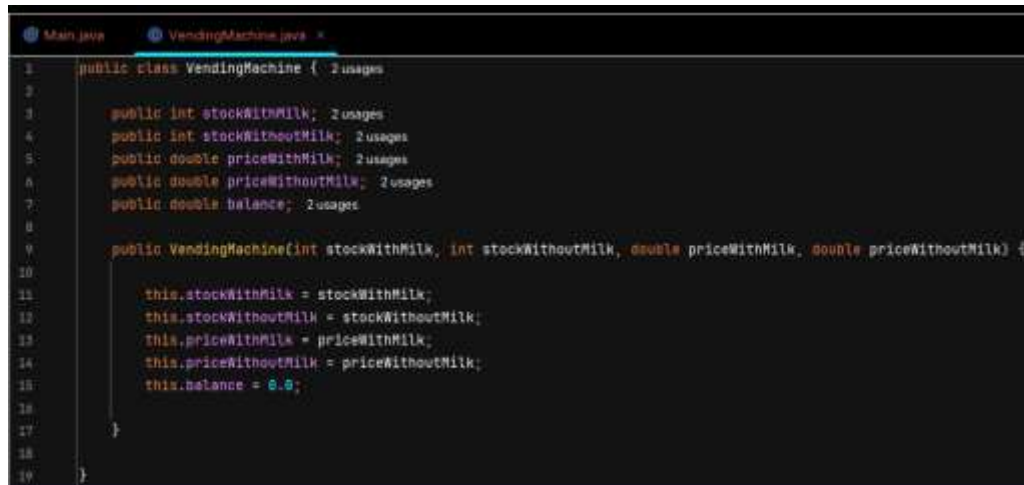
A2.1: Yes, these are good choices. Stock needs to be integer, as we cannot have number of coffees with point decimals. Prices should be double, as they can have cents.

- The *VendingMachine* class needs a constructor which initialises objects as

follows:

- The fields *stockWithMilk* and *stockWithoutMilk* should be initialised using parameter values
- The fields *priceWithMilk* and *priceWithoutMilk* should also be initialised using parameter values
- The field *balance* should be set to *0.0* initially

Add code to define a suitable constructor into your class.



```

1 public class VendingMachine {
2
3     public int stockWithMilk;
4     public int stockWithoutMilk;
5     public double priceWithMilk;
6     public double priceWithoutMilk;
7     public double balance;
8
9     public VendingMachine(int stockWithMilk, int stockWithoutMilk, double priceWithMilk, double priceWithoutMilk) {
10
11         this.stockWithMilk = stockWithMilk;
12         this.stockWithoutMilk = stockWithoutMilk;
13         this.priceWithMilk = priceWithMilk;
14         this.priceWithoutMilk = priceWithoutMilk;
15         this.balance = 0.0;
16
17     }
18
19 }

```

3. To **test your constructor**, you should write test code in the *main* method (in the class *Main*), replacing any code that is currently in *main*.
4. Write code to create a new instance of *VendingMachine* with identifier *vend1*. Your code should include values for four parameters. You can decide how much stock there will be and the price of each product.

Run the program. If there are no errors the program will run, but you will not see any output.

5. Add code to print the values of all the fields of the object, so that you can check that the object has been created correctly. For example, to print the field *stockWithMilk* use the following code. Note that IntelliJ will probably give you a pop-up list of possible options as you type *vend1* followed by a dot.

```
System.out.println(vend1.stockWithMilk);
```



```

1 public class Main {
2     public static void main(String[] args) {
3
4         VendingMachine vend1 = new VendingMachine(
5             stockWithMilk: 10, stockWithoutMilk: 10, priceWithMilk: 2.5, priceWithoutMilk: 1.5);
6         System.out.println(vend1.stockWithMilk);
7         System.out.println(vend1.stockWithoutMilk);
8         System.out.println(vend1.priceWithMilk);
9         System.out.println(vend1.priceWithoutMilk);
10        System.out.println(vend1.balance);
11    }
12 }

```

Run the program, and check that the stock and price fields have been initialised

correctly and that the *balance* field has the expected value.

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDE
10
10
2.5
1.5
0.0

Process finished with exit code 0
```

Q2.2 In the pop-up list mentioned in step 5, what symbol is shown beside the names *stockWithMilk*, *stockWithoutMilk*, and so on? what does this tell you?



A2.2: The symbol shown is an “f”, which indicates this is a field or attribute of this object.

Task 3 : Adding a default constructor

1. A class can have more than one constructor, and it is often useful to have a default constructor that allows an object to be created with sensible default values for fields when no parameter values are supplied. The *VendingMachine* needs a default (parameterless) constructor which initialises objects as follows:
 - The fields *stockWithMilk* and *stockWithoutMilk* should be initialised to the value 0
 - The fields *priceWithMilk* and *priceWithoutMilk* should be initialised to the value 0.0

Add code to define a suitable constructor into your class. Compile and fix errors as necessary.

2. To **test your constructor**, add code to *main* to create a new instance of *VendingMachine* using an empty parameter list so the default constructor is called.

Add code to print the values of the stock fields to check that the defaults have been applied. Rather than printing the names only, you should write an expression in the *System.out.println* call that uses concatenation (joining strings together using the + operator) to give output similar to this:

```
This machine contains: 0 coffee with milk and 0 coffee
without milk
```

You should now have a *VendingMachine* class that has the correct fields and can be instantiated and initialised in two different ways.

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

        VendingMachine vend1 = new VendingMachine(10, 10, 2.5, 1.5);
        System.out.println(vend1.stockWithMilk);
        System.out.println(vend1.stockWithoutMilk);
        System.out.println(vend1.priceWithMilk);
        System.out.println(vend1.priceWithoutMilk);
        System.out.println(vend1.balance);

        VendingMachine vend2 = new VendingMachine();
        System.out.println("This machine contains: " + vend2.stockWithMilk + " coffee with milk and " +
            vend2.stockWithoutMilk + " coffee without milk");
    }
}
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program F
10
10
2.5
1.5
0.0
This machine contains: 0 coffee with milk and 0 coffee without milk

Process finished with exit code 0
```

Q3.1 Do you have any lines of code that appear and are identical in both constructors? Repetition of code should generally be avoided – why? If you have repeated code in your constructors, could you rewrite one of them to avoid this?

A3.1: I can use the method *this()*, in the default constructor, to make a call to the non-default constructor to reduce boilerplate code:

```

public class VendingMachine { 4 usages

    public int stockWithMilk; 3 usages
    public int stockWithoutMilk; 3 usages
    public double priceWithMilk; 2 usages
    public double priceWithoutMilk; 2 usages
    public double balance; 2 usages

    public VendingMachine(int stockWithMilk, int stockWithoutMilk, double priceWithMilk, double priceWithoutMilk) {
        this.stockWithMilk = stockWithMilk;
        this.stockWithoutMilk = stockWithoutMilk;
        this.priceWithMilk = priceWithMilk;
        this.priceWithoutMilk = priceWithoutMilk;
        this.balance = 0.0;
    }

    public VendingMachine() 1 usage
    {
        this(stockWithMilk: 0, stockWithoutMilk: 0, priceWithMilk: 0.0, priceWithoutMilk: 0.0);
    }
}

```

Q3.2 What would happen if you tried to create an object with the statement `VendingMachine vend2 = new VendingMachine (3,3);`?

A3.2: A compile time error, given our constructor requires 4 parameters (unless we call the default constructor).

Task 4 : Accessing fields

1. In the previous task you printed the values of fields of the *VendingMachine* instances you created. As a reminder, add code to *main* to create a new instance *vend3* of *VendingMachine* and to print the value of the *balance* field *vend3.balance*.
2. You can also change the value of a field using code. Add the following statement:

```
vend3.balance = 5.0;
```

Add code to print the value of the *balance* field of this object again and check that it has changed.

```

Main.java  VendingMachine.java
1  public class Main {
2      public static void main(String[] args) {
3
4          System.out.println("The stock with milk is: " + vend1.stockWithMilk);
5          System.out.println("The stock without milk is: " + vend1.stockWithoutMilk);
6          System.out.println("The price with milk is: " + vend1.priceWithMilk);
7          System.out.println("The price without milk is: " + vend1.priceWithoutMilk);
8
9          // Calling the constructor with no parameters
10         VendingMachine vend2 = new VendingMachine();
11         System.out.println("This machine contains: " + vend2.stockWithMilk + " coffee with milk and " +
12             vend2.stockWithoutMilk + " coffee without milk");
13
14         VendingMachine vend3 = new VendingMachine(stockWithMilk: 15, stockWithoutMilk: 10, priceWithMilk: 2.5, priceWithoutMilk: 1.5);
15         System.out.println("The balance of vend3 is: " + vend3.balance);
16         vend3.balance = 5.0;
17         System.out.println("The balance of vend3 is: " + vend3.balance);
18     }
19 }
20
21
22

```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files
The stock with milk is: 10
The stock without milk is: 10
The price with milk is: 2.5
The price without milk is: 1.5
This machine contains: 0 coffee with milk and 0 coffee without milk
The balance of vend3 is: 0.0
The balance of vend3 is: 5.0

Process finished with exit code 0
```

Q4.1 What two things can you do with the value of an instance variable using code?

A4.1: We can read (get) and write (set) its value using code.

The previous step actually shows up a serious problem with the *VendingMachine* class at this point. The *balance* field value depends on money inserted and money applied to purchases, so should only change when money is inserted and when a product is vended. However, because you can (and did) change the value of *balance* to some arbitrary value then the *VendingMachine* object is not correctly modelling the way a vending machine should behave.

Can you think how you could prevent this problem occurring? In the next lab you will modify the *VendingMachine* class to do so.

We can first set the field to private, so it is only accessible to the class, and then create methods to calculate the balance after money is inserted and product is vended.