André Luis Quiosi

Diego Gomes

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.

GeradorCupomCliente();

public <List> GeradorCupomCliente(){

List<Mae> listaMae = new ArrayList<>();
for (Cliente cliente : clientes){
listaMae.add(cliente.getMae());
}

return listaMae;
}

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 abstract class Logger {
    public bool IsLogMessageValid(String message){
        return (!isString.IsNullOrEmpty(message));
    }

    public abstract bool DoLog(string message);
}

public class FileLogger : Logger {
    public override bool DoLog(string message){
        // Implementação específica do FileLogger
```

```
        return true;
    }
}

public class DbLogger : Logger {
    public override bool DoLog(string message){
        // Implementação específica do DbLogger
        return true;
    }
}
```

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.

```
class DateUtil {
    public bool IsAfter(DateTime date1, DateTime date2) {
        return date1 > date2;
    }

    public int DifferenceInDays(DateTime date1, DateTime date2) {
        return (date1 - date2).Days;
    }
}
```

4. Como melhorar o trecho de código a seguir?

```java
// 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.

```
double orderAmount = order.getAmount();
double discount = orderAmount > 10 ? 0.9 : 1.0;
double productPrice = product.getPrice();
double discountedPrice = productPrice * discount;
double totalPrice = orderAmount * discountedPrice;
```

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.

```
final double MILES_PER_GALLON = 20.0;

public double getMilesDriven(Car car) {

   if (!car.hasFuel() || !car.engineWorks()) {
   return 0.0;
   }

double startingMiles = car.getMiles();
car.drive();
double endingMiles = car.getMiles();
return endingMiles - startingMiles;
}

double miles = getMilesDriven(car) ?? 0.0;
double gallons = miles / MILES_PER_GALLON;
```

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 formatação

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


class Service {
   Repository repository;


   public Service(Repository repository) {
      this.repository = repository;
   }
```

```
}

class Context {
    Service service;

    public Context(Service service) {
        this.service = service;
    }

    void useCase() {
        Entity entity = service.repository.findById(1);
        // fazer algo com a entidade encontrada
    }
}
```

Com essas modificações, o código se torna mais simples e mais legível. A classe Repository agora tem um retorno definido para o método findById, e a classe Service recebe uma instância de Repository em seu construtor, em vez de acessá-la por meio de um método intermediário. Na classe Context, adicionamos um construtor que recebe uma instância de Service, tornando o código mais modular e mais fácil de testar. Por fim, acessamos o atributo repository diretamente no método useCase, tornando o código mais legível e mais fácil de entender.

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.

Respostas:

A) O trecho de código que decrementa a qualidade do item:

```
if (items[i].quality > 0)
{
    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros"))
    {
        items[i].quality = items[i].quality - 1;
    }
}
```

Onde se consegue remover o método privado private void decrementQuality(Item item).

Minha incremento.

```
if (items[i].quality < 50)
{
    items[i].quality = items[i].quality + 1;
}
```

A extração do segundo bloco de código longo para o método UpdateQualityForExpiredItems torna o código mais legível e fácil de entender, seguindo o princípio da responsabilidade única. Além disso, a refatoração do método UpdateQualityForItemsThatAgeWell permite a remoção de várias declarações desnecessárias de variáveis e simplifica a lógica, tornando o código mais fácil de manter e entender. A execução dos

testes após a refatoração confirma que a funcionalidade original foi mantida, enquanto o código se tornou mais limpo e eficiente.