João Marcos 2264501

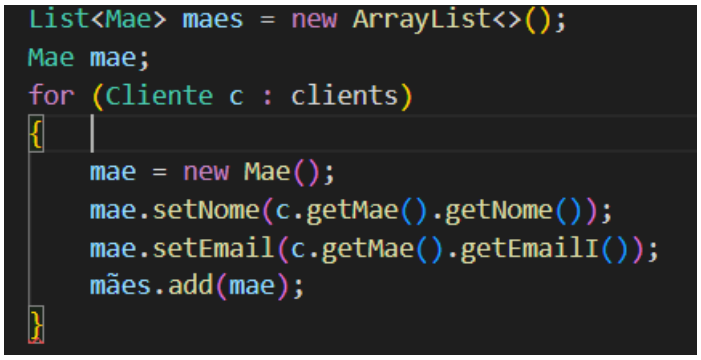
1. Nos dias das mães, uma loja resolveu coletar o nome das mães de seus clientes e seus respectivos e-mails para mandar cupons de presentes da loja. Parte do código era assim:

```
Mae m = new Mae();
List<Mae> l = new ArrayList<>();

for (Cliente c : clientes)
{
   m.setNome(c.getMae().getNome());
   m.setEmail(c.getMae().getEmail());

  l.add(m);
}
```

Na segunda-feira, após o dia das mães, uma senhora chegou com mais de mil cupons para trocar, e para surpresa da loja, todos eram válidos, com o nome dela. Como era de se esperar, entrou o procon, a justiça e advogados no meio, e a loja saiu perdendo dezenas de milhares de reais. Estranhamente, nenhuma outra pessoa apareceu na loja com cupons. O que aconteceu? Corrija e refatore o trecho de código aqui mesmo na forma de texto.



2. Analise as classes FileLogger e DbLogger

```
public class FileLogger
{
    0 references
    public bool IsLogMessageValid(string message)
    {
        //Check if message contains sensitive data such as SSN, Credit Card number, etc.
        return (!string.IsNullOrEmpty(message));
    }
    0 references
    public bool DoLog(string message)
    {
        //Code to log to a text file
        return true;
    }
}

public class DbLogger
{
    0 references
    public bool IsLogMessageValid(string message)
    {
        //Check if message contains sensitive data such as SSN, Credit Card number, etc.
        return (!string.IsNullOrEmpty(message));
    }
    0 references
    public bool DoLog(string message)
    {
        //Code to log to a database
        return true;
    }
}
```

Identifique os pontos que podem ser considerados “code smell” e proponha a refatoração adequada. Monte o código em um editor java (Eclipse, Notepad++, VSCode,...), não implemente os “códigos” que estão em forma de comentário, deixe como está, apenas faça a refatoração necessária, copie e cole aqui mantendo a formatação.

```
public class Logger {
    public boolean isLogMessageValid(String message) {
        if (String.IsEmptyOrNull(message)) {
            return false;
        } else {
            return true;
        }
    }

    public boolean DoLog(String Message) {
        return true;
    }
    public Logger() {
    }
}


public class DBLogger extends Logger{
```

```
    public DBLogger() {
        super();
    }
}


public class FileLogger extends Logger{
    public FileLogger() {
        super();
    }
}
```

3. O que há para refatorar na classe DateUtil? Proponha uma adequação.

```java
class DateUtil {
    boolean isAfter(int year1, int month1, int day1, int year2, int month2, int day2) {
        // implementation
    }

    int differenceInDays(int year1, int month1, int day1, int year2, int month2, int day2) {
        // implementation
    }

    // other date methods
}
```

Codifique em Java, copie e cole aqui mantendo a formatação.

```java
class DateUtil
{
    boolean isAfter(Date date1, Date date2){

    }

    int differenceInDays(Date date1, Date date2){

    }

}
```

4. Como melhorar o trecho de código a seguir?

```
// amount
double a = order.getAmount();
// discount factor
double b = 1;
if (a > 10) {
    b = 0.9;
}
// discounted price
double c = product.getPrice() * b;
// order sum price
double d = a * c;
```

Refatore aqui mesmo em forma de texto.

Refatore aqui mesmo em forma de texto.

```
double amount = order.getAmount();
double price = product.getPrice();
double discount = 0;

if(amount>10){
    discount = price * 0.9 ;
}

double finalPrice = (price - discount) * amount;
```

5. Considere o código a seguir

```
var miles = 0.0;

if (car.HasFuel)
{
  if (car.EngineWorks)
  {
     var startingMiles = car.Miles;
     car.Drive();
     var endingMiles = car.Miles;
     miles = endingMiles - startingMiles;
  }
}

return miles;
```

proponha uma refatoração para melhorar a legibilidade, escreva a nova versão do código aqui mesmo em forma de texto.

```
var miles  = 0.0;

if(!car.HasFuel() || !car.EngineWorks()){

   return miles;

}else{

   var startingMiles = car.Miles;

   car.Drive();

   var endingMiles = car.Miles;


   return miles = endingMiles - startingMiles;

}

}
```

6. Identifique os problemas no código a seguir e proponha uma solução via refatoração. Explique as suas modificações.

```
class Repository {
    Entity findById(long id) {
        // implementation
    }
}

class Service {
    Repository repository;

    Repository getRepository() {
        return repository;
    }
}

class Context {
    Service service;

    void useCase() {
        // the following is a message chain
        Entity entity = service.getRepository().findById(1);
        // using entity
    }
}
```

Copie e cole o código aqui mantendo a formataçãoclass Repository {

Entity findById(long id){}

static Repository getRepository(){

return new Repository;

}

}

class Context {

void useCase() {

Entity entity = Repository.getRepository().findById(1);

}

}

7. Considere de GildedRose class (code.zip), o método updateQuality tem mais de 75 linhas. Ao executar o teste a saída é

```
OMGHAI!
-------- day 0 --------
name, sellIn, quality
+5 Dexterity Vest, 10, 20
Aged Brie, 2, 0
Elixir of the Mongoose, 5, 7
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 15, 20
Backstage passes to a TAFKAL80ETC concert, 10, 49
Backstage passes to a TAFKAL80ETC concert, 5, 49
Conjured Mana Cake, 3, 6

-------- day 1 --------
name, sellIn, quality
+5 Dexterity Vest, 9, 19
Aged Brie, 1, 1
Elixir of the Mongoose, 4, 6
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 14, 21
Backstage passes to a TAFKAL80ETC concert, 9, 50
Backstage passes to a TAFKAL80ETC concert, 4, 50
Conjured Mana Cake, 2, 5
```

   a. Identifique quaisquer trechos de código repetidos que possam ser extraídos para métodos privados. Realize:
      i. **private void** decrementQuality ...
      ii. **private void** incrementQuality ...
   b. Execute o teste e confira a saída atual com a original
   c. Considere extrair o código longo na primeira estrutura if/else para um método privado. Realize:
      i. **private void** UpdateQualityForItemsThatAgeWell...

d. Execute o teste e confira a saída atual com a original
e. Considere extrair um segundo bloco de código longo de estrutura if/else para o método **private void** UpdateQualityForExpiredItems
f. Execute o teste e confira a saída atual com a original
g. Analise o código do método UpdateQualityForItemsThatAgeWell e refatore.
h. Execute o teste e confira a saída atual com a original
i. Comente sobre esta questão em 3 a 5 linhas.

O código dado é um exemplo claro de como um código mal escrito dificulta a interpretação e a manutenção. O que mais me chamou a atenção foi a quantidade de código repetido e o tanto de if/else aninhados e um mais confuso que o outro.

```
public class GildedRose {

Item[] items;

public GildedRose(Item[] items) {

this.items = items;

}

public void updateQuality() {

for (int i = 0; i < items.length; i++) {

UpdateQualityForItemsThatAgeWell(items[i]);

if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {

items[i].sellIn = items[i].sellIn - 1;

}

UpdateQualityForExpiredItems(items[i]);

}

}

private void decrementQuality(Item item) {

item.quality -= 1;

}

private void incrementQuality(Item item) {

item.quality += 1;

}
```

```
private void UpdateQualityForExpiredItems(Item item) {
if (item.sellIn < 0) {
if (!item.name.equals("Aged Brie")) {
if (!item.name.equals("Backstage passes to a TAFKAL80ETC concert")) {
if (item.quality > 0) {
if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
decrementQuality(item);
}
}
} else {
item.quality = item.quality - item.quality;
}
} else {
if (item.quality < 50) {
incrementQuality(item);
}
}
}
}
private void UpdateQualityForItemsThatAgeWell(Item item) {
if (isNotAgedBrieOrBackstagePass(item)) {
decrementQualityIfItemQualityGreaterThanZero(item);
} else {
incrementQualityIfItemQualityLessThanFifty(item);
if (isBackstagePass(item)) {
incrementQualityIfSellInLessThanEleven(item);
incrementQualityIfSellInLessThanSix(item);
}
```

```
}

}

private boolean isNotAgedBrieOrBackstagePass(Item item) {

return !item.name.equals("Aged Brie") && !item.name.equals("Backstage
passes to a TAFKAL80ETC concert");

}

private void decrementQualityIfItemQualityGreaterThanZero(Item item) {

if (item.quality > 0 && !isSulfuras(item)) {

decrementQuality(item);

}

}

private void incrementQualityIfItemQualityLessThanFifty(Item item) {

if (item.quality < 50) {

incrementQuality(item);

}

}

private boolean isBackstagePass(Item item) {

return item.name.equals("Backstage passes to a TAFKAL80ETC concert");

}

private void incrementQualityIfSellInLessThanEleven(Item item) {

if (item.sellIn < 11) {

incrementQualityIfItemQualityLessThanFifty(item);

}

}

private void incrementQualityIfSellInLessThanSix(Item item) {

if (item.sellIn < 6) {

incrementQualityIfItemQualityLessThanFifty(item);

}
```

```
}

private boolean isSulfuras(Item item) {

return item.name.equals("Sulfuras, Hand of Ragnaros");

}

}
```