# SOFTENG 281: Object-Oriented Programming
# Assignment 2 (20% of final grade)

**Due:** 9:00pm, 11th April 2022 (Start of Week 7)

## Learning outcomes

The purpose of this assignment is to target the following learning outcomes:

- Gain confidence modelling an object-oriented programming (OOP) problem.
- Gain confidence programming in Java.
- Apply OOP concepts such as inheritance, polymorphism, and encapsulation (information hiding).

## 1 The SOFTENG 281 Burger Shop

Your client opened a Burger Shop in Auckland CBD, and wants you to implement a Java program to manage the orders. The shop is a takeaway restaurant without seated spots. Customers make an order via a terminal. When an order is ready, the customers pay and take the food away. For simplicity, let's assume that there is only one terminal, and only one customer can use the terminal at any given time. The available commands to be used at the terminal are:

```
menu         [no args]                shows the Menu
add          [1 arg, Menu item ID]    adds to the CART the Menu item with the specified ID
add-combo    [no args]                asks the user to choose the elements of a combo meal
remove       [1 arg, cart position]   removes the ith element in the cart
clear-cart   [no args]                removes all elements in the cart
cart         [no args]                shows the content of the cart
order        [no args]                confirms the order
exit         [no args]                exit the terminal
```

You have been given an incomplete implementation of the application, with only the command line interface implemented. You need to implement the commands using Object-Oriented Programming! We recommend using Eclipse for doing the assignment (see Section 3), and using the Maven wrapper before submitting to check that everything compiles fine and the test cases pass. We also suggest you do this assignment with a full stomach; some burgers and snacks sound very delicious! Let's begin!

## 2 Files provided

**maven wrapper** ( `mvnw` for Unix/Mac OS and `mvnw.cmd` for Windows). This file will allow you to build and run this assignment. We will use a Maven wrapper to guarantee that we all have a common configuration to run and test the assignment. You can either build and run the code in your machine with the Maven wrapper via the command line, or in Eclipse (`https://softeng281.digitaledu.ac.nz/resources/maven-wrapper-eclipse`). Ultimately, once you finish your project and before you submit, you will want to make sure that your project compiles without errors and passes the test cases using the Maven wrapper via the command line. This is how your project will be marked. There are three main Maven commands relevant to this assignment:

**clean** ( `./mvnw clean` for Unix/Mac OS and `.\mvnw.cmd clean` for Windows) Removes the Java binary files (*.class) by clearing the `target` directory into which Maven normally builds your project. Running `clean` periodically is useful, for example to remove those `.class` files referring to deleted or renamed Java source files.

**compile** ( `./mvnw compile` for Unix/Mac OS and `.\mvnw.cmd compile` for Windows) Compiles the Java classes (will only compile the Java source files that have changed from the previous build). Run this command before starting the assignment to make sure that is compiling correctly. You should see a `Build Success`:

```
[INFO] ------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------
[INFO] Total time:  1.007 s
[INFO] Finished at: 2022-02-13T16:06:39+08:00
[INFO] ------------------------------------------------
```

**test** ( `./mvnw test` for Unix/Mac OS and `.\mvnw.cmd test` for Windows) Compiles the Java classes and runs the test cases (will only compile the files that have changed from the previous build).

**exec:java** ( `./mvnw exec:java` for Unix/Mac OS and `.\mvnw.cmd exec:java` for Windows) Compiles the Java classes and executes the `Main` class, which will launch the application.

Note that the first time you run the wrapper might take some time (a couple of minutes) to download the required libraries and dependencies.

You can append commands together. For example, `./mvnw clean test` for Unix/Mac OS and `.\mvnw.cmd clean test` for Windows execute the `clean` command followed by the `test` command.

For more information about Maven, visit the related page in our course website `https://softeng281.digitaledu.ac.nz/topics/development-environment/`.

**src/main/java/nz/ac/auckland/se281/a2/cli/Main.java** This class implements the command line interface (CLI). **You should not modify this class**.

**src/main/java/nz/ac/auckland/se281/a2/cli/Menu.java** This class implements the Menu. **You should not modify this class**. Do not add or remove items in the Menu, otherwise the test cases we will use for marking might not work as expected. You will likely use the `enum SIZE` declared in the `Menu` class.

**src/main/java/nz/ac/auckland/se281/a2/cli/MessagesCLI.java** This class declares the messages of the application. **You should not modify this class**. You are encouraged to refer to these Messages in the code you write, as it will be easier for you to avoid spelling and formatting issues. For example, to get the "discount" message ("You are spending $100 or more, we applied 25% discount!") as a String, you can invoke `MessagesCLI.DISCOUNT.getMessage()`. To print the message, `MessagesCLI.DISCOUNT.printMessage()`.

**`src/main/java/nz/ac/auckland/se281/a2/cli/BurgerShop.java`** This class declares the methods that are invoked by the command line interface. This is where you need to start your coding. Of course, you cannot change the signature of the existing methods of this class (such as changing method names, parameters, and return types). You can (and you should) add instance fields and new methods. If you need to instantiate/initialize fields of the object `BurgerShop` you can only use the provided constructor (`public BurgerShop()`).

**`src/test/java/nz/ac/auckland/se281/a2/BurgerShopTestSuite.java`** This Java file contains the JUnit test cases for this assignment. These test cases make sure that you have implemented most of the assignment correctly. Making sure that your program passes all of the tests you have been given will not guarantee that you will get full marks for this assignment, but it will mean that you get **at least** half of the marks for the parts that work according to those tests. In fact, we only provided to you half of the test cases that we will use for marking your assignment. Because we wanted to give you freedom in designing the application following object-oriented concepts, these are not traditional JUnit test cases like those used in A1. They do not directly invoke the code (as we don't know how you are going to implement it, which classes and methods you will create, etc...). Instead, they test the application from the command line interface, simulating a customer typing the commands with the keyboard and observing the result shown on the screen.

Writing your own tests or extending the ones you have been given is strongly recommended. Add them inside `BurgerShopTestSuite.YourTests`, which you can find at the bottom of the test class. You also need to uncomment as a Suite Class on top of the test class:

```
@RunWith(Suite.class)
@SuiteClasses({ BurgerShopTestSuite.Task1Test.class, //
        // BurgerShopTestSuite.Task2Test.class, //
        // BurgerShopTestSuite.Task3Test.class, //
        // BurgerShopTestSuite.Task4Test.class, //
        BurgerShopTestSuite.YourTest.class   // <- UNCOMMENT HERE
                })
```

You can easily write a test using the helper methods `runCommands` and `assertContains` and `assertDoesNotContain` (which are implemented by us—they are not standard JUnit). First, with `runCommands` you specify the commands to be executed (separated by a comma). Then, with `assertContains` you specify what is the text that you expect to see on the console after executing all commands. For example, this test case checks that the add command correctly adds a Cheese Burger to the cart:

```
@Test
public void add_one_cheese_burger() {
      runCommands(ADD + " 0", SHOW_CART);
      assertContains("0 - Cheese Burger: $15.50");
}
```

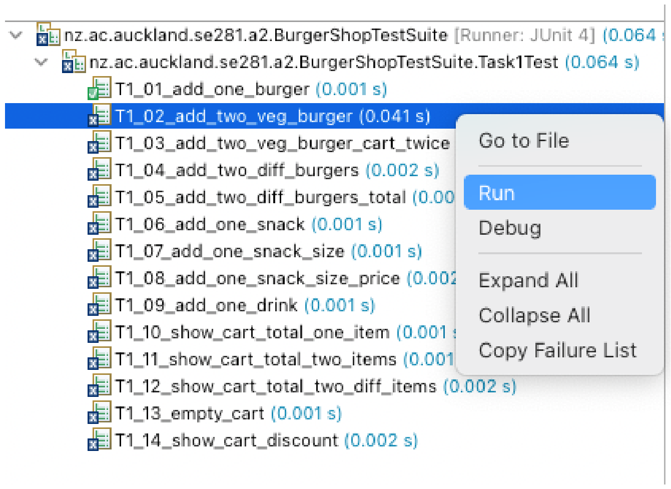
`ADD` and `SHOW_CART` are the enums imported from `nz.ac.auckland.se281.a2.cli.Main`, which corresponds to the `add` and `cart` commands.

While you can add as many tests as you want, we will not consider them when marking your assignment. Also, it is important that you do not change the existing tests. Be aware that your code must still work with the original version of the `BurgerShopTestSuiteTestSuite.java` file, where we will reuse the test cases initially provided to you (plus adding other test cases not visible to you).

Note that we also provide additional "hidden" files (those starting with ".")—you need to change the folder options to be able to see them. We provide a `.gitignore` file that specifies intentionally untracked files that Git should ignore, and `.mvn` a folder containing the Maven wrapper. You should not change these files, and you should not delete them.

## 3 Hot Tip: Eclipse JUnit Integration

While you still need to check that your code works using the Maven wrapper, you can work more efficiently in Eclipse using its JUnit integration. In particular, you can execute test cases individually. See the course website on how to use the JUnit GUI in Eclipse `https://softeng281.digitaledu.ac.nz/resources/junit-eclipse/`.



```
nz.ac.auckland.se281.a2.BurgerShopTestSuite [Runner: JUnit 4] (0.064
  nz.ac.auckland.se281.a2.BurgerShopTestSuite.Task1Test (0.064 s)
    T1_01_add_one_burger (0.001 s)
    T1_02_add_two_veg_burger (0.041 s)
    T1_03_add_two_veg_burger_cart_twice
    T1_04_add_two_diff_burgers (0.002 s)
    T1_05_add_two_diff_burgers_total (0.00
    T1_06_add_one_snack (0.001 s)
    T1_07_add_one_snack_size (0.001 s)
    T1_08_add_one_snack_size_price (0.002
    T1_09_add_one_drink (0.001 s)
    T1_10_show_cart_total_one_item (0.001 s
    T1_11_show_cart_total_two_items (0.001
    T1_12_show_cart_total_two_diff_items (0.002 s)
    T1_13_empty_cart (0.001 s)
    T1_14_show_cart_discount (0.002 s)
```

Figure 1: You can run test cases individually with the Eclipse JUnit Integration

## 4 Object-Oriented Programming

For this assignment, you should use the OO programming concepts we covered in class. As such, you should not implement all your code inside `BurgerShop.java`. You should create appropriate classes and objects. We are giving you complete freedom on how to do the OO design. You will decide how many and which Java classes to create. However, to be sure that you are using inheritance, we will check for the following:

Your code should have at least one abstract class

**AND**

Your code should have at least three (sub)classes that extend a class that you have created.

Note that, not all the sub-classes necessarily need to extend the abstract class. It is up to you. For example, you meet the requirement if you create an abstract class `public abstract class A`, a `public class B`, and if you have three (sub)classes `public class C extends A`, `public class D extends B`, and `public class E extends B`. As another example, you meet the requirement if you create an abstract class `public abstract class A`, and if you have three (sub)classes `public class B extends A`, `public class C extends A`, and `public class D extends A`. You can of course, create more classes on top of these, depending on your OO design choices. It goes without saying that just creating the classes is not sufficient. Your code has to "use" the classes.

## 5 Tasks

Before proceeding with the tasks, read the code provided to you and start to familiarise yourself with the command-line interface. To run the application in Eclipse run the `Main` class. With the Maven wrapper ( `./mvnw exec:java` for Unix/Mac OS and `.\mvnw.cmd exec:java` for Windows) or with Eclipse by right-clicking the class `Main.java` → run As → Java Application. Try the command `menu`, will print the menu of the Burger Shop:

```
burger-shop>menu
ID - ITEM                          PRICE NZD
---------------------------------------
BURGERS
0  - Cheese Burger----------------- 15.50
1  - Avocado Burger---------------- 17.00
2  - Vegan Burger------------------ 18.50
3  - Fisherman Burger-------------- 17.00
4  - Buffalo Chicken Burger-------- 13.00
5  - Black Truffle Burger---------- 27.50
6  - Crispy Foie Gras Burger------- 34.00
SNACKS
7  - Chips-------------------------  7.50
8  - Sweet Potato Chips------------ 10.00
9  - Onion Rings-------------------  5.00
10 - Buffalo Chicken Wings--------- 12.00
11 - Fish Fingers------------------  8.00
DRINKS
12 - Coke--------------------------  2.00
13 - Sprite------------------------  2.00
14 - Fanta-------------------------  2.00
15 - Milkshake---------------------  4.00
```

Try the other commands, they will not perform the intended actions! Only `menu` and `exit` works. It's time to implement the other commands! Let's begin!

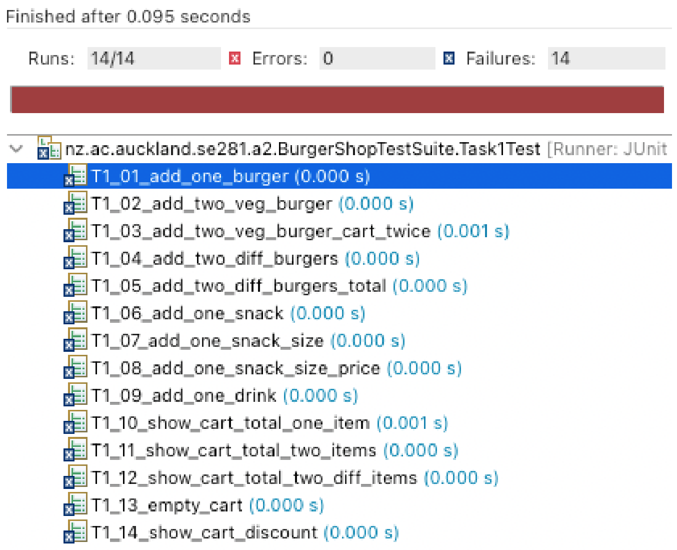
## 5.1 Add and Cart commands [Task 1: 5%]

As soon as you compile and run the tests (using Eclipse to run the `BurgerShopTestSuite` class), you will notice the following output:

With the Maven wrapper:

```
[INFO]
[ERROR] Tests run: 14, Failures: 14, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.600 s
[INFO] Finished at: 2022-03-11T11:57:21+08:00
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.0:test
(default-test) on project a2: There are test failures.
[ERROR]
```

With the JUnit integration of Eclipse:



```
Finished after 0.095 seconds

Runs: 14/14    Errors: 0    Failures: 14

nz.ac.auckland.se281.a2.BurgerShopTestSuite.Task1Test [Runner: JUnit
    T1_01_add_one_burger (0.000 s)
    T1_02_add_two_veg_burger (0.000 s)
    T1_03_add_two_veg_burger_cart_twice (0.001 s)
    T1_04_add_two_diff_burgers (0.000 s)
    T1_05_add_two_diff_burgers_total (0.000 s)
    T1_06_add_one_snack (0.000 s)
    T1_07_add_one_snack_size (0.000 s)
    T1_08_add_one_snack_size_price (0.000 s)
    T1_09_add_one_drink (0.000 s)
    T1_10_show_cart_total_one_item (0.001 s)
    T1_11_show_cart_total_two_items (0.000 s)
    T1_12_show_cart_total_two_diff_items (0.000 s)
    T1_13_empty_cart (0.000 s)
    T1_14_show_cart_discount (0.000 s)
```

Figure 2: Output of JUnit for Task 1 before implementing the commands `add` and `cart`.

This is telling you that 14 test cases were run, none of them passed, and all of them failed. In the output, you will find more details on which particular tests failed.

For example, let's consider the first test case.

```
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public static class Task1Test extends TaskTest {

        @Test
        public void T1_01_add_one_burger() {
                runCommands(ADD + " 1", SHOW_CART);
                assertContains("0 - Avocado Burger: $17.00");
        }
```

This test case executes the command `add 1` followed by the command `cart`, which adds the Avocado Burger (which has ID 1) in the cart and then prints the content of the cart. In other words, the test is expecting the following behavior:

```
burger-shop>add 1
burger-shop>cart
0 - Avocado Burger: $17.00
Total: $17.00
```

Line `runCommands(ADD + " 1", SHOW_CART);` executes the commands `add 1` and `cart`. Line `assertContains("0 - Avocado Burger: $17.00");` will make the test pass (green) if, after executing the two commands, the console contains the String "0 - Avocado Burger: $17.00". It will make the test fail, if, after executing the two commands, the console **does not contain** the String "0 - Avocado Burger: $17.00". The test case is failing because the commands `add 1` and `cart` are not implemented yet.

```
burger-shop>add 1
burger-shop>cart
burger-shop>
```

You can get the information about what is the expected behavior of each test case by reading the test case implementation or by reading the console output of the test case. For example, for the first test case the failure message in the console output is:

```
java.lang.AssertionError: The test is expecting the following output in the console but was not there
0 - Avocado Burger: $17.00
at org.junit.Assert.fail(Assert.java:88)
at nz.ac.auckland.se281.a2.BurgerShopTestSuite$TaskTest.assertContains(BurgerShopTestSuite.java:85)
at nz.ac.auckland.se281.a2.BurgerShopTestSuite$Task1Test.T1_01_add_one_burger(BurgerShopTestSuite.java:126)
```

Now go to `BurgerShop.java` and implement the methods related to the command `add` : `addBurger()`, `addSnack()`, `addDrink()` and the command `cart`: `showCart()`. The requirements are:

- Each item in the Menu has an ID. The customer types `add ID` to insert the item with that ID into the cart.
- For snacks and drinks, the CLI asks for the size of the snack or drink. There are three possible sizes: M, L, and XL. The price in the Menu refers to the medium M size. For both Snack and Drink, if the customer chooses Large (L), the price should be incremented by $3, while it is incremented by $4 if the customer chooses eXtra Large (XL). Note that, the price given to the method `addSnack()` and `addDrink()` refers to the medium M size. The item added to the cart should have the price incremented accordingly. Size "M" no increment, size "L" plus $3, and size "XL" plus $4.
- Each item in the cart should be represented exactly like this:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| position in the cart (starting from zero) | white space | - char | white space | item's name | : char | white space | $ char | price format #.## |

  For example:

```
burger-shop>add 1
burger-shop>cart
0 - Avocado Burger: $17.00
```

- The items in the cart are ordered by following the insertion order. For example:

```
burger-shop>add 5
burger-shop>add 3
burger-shop>cart
0 - Black Truffle Burger: $27.50
1 - Fisherman Burger: $17.00
Total: $44.50
```

- If the customer adds the same item more than once, the item will be repeated in the cart. For example:

```
burger-shop>add 1
burger-shop>add 1
burger-shop>cart
0 - Avocado Burger: $17.00
1 - Avocado Burger: $17.00
```

  To print the prices with the correct format (only two decimals), you can use `String.format("%.02f", value)`. Also, to be safe, it is better to always use float data type in this assignment and not doubles data type. This to avoid small differences in decimals when running the test cases (e.g., 4.99 vs 5.00).

- The name of the item is an input of the `addBurger()`, `addSnack()`, `addDrink()` methods. For drink and snacks, you need to append | white space | "(M)" | | white space | "(L)" | | white space | "(XL)" | at the end of the name, based on the size chosen by the customer. For example:

```
burger-shop>add 8
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): M
burger-shop>add 13
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): L
```

```
burger-shop>add 7
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): XL
burger-shop>cart
0 - Sweet Potato Chips (M): $10.00
1 - Sprite (L): $5.00
2 - Chips (XL): $11.50
```

- After showing the ordered items in the cart, the command `cart` gives the total amount in the cart. The total should be represented exactly like this:

  | "Total" | : char | white space | $ char | price format #.## |
  |---|---|---|---|---|

  For example:

  ```
  burger-shop>add 1
  burger-shop>add 9
  Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): L
  burger-shop>cart
  0 - Avocado Burger: $17.00
  1 - Onion Rings (L): $8.00
  Total: $25.00
  ```

- if the cart is empty, then the command `cart` will print "Cart is empty" (you can use `MessagesCLI.CART_EMPTY.printMessage()`).
- If the total price is $100 or higher, before the total the command `cart` should print "You are spending $100 or more, we applied 25% discount!" and show the updated total after applying the discount (you can use `MessagesCLI.DISCOUNT.printMessage()`.).

For example:

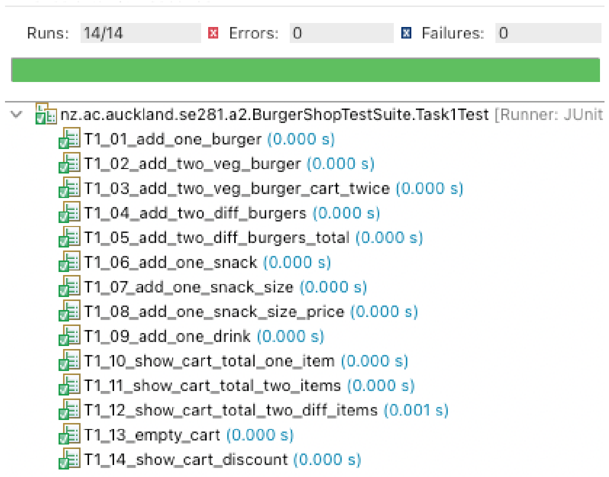
```
burger-shop>add 1
burger-shop>add 9
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): L
burger-shop>cart
0 - Avocado Burger: $17.00
1 - Onion Rings (L): $8.00
Total: $25.00
burger-shop>clear-cart
burger-shop>add 3
burger-shop>cart
0 - Fisherman Burger: $17.00
Total: $17.00
burger-shop>add 3
burger-shop>cart
0 - Fisherman Burger: $17.00
1 - Fisherman Burger: $17.00
Total: $34.00
burger-shop>add 6
burger-shop>add 9
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): XL
burger-shop>add 14
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): M
burger-shop>add 8
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): L
burger-shop>cart
```

```
0 - Fisherman Burger: $17.00
1 - Fisherman Burger: $17.00
2 - Crispy Foie Gras Burger: $34.00
3 - Onion Rings (XL): $9.00
4 - Fanta (M): $2.00
5 - Sweet Potato Chips (L): $13.00
Total: $92.00
burger-shop>add 2
burger-shop>cart
0 - Fisherman Burger: $17.00
1 - Fisherman Burger: $17.00
2 - Crispy Foie Gras Burger: $34.00
3 - Onion Rings (XL): $9.00
4 - Fanta (M): $2.00
5 - Sweet Potato Chips (L): $13.00
6 - Vegan Burger: $18.50
You are spending $100 or more, we applied 25% discount!
Total: $82.88
```

When all the test cases are passing (all green) and you are sure that you have correctly followed the requirements, you can move to Task 2! Remember to commit frequently.
With the Maven wrapper:

```
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.627 s
- in nz.ac.auckland.se281.a2.BurgerShopTestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.010 s
[INFO] Finished at: 2022-03-11T12:24:19+08:00
[INFO] ------------------------------------------------------------------------
```

With the JUnit integration of Eclipse:



```
Runs: 14/14    Errors: 0    Failures: 0

nz.ac.auckland.se281.a2.BurgerShopTestSuite.Task1Test [Runner: JUnit
  T1_01_add_one_burger (0.000 s)
  T1_02_add_two_veg_burger (0.000 s)
  T1_03_add_two_veg_burger_cart_twice (0.000 s)
  T1_04_add_two_diff_burgers (0.000 s)
  T1_05_add_two_diff_burgers_total (0.000 s)
  T1_06_add_one_snack (0.000 s)
  T1_07_add_one_snack_size (0.000 s)
  T1_08_add_one_snack_size_price (0.000 s)
  T1_09_add_one_drink (0.000 s)
  T1_10_show_cart_total_one_item (0.000 s)
  T1_11_show_cart_total_two_items (0.000 s)
  T1_12_show_cart_total_two_diff_items (0.001 s)
  T1_13_empty_cart (0.000 s)
  T1_14_show_cart_discount (0.000 s)
```

Figure 3: Output of JUnit for Task 1 after implementing the commands `add` and `cart`.

## 5.2 Add-combo command [Task 2: 3%]

Go at the beginning of `BurgerShopTestSuite` and uncomment `BurgerShopTestSuite.Task2Test.class` to enable the executions of Task 2 test cases.

```
@RunWith(Suite.class)
@SuiteClasses({ BurgerShopTestSuite.Task1Test.class, //
        BurgerShopTestSuite.Task2Test.class, //
        // BurgerShopTestSuite.Task3Test.class, //
        // BurgerShopTestSuite.Task4Test.class, //
        // BurgerShopTestSuite.YourTests.class //
})
```

Run the test classes you should see that 8 test cases are failing. Go to `BurgerShop.java` and implement the method related to the command `add-combo` : `addCombo()`. The requirements are:

- A combo item is added and represented in the same way as a menu item (see Task 1). However, the name of a Combo item in the cart should be represented exactly like this:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COMBO | white space | : | white space | ( | burger name | , | white space | snack name | , | white space | drink name | ) |

- The price of a combo is the sum of all the items, but the drink would be half price. Note that in a combo, both snacks and drinks have the same size, and the combo price must consider the size.

For example:

```
burger-shop>add-combo
Choose a Burger ID : 3
Choose a Snack ID : 11
Choose a Drink ID : 15
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): M
burger-shop>cart
0 - COMBO : (Fisherman Burger, Fish Fingers (M), Milkshake (M)): $27.00
Total: $27.00
burger-shop>add-combo
Choose a Burger ID : 5
Choose a Snack ID : 8
Choose a Drink ID : 12
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): L
burger-shop>add-combo
Choose a Burger ID : 2
Choose a Snack ID : 10
Choose a Drink ID : 14
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): XL
burger-shop>cart
0 - COMBO : (Fisherman Burger, Fish Fingers (M), Milkshake (M)): $27.00
1 - COMBO : (Black Truffle Burger, Sweet Potato Chips (L), Coke (L)): $43.00
2 - COMBO : (Vegan Burger, Buffalo Chicken Wings (XL), Fanta (XL)): $37.50
You are spending $100 or more, we applied 25% discount!
Total: $80.63
```

When all the test cases are passing (all green) and you are sure that you have correctly followed the requirements, you can move to Task 3! Remember to commit frequently.

## 5.3 remove and clear commands [Task 3: 2%]

Go at the beginning of `BurgerShopTestSuite` and uncomment `BurgerShopTestSuite.Task3Test.class` to enable the executions of Task 3 test cases.

```
@RunWith(Suite.class)
@SuiteClasses({ BurgerShopTestSuite.Task1Test.class, //
        BurgerShopTestSuite.Task2Test.class, //
        BurgerShopTestSuite.Task3Test.class, //
        // BurgerShopTestSuite.Task4Test.class, //
        // BurgerShopTestSuite.YourTests.class //
                })
```

Run the test classes you should see that 6 test cases are failing. Go to `BurgerShop.java` and implement the methods related to the command `remove` : `removeItem()` and the command `clear-cart` : `clearCart()`. The requirements are:

- The command `remove` removes the $i^{th}$ element in the cart (cart position starts from zero).
- If the provided position is not a valid position in the current cart, the application should print the following: "`!!!Error!!! Not a valid Cart item position`" (`MessagesCLI.NOT_VALID_CART_POSITION.printMessage()`).
- If the item at the $i^{th}$ position is removed, all subsequent items are shifted up by one.
- The command `clear-cart` removes all the elements in the cart.

For example:

```
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): L
burger-shop>add 2
burger-shop>add 5
burger-shop>add 1
burger-shop>cart
0 - Sprite (L): $5.00
1 - Vegan Burger: $18.50
2 - Black Truffle Burger: $27.50
3 - Avocado Burger: $17.00
Total: $68.00
burger-shop>remove 1
burger-shop>cart
0 - Sprite (L): $5.00
1 - Black Truffle Burger: $27.50
2 - Avocado Burger: $17.00
Total: $49.50
burger-shop>remove 2
burger-shop>cart
0 - Sprite (L): $5.00
1 - Black Truffle Burger: $27.50
Total: $32.50
burger-shop>clear-cart
burger-shop>cart
Cart is empty
burger-shop>
```

When all the test cases are passing (all green) and you are sure that you have correctly followed the requirements, you can move to Task 4! Remember to commit frequently.

## 5.4 Order command [Task 4: 3%]

Go at the beginning of `BurgerShopTestSuite` and uncomment `BurgerShopTestSuite.Task4Test.class` to enable the executions of Task 4 test cases.

```
    @RunWith(Suite.class)
    @SuiteClasses({ BurgerShopTestSuite.Task1Test.class, //
            BurgerShopTestSuite.Task2Test.class, //
            BurgerShopTestSuite.Task3Test.class, //
            BurgerShopTestSuite.Task4Test.class, //
            // BurgerShopTestSuite.YourTests.class //
    })
```

Run the test classes you should see that 13 test cases are failing. Go to `BurgerShop.java` and implement the methods related to the command `order` : `confirmOrder()`. The requirements are:

- If the cart is empty, the command `order` will return "!!!Error!!! Your Cart is empty!" (you can print the message like this: `MessagesCLI.ORDER_INVALID_CART_EMPTY.printMessage()`).
- The command `order` checks if the current cart contains **at least** a burger, a snack, and a drink that could be a combo—but is not. There could be a combo if the snack and drink have the same size. If this is the case the application show a warning: "WARNING! You added a Burger and Snacks and Drink of same size, you should change to a combo and save money!"(`MessagesCLI.MISSED_COMBO.printMessage()`)) and the order will not go through.
- If the order does not contain missed combos, the cart is shown, the order is confirmed, the cart is emptied, and it is printed the estimated waiting time (see `MessagesCLI.ESTIMATE_WAITING_TIME.getMessage`). The waiting time is calculated as follows:

  + 5 minutes for the first burger (+ 1 minute for each other burger).

  + 3 minutes for the first snack (+ 30 seconds for each other snack).

  + 45 seconds for the first drink (+ 15 seconds for each other drink).

  Note that a combo contains a burger, a snack, and a drink.

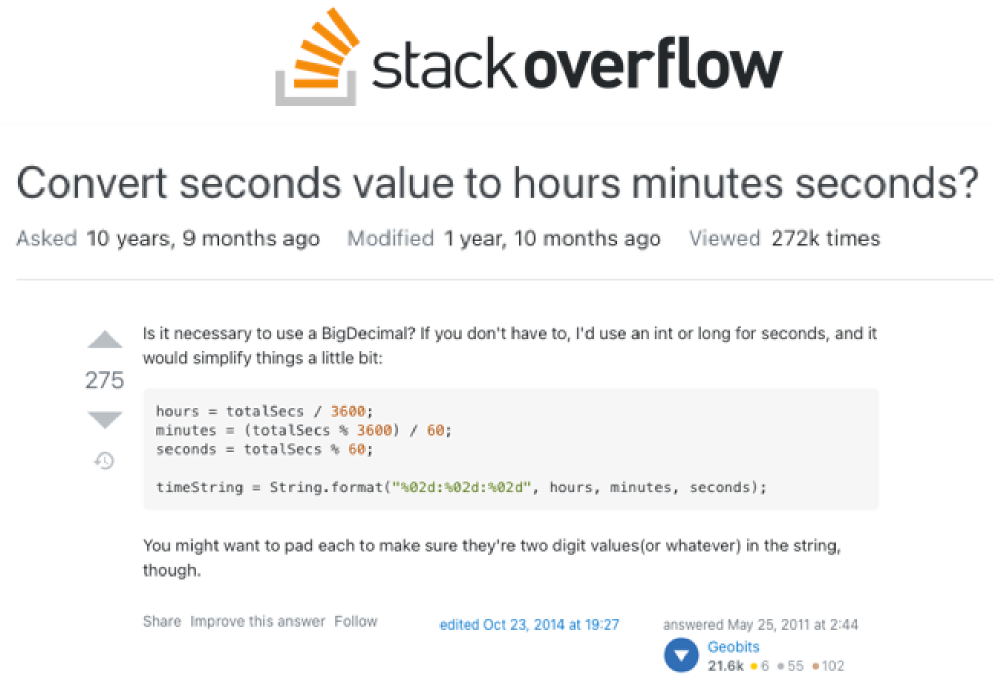
For example:

```
burger-shop>add 1
burger-shop>add 9
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): XL
burger-shop>add 12
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): XL
burger-shop>order
You added a Burger and Snacks and Drink of same size, you should change to a combo and save money!
burger-shop>clear-cart
burger-shop>add-combo
Choose a Burger ID : 1
Choose a Snack ID : 9
Choose a Drink ID : 12
Choose a size: 'M' or 'L' (+$3) or 'XL' (+$4): XL
burger-shop>order
0 - COMBO : (Avocado Burger, Onion Rings (XL), Coke (XL)): $29.00
Total: $29.00
Order confirmed! The estimated waiting time for preparing your order is: 0 hours 8 minutes 45 seconds
```

## 6 Hot Tip: StackOverflow.com

The website `www.StackOverflow.com` (SO) is the best friend of every software developer, no matter if you are a novice or an expert developer. Even expert developers visit SO many times per day! You are encouraged to do so.

For example, to print the waiting time in `xx hours xx minutes xx seconds`, you can take inspiration from this code snippet `https://stackoverflow.com/a/6118983`.



stackoverflow

Convert seconds value to hours minutes seconds?

Asked 10 years, 9 months ago Modified 1 year, 10 months ago Viewed 272k times

275

Is it necessary to use a BigDecimal? If you don't have to, I'd use an int or long for seconds, and it would simplify things a little bit:

```
hours = totalSecs / 3600;
minutes = (totalSecs % 3600) / 60;
seconds = totalSecs % 60;

timeString = String.format("%02d:%02d:%02d", hours, minutes, seconds);
```

You might want to pad each to make sure they're two digit values(or whatever) in the string, though.

Share Improve this answer Follow edited Oct 23, 2014 at 19:27 answered May 25, 2011 at 2:44 Geobits 21.6k ●6 ●55 ●102

Figure 4: Stackoverflow post showing how to get hours, minutes and seconds from the total number of seconds in Java `https://stackoverflow.com/a/6118983`

**Important**. While you are encouraged to use stackoverflow to find inspiration to resolve common and general programming problems, you should consider the following:

- If you reuse/adapt some code snippets from StackOverflow you have to put an attribution. For example, by adding the comment:

```
// code adapted from https://stackoverflow.com/a/6118983
hours = totalSecs / 3600;
minutes = (totalSecs % 3600) / 60;
seconds = totalSecs % 60;
```

  This is not only a good practice, but it also protects your code from being flagged as plagiarized.

- It is very important to always read carefully the description of the code and always try and test the code before using it. Especially in Stackoverflow.com, never just copy and paste code. Often there are multiple answers to the same question, you need to understand the one that is suitable in your case. Likely, you also need to adapt the solution. For example, the solution in the code snippet has to be modified to print the result as requested by the assignment `timeString = String.format("%02d:%02d:%02d", hours, minutes, seconds);`

## 7 Important: Rules and remarks

1. See the "Final Submission" section below for details you need to follow to ensure your code is submitted correctly.

2. Your code will be marked using a semi-automated setup. If you fail to follow the setup given, your code **will not be marked**. All submitted files must compile without requiring any editing. Use the provided tests and `maven wrapper` to ensure your code compiles and runs without errors. Any tests that run for longer than 10 seconds will be terminated and will be recorded as failed.
3. We encourage you to write high quality code by following good Java code conventions. In this assignment we will mark your code style. In our website you can find what are the aspects of code style that we will mark `https://softeng281.digitaledu.ac.nz/resources/code-conventions-best-practices/`.
4. You should not modify any of the classes inside the package `src/main/java/nz/ac/auckland/se281/a2/cli`. This includes also the methods and field visibility (do not change the visibility from `protected` to `public`). Also, you should not add classes inside such a package. All the classes that you will create must be placed inside the parent package `src/main/java/nz/ac/auckland/se281/a2`.
5. Although you may add more methods to the class `src/main/java/nz/ac/auckland/se281/a2/BurgerShop.java` you must leave unchanged the signature of the existing methods.
6. Do not move any existing code files to a new directory.
7. You cannot add and use external libraries. You can only use external classes that belong to the JDK (the import statement starts with **java**, for example `import java.util.ArrayList;`). **HINT:** this assignment can be correctly completed by only importing the external class `import java.util.ArrayList;`.
8. To check if you followed Object-Oriented principles, we will make sure that all classes you have created are used by the test cases. Creating classes that are not invoked by the application will not help you to reach the requirements (at least one abstract class, and at least three classes that extends a class that you have created).
9. Make sure that you do not modify the given test cases. Your code must still work with the original version of the `BurgerShopTestSuiteTestSuite.java` file.
10. Do not change the `pom.xml` file, and do not change any of the hidden files (those starting with ".").

## Marking Scheme

| | max mark | Note |
|---|---|---|
| Task 1 | 5% | |
| Task 2 | 3% | We have provided to you with **50% of the test cases** that we will use for marking (for each task). |
| Task 3 | 2% | |
| Task 4 | 3% | |
| OO-design | 5% | Create (and use) at least one abstract class, and at least three (sub)classes that extend one of the classes that you created. |
| Code style | 2% | Visit `https://softeng281.digitaledu.ac.nz/resources/code-conventions-best-practices/`to see what are the conventions that we will mark against in this assignment. |
| **Total** | **20%** | % of your final grade |

## Frequent Submissions (GitHub)

**Make sure that the final version you submit on Canvas is identical to the one in your GitHub repo.**

You **must** use GitHub as version control for all assignments in this course, starting with this assignment. To get the starting code for this assignment, you must accept the GitHub Classroom invitation that will be shared with you. This will then clone the code into a **private** GitHub repository for you to use. This repository must remain private during the lifetime of the assignment. When you accept the GitHub assignment, you can clone the repository to your own computer.

As you work on your assignment, **you must make frequent git commits**. If you only commit your final code, **it will look suspicious and you will be penalised**. In the case of this assignment, you should aim to commit your changes to GitHub every time you pass a few test cases (at the very least, once for every task completed). You can check your commits in your GitHub account to make sure the frequent commits are being recorded. **Using GitHub to frequently commit your changes is mandatory in this course.** In addition to teaching you to use a useful software development skill (version control with git), it will also protect you two very important ways:

1. Should something terrible happen to your laptop, or your files get corrupted, or you submitted the wrong files to Canvas, or you submitted late, etc, then you are protected as the commits in GitHub verify the progress in your assignment (i.e. what you did and when), and
2. If your final code submission looks suspiciously similar to someone else (see academic honesty section below), then GitHub provides a track record demonstrating how you progressed the assignment during that period.

Together, GitHub is there to help you.

## Final Submission (GitHub and Canvas)

In addition to the frequent GitHub submissions, you will **also need submit via Canvas**. You can find the instructions on how to submit the assignment on Canvas in the related FAQ on the course website `https://softeng281.digitaledu.ac.nz/resources/faqs/#submit`.

**You must double check that you have uploaded the correct code for marking!** There will be no exceptions if you accidentally submitted the wrong files, regardless of whether you can prove you did not modify them since the deadline. No exceptions. Get into the habit of downloading them again, and double-checking all is there.

## Academic honesty

- The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone for help (struggling a little with it will help you learn, especially if you end up solving it). If you still need help, check on Canvas (if it is about interpreting the assignment specifications) or ask in the Lab help clinics (if you would like more personal help with Java). Under no circumstances should you take or pay for an electronic copy of someone else's work.
- **All submitted code will be checked using software similarity tools.** Submissions with suspicious similarity will result in an Investigative Meeting and will be forwarded to the Disciplinary Committee.
- Penalties for copying will be severe – to avoid being caught copying, don't do it.
- You must not attempt to tamper with your GitHub commits. This means all commits must be preserved, including the one that is automatically created by GitHub Classroom. Disregarding this will result in major penalties.
- To ensure you are not identified as cheating you should follow these points:
  - Always do individual assignments by yourself.
  - Never show or give another person your code.
  - Keep your Repl workspace private, and do not share your Repl with anyone.
  - Never put your code in a public place (e.g. Reddit, public GitHub repository, forums, your website).
  - Never leave your computer unattended. You are responsible for the security of your account.

  - Ensure you always remove your USB flash drive from the computer before you log off.
  - Frequently commit your code to GitHub. This provides a track record of your work and will allow the teaching team to follow your footsteps as you completed your assignment. If you do not frequently commit your code, it will look suspicious.

## Late submissions

Late submissions will incur the following penalties:

- 15% penalty for zero to 24 hours late
- 30% penalty for 25 to 48 hours late
- 100% penalty for over 48 hours late (Canvas submission automatically closes)