UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

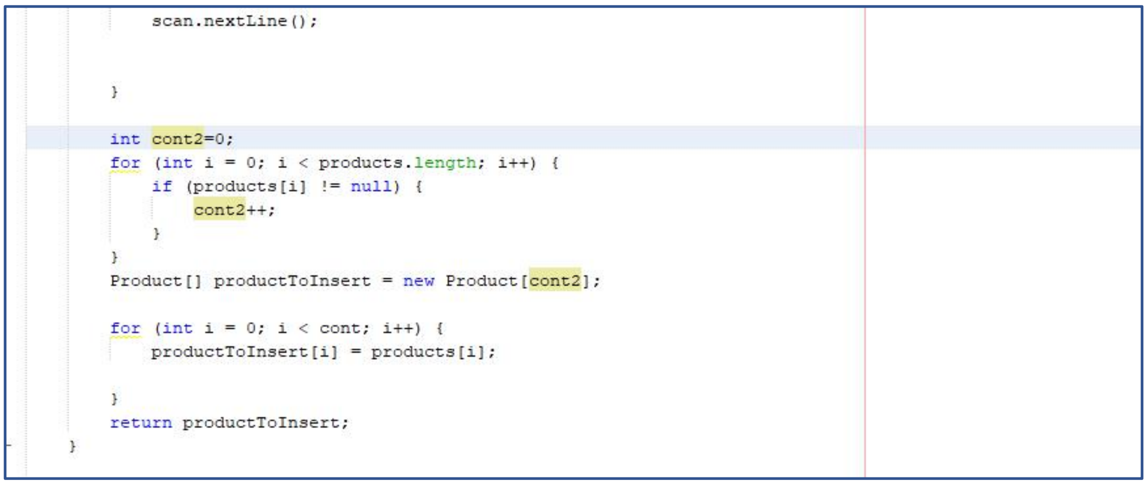
- Object Oriented Programming -

< SMELL CODE >

TEAM #3

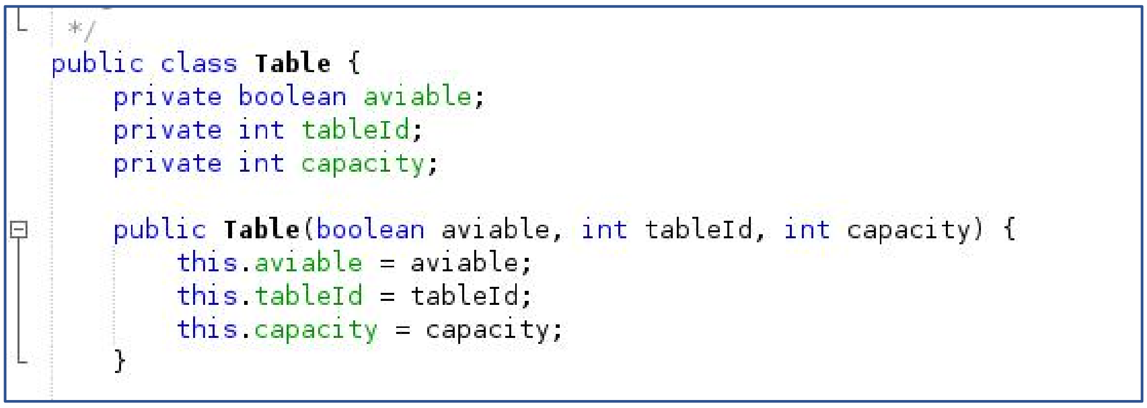
Inspection of Team #2

1. UNCOMMUNICATIVE NAMES



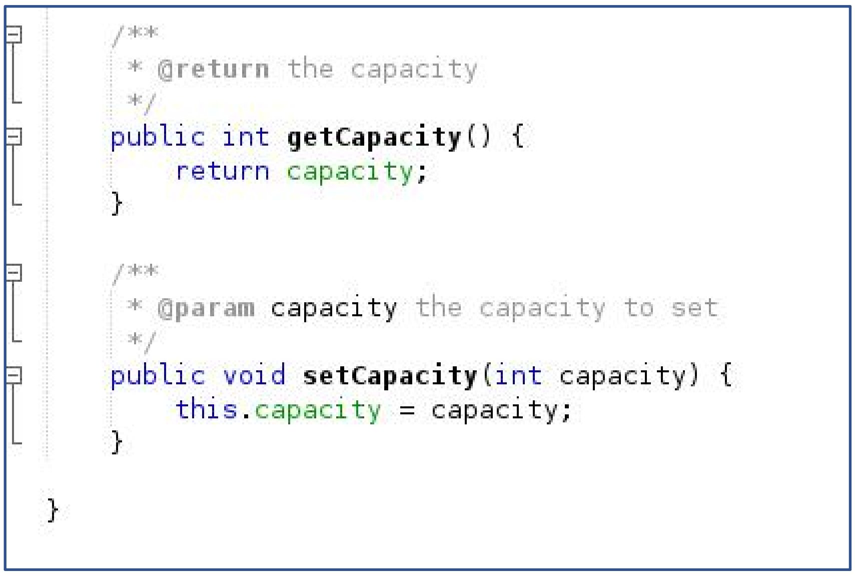
```
            scan.nextLine();


        }

        int cont2=0;
        for (int i = 0; i < products.length; i++) {
            if (products[i] != null) {
                cont2++;
            }
        }
        Product[] productToInsert = new Product[cont2];

        for (int i = 0; i < cont; i++) {
            productToInsert[i] = products[i];

        }
        return productToInsert;
    }
```

// The variable cont2 is a name that does not communicate what it does.



```
 */
public class Table {
    private boolean aviable;
    private int tableId;
    private int capacity;

    public Table(boolean aviable, int tableId, int capacity) {
        this.aviable = aviable;
        this.tableId = tableId;
        this.capacity = capacity;
    }
```

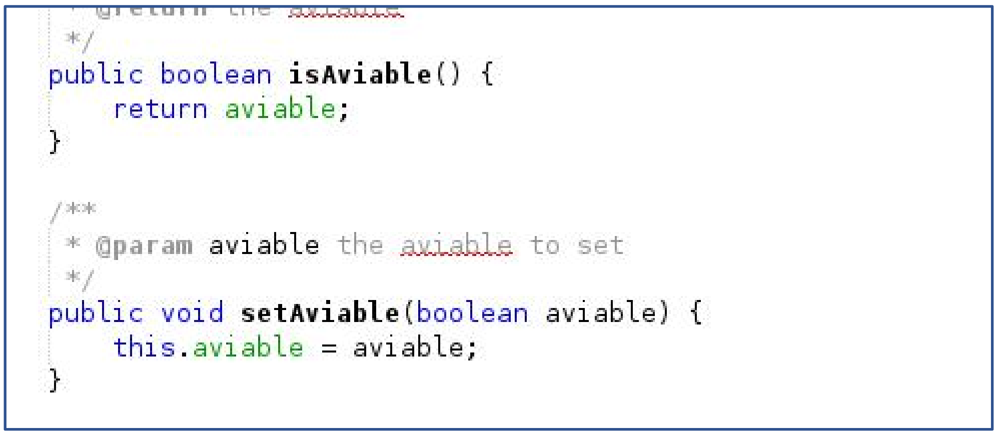
// This class name does not give any information about its utility.



```
    /**
     * @return the capacity
     */
    public int getCapacity() {
        return capacity;
    }

    /**
     * @param capacity the capacity to set
     */
    public void setCapacity(int capacity) {
        this.capacity = capacity;
    }

}
```

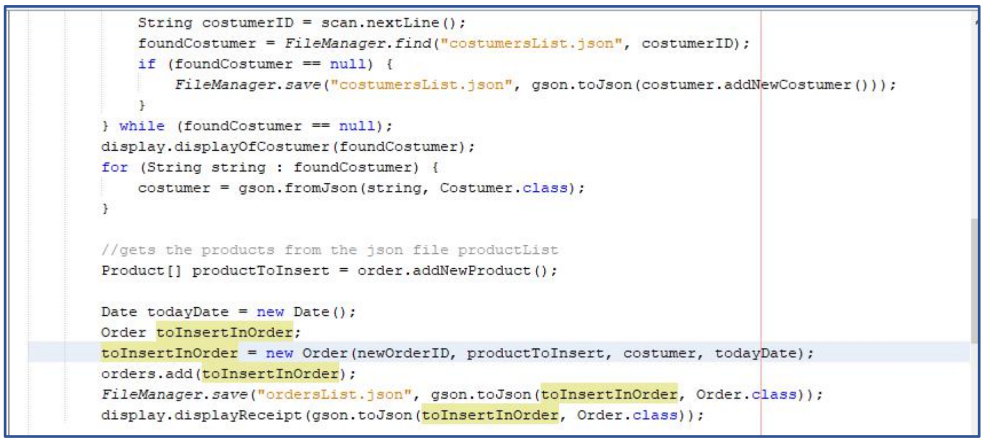
// This names don’t specify the things capacity that get or set.

2. INCONSISTENT NAMES



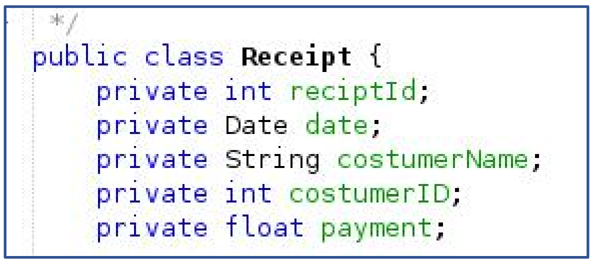
```
 */
public boolean isAviable() {
    return aviable;
}

/**
 * @param aviable the aviable to set
 */
public void setAviable(boolean aviable) {
    this.aviable = aviable;
}
```

// The correct form of write is AVALIABLE, not AVIABLE.



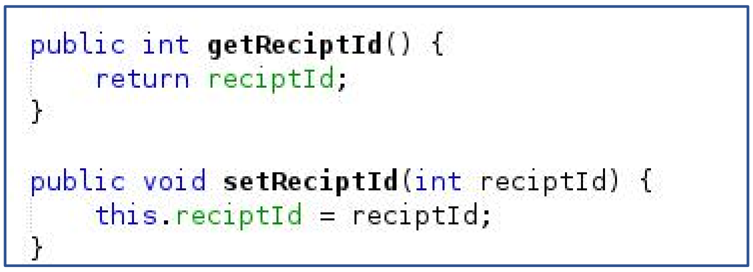
```
    String costumerID = scan.nextLine();
    foundCostumer = FileManager.find("costumersList.json", costumerID);
    if (foundCostumer == null) {
        FileManager.save("costumersList.json", gson.toJson(costumer.addNewCostumer()));
    }
} while (foundCostumer == null);
display.displayOfCostumer(foundCostumer);
for (String string : foundCostumer) {
    costumer = gson.fromJson(string, Costumer.class);
}

//gets the products from the json file productList
Product[] productToInsert = order.addNewProduct();

Date todayDate = new Date();
Order toInsertInOrder;
toInsertInOrder = new Order(newOrderID, productToInsert, costumer, todayDate);
orders.add(toInsertInOrder);
FileManager.save("ordersList.json", gson.toJson(toInsertInOrder, Order.class));
display.displayReceipt(gson.toJson(toInsertInOrder, Order.class));
```

// The name for the object must be a noun. And in this case the name is a verb.



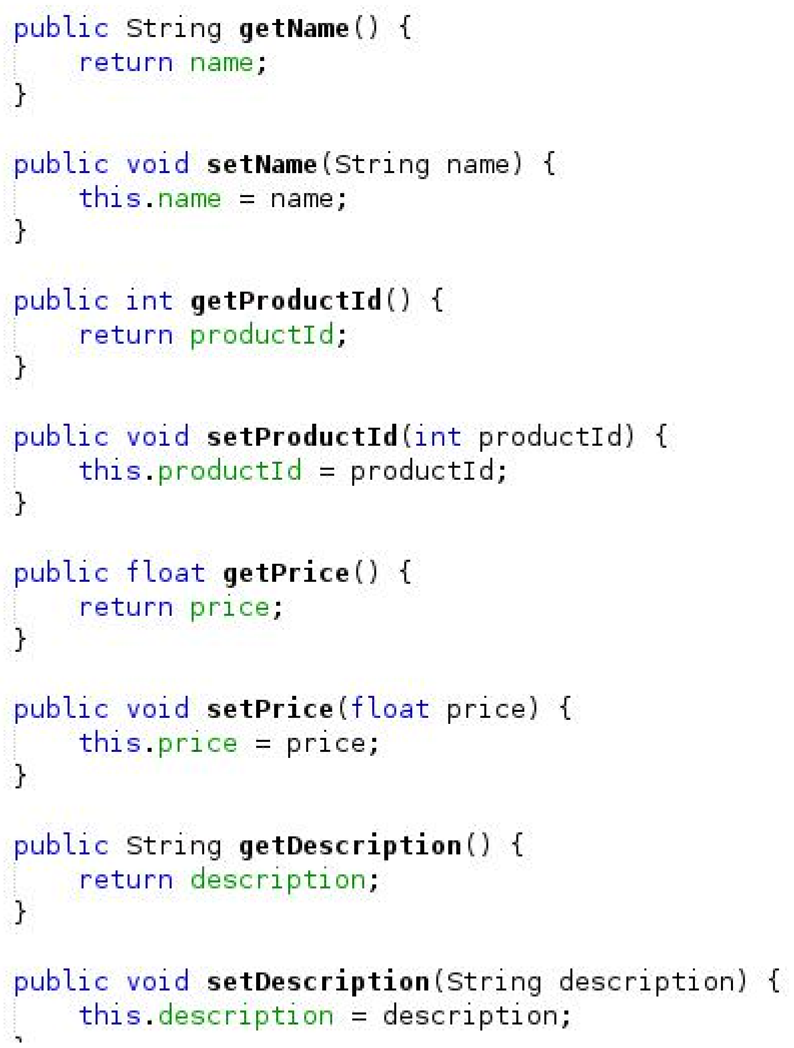
```
 */
public class Receipt {
    private int reciptId;
    private Date date;
    private String costumerName;
    private int costumerID;
    private float payment;
```

// reciptID variable name is not consistent with the class name. Also, Receipt class should be called Bill.



```
public int getReciptId() {
    return reciptId;
}

public void setReciptId(int reciptId) {
    this.reciptId = reciptId;
}
```

// This methods names are not consistent with the class name.. ReciptId - Receipt



```java
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getProductId() {
    return productId;
}

public void setProductId(int productId) {
    this.productId = productId;
}

public float getPrice() {
    return price;
}

public void setPrice(float price) {
    this.price = price;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
```

// All methods should specify its class in name, (EX: getProductDescription, setProductPrice, etc.) or contrary, method setProductId should be called only setID.

3. TYPES EMBEDDED IN NAMES.

4. LONG METHODS

5. DUPLICATE CODE

6. LONG MESSAGE CHAINS

7. CLASS EXPLOSION

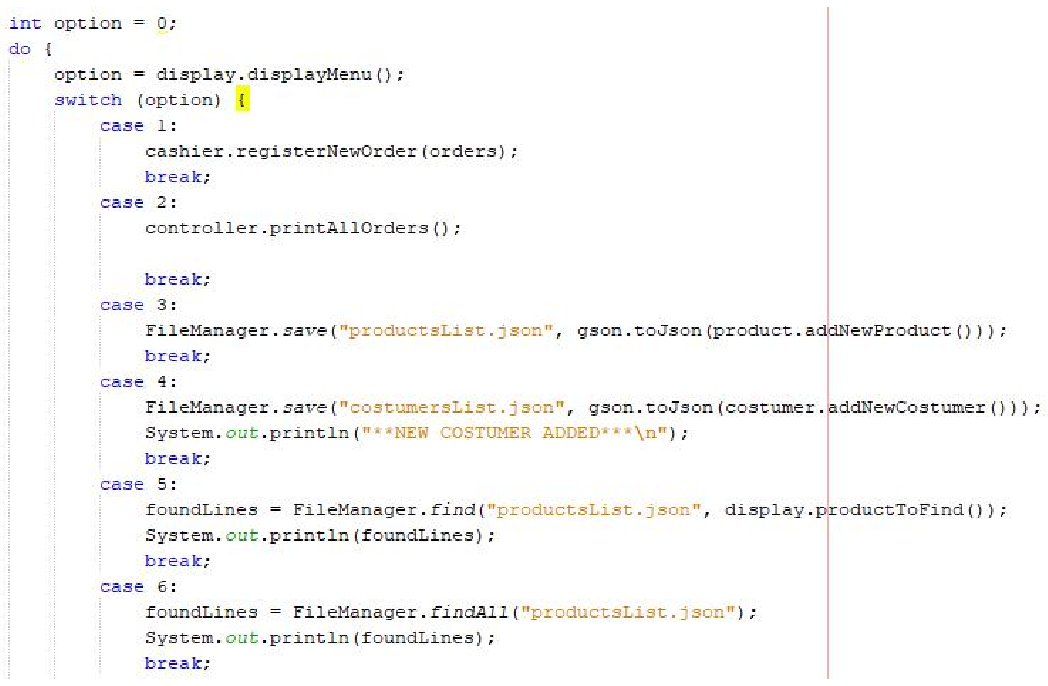
8. LARGE MESSAGE CHAINS

9. LARGE CLASSES

10. CONDITIONAL COMPLEXITY

11. ODDBALL SOLUTION

12. REDUNDANT OR MEANINGLESS COMMENTS
13. DEAD CODE
14. SPECULATIVE GENERALITY
15. TEMPORARY FIELD
16. REFUSED BEQUEST
17. INAPPROPIATE INTIMACY
18. FEATURE ENVY



```
int option = 0;
do {
    option = display.displayMenu();
    switch (option) {
        case 1:
            cashier.registerNewOrder(orders);
            break;
        case 2:
            controller.printAllOrders();

            break;
        case 3:
            FileManager.save("productsList.json", gson.toJson(product.addNewProduct()));
            break;
        case 4:
            FileManager.save("costumersList.json", gson.toJson(costumer.addNewCostumer()));
            System.out.println("**NEW COSTUMER ADDED***\n");
            break;
        case 5:
            foundLines = FileManager.find("productsList.json", display.productToFind());
            System.out.println(foundLines);
            break;
        case 6:
            foundLines = FileManager.findAll("productsList.json");
            System.out.println(foundLines);
            break;
```

```
// There should be no file manager code in main.
```