1. Instance and class variables

Count that does not work

Consider the following Java code:

Product.java

```java
public class Product {
  private int id;
  private double price;
  private String name;
  private int quantity;

  public Product(int id, double price, String name){
    this.id = id;
    this.price = price;
    this.name = name;
    this.quantity ++;
  }
  public void applySaleDiscount(double percentage){
    this.price = this.price - ((percentage/100) * this.price);
  }

  public void addToShoppingCart(){
    System.out.println(this.name + " has been added to the shopping cart.");
  }

  public int getTotalQuantity(){
    return this.quantity;
  }

  public void getSellableStatus(){
    System.out.println("This product is sellable");
  }

  public String toString(){
    return "Product info:\n+Id: " + this.id + "\t" + "name: " + this.name +
      "\tPrice: SR" + this.price;
  }
}
```

App.java

```java
public class App{

  public static void main(String[]args){
    Product p1 = new Product(6745, 5.50, "Penne Pasta");
    Product p2 = new Product(8853, 6.50, "Spaghetti Pasta");
    Product p3 = new Product(2106, 4.50, "Linguine Pasta");
    System.out.println("Total Quantity: " + p3.getTotalQuantity());
  }
}
```

We get the following output:

```
Total Quantity: 1
```

Question: How would you fix this code to print the correct total quantity, 3?

Answer: by creating a Class Variables (static) which Hold values that are shared by all objects or instances of the class.

```
private static int quantity;
```

```
public static int getTotalQuantity() {
    return Product.quantity;
}
```

2. Testing

A unit test is a piece of code that executes a specific functionality in the code to be tested and ensures it behaves as intended. This will help you in the future when additional changes are made to the code.

Question: Complete the following unit test for the previous problem to ensure that the code always returns the correct quantity.

Answer: `assertEquals(Product.getTotalQuantity(), 3);`

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class ProductTest
{
  @Test
  public void shouldCountQuantity()
  {
    Product p1 = new Product(6745, 5.50, "Penne Pasta");
    Product p2 = new Product(8853, 6.50, "Spaghetti Pasta");
    Product p3 = new Product(2106, 4.50, "Linguine Pasta");
    assertEquals(p3.getTotalQuantity(), 3);
  }

}
```

```
public class ProductTest {

    @Test
    public void testSomeMethod() {
        Product p1 = new Product(6745, 5.50, "Penne Pasta");
        Product p2 = new Product(8853, 6.50, "Spaghetti Pasta");
        Product p3 = new Product(2106, 4.50, "Linguine Pasta");
        assertEquals(Product.getTotalQuantity(), 3);
    }

}
```

3. Inheritance

Extend the product class by adding the following two derived classes (a.k.a child classes and subclasses):

Question: The Product class should not be instantiated directly as done in the first line of the main method. Change the Product class, so only concrete classes should be instantiated. What changes would you make?

Answer: make the Product class abstract by adding the abstract keyword to its class declaration

```
public abstract class Product {
```

4. Polymorphism “Many Forms”

Question: Change the main class to utilize the use of Polymorphism and iterate through an array of Products using the enhanced for statement (a.k.a For-Each Loop)?

Answer:

```
public static void main(String[] args) {
    FoodProduct p4 = new FoodProduct(3452, 10.0, "Cheddar Cheese", LocalDate.parse("2022-06-07"));
    ElectricProduct p5 = new ElectricProduct(4875, 30.0, "Extension cord", "220v");

    Product[] products = {p4, p5};
    for (Product p : products) {
        System.out.println(p.toString());
    }

}
```

5. Controlling Changes

Question: What would you do to prevent subclasses from overriding the addToShoppingCart() method of the Product class without changing its visibility?

Answer:

declare the **addToShoppingCart()** method as final.

```
public final void addToShoppingCart() {
    System.out.println(this.name + " has been added to the shopping cart.");
}
```

6. Abstraction

We decided to add a feature to the Product class by adding Order information:

```
public class Product {
  private int id;
  private double price;
  private String name;
  private int quantity;
  private int orderId;
  private String orderStatus;

  public Product(int id, double price, String name, int orderId, String orderStatus){
    this.id = id;
    this.price = price;
    this.name = name;
    this.orderId = orderId;
    this.orderStatus = "created"
    this.quantity ++;
  }
```

Question: Why is this considered bad? How would you fix it?
Answer: this is considered bad because its poor abstraction, means we have a class that is not focused on a single task. Like the order information should be moved to another class and have a separate association between **Order** And **Product**

```
public class Order {

    private int orderId;
    private String orderStatus;
    private List<Product> products;

    public Order(int orderId, String orderStatus, List<Product> products) {
        this.orderId = orderId;
        this.orderStatus = orderStatus;
        this.products = products;
    }
```

7. Encapsulation

We decided to add a feature to the Product class by adding the product's weight information. We decided to make it public, so any subclass can change it easily.

```
public class Product {
  private int id;
  private double price;
  private String name;
  public double weight;

  public Product(int id, double price, String name, int orderId, String orderStatus){
    this.id = id;
    this.price = price;
    this.name = name;
    this.orderId = orderId;
    this.orderStatus = "created"
    this.quantity ++;
  }
```

Question: Why is this considered bad? How would you fix it?

Answer: it breaks the principle of encapsulation, we are allowing any class to modify the state of the **Product** object directly, which can lead to unexpected behavior and make the code harder to maintain. We can fix this by making a public method **setter.**

```
public double getWeight() {
    return weight;
}

public void setWeight(float weight) {
    this.weight = weight;
}
```

Here is the source code for the lab1.

https://github.com/Malek-O/lab1