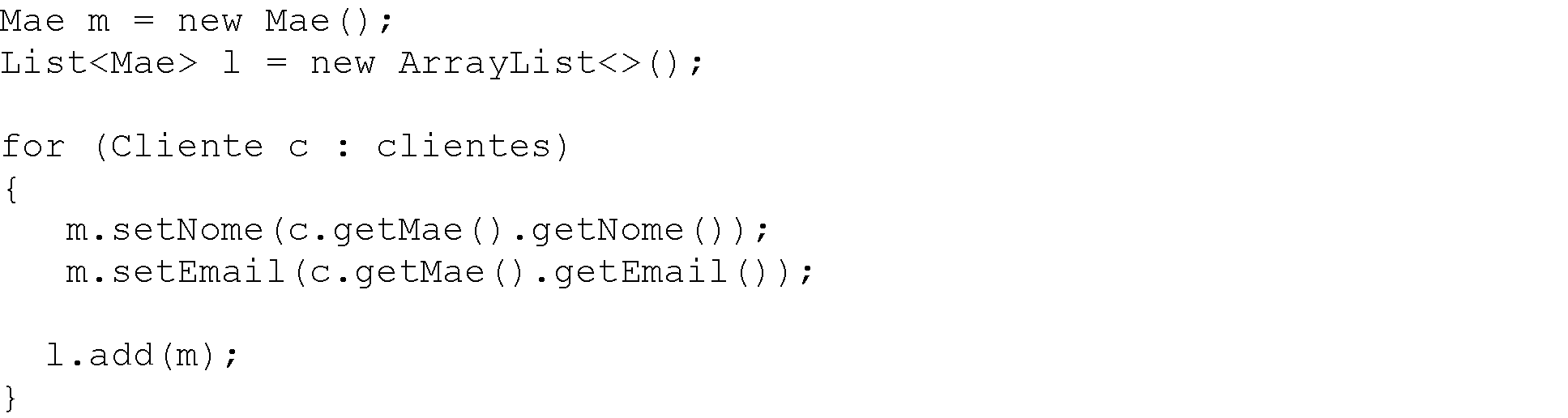
**André Luis Quiosi**

**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:

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());

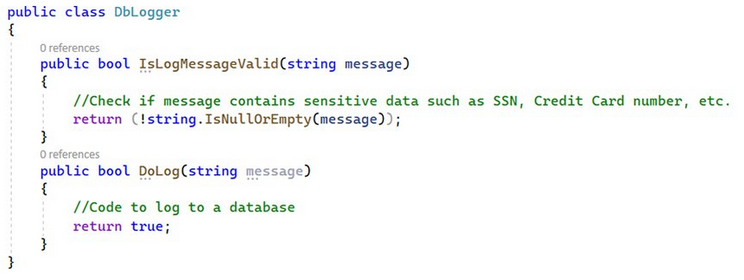
}

return listaMae;

}

1. Analise as classes FileLogger e DbLogger





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){

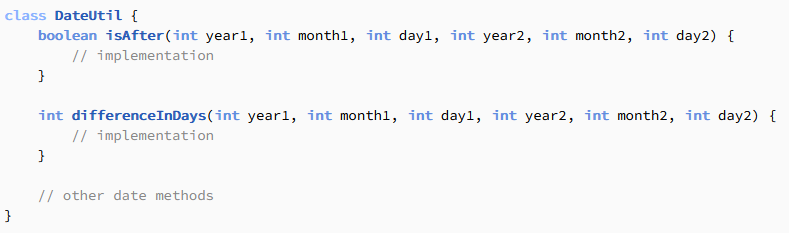
// Implementação específica do DbLogger

return true;

}

}

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



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

class DateUtil {

public bool IsAfter(DateTime date1, DateTime date2) {

return date1 > date2;

}

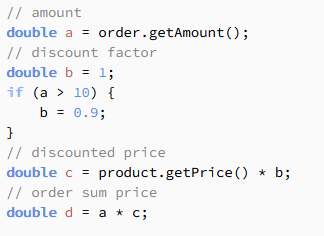
public int DifferenceInDays(DateTime date1, DateTime date2) {

return (date1 - date2).Days;

}

}

1. Como melhorar o trecho de código a seguir?



Refatore aqui mesmo em forma de texto.

double orderAmount = order.getAmount();

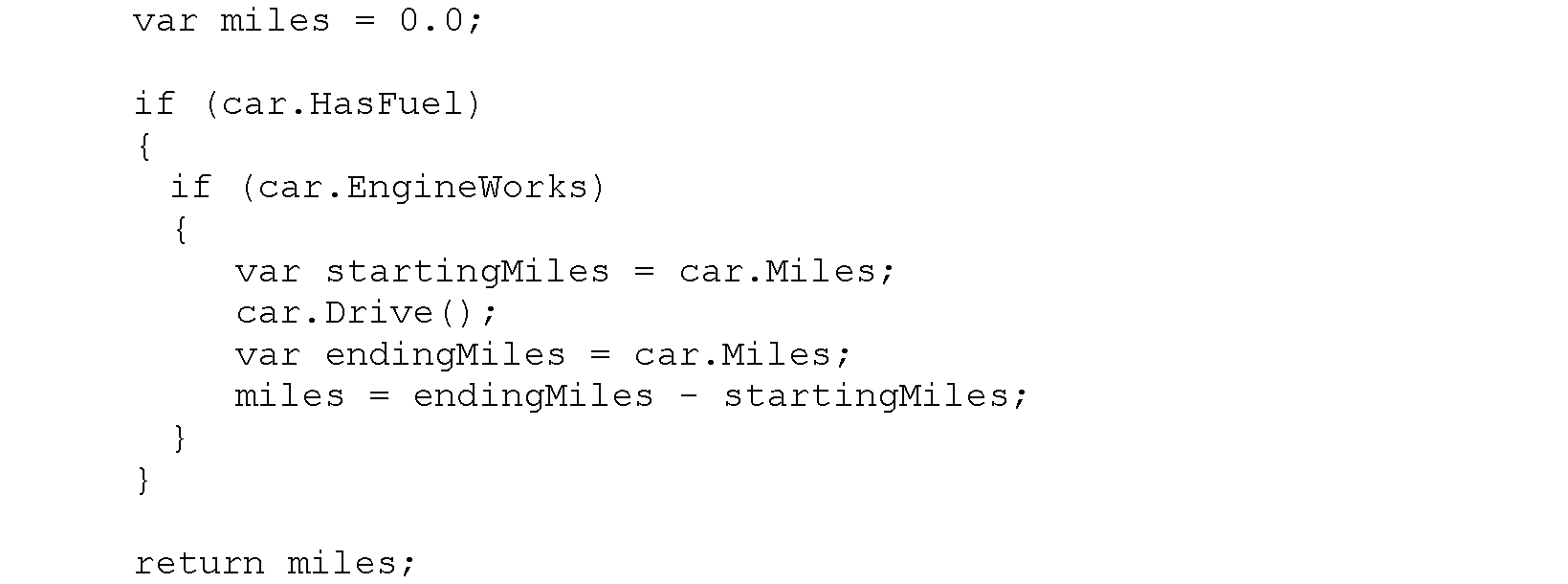
double discount = orderAmount > 10 ? 0.9 : 1.0;

double productPrice = product.getPrice();

double discountedPrice = productPrice \* discount;

double totalPrice = orderAmount \* discountedPrice;

1. Considere o código a seguir



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();

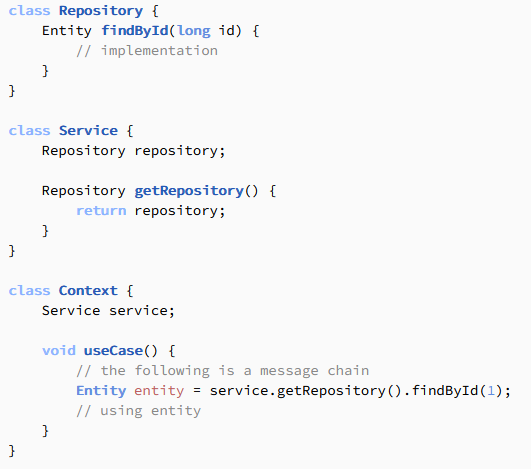
return endingMiles - startingMiles;

}

double miles = getMilesDriven(car) ?? 0.0;

double gallons = miles / MILES\_PER\_GALLON;

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



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);

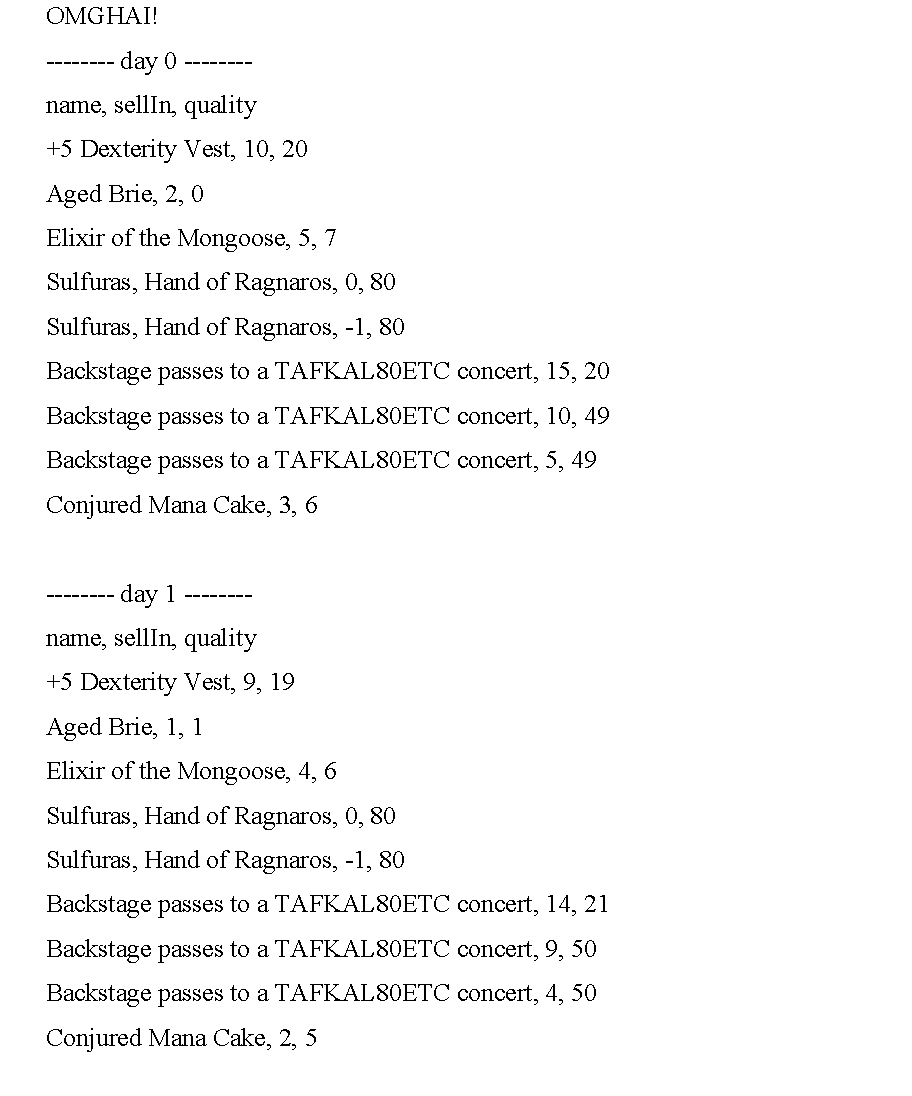
// fazer algo com a entidade encontrada

}

}

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.

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



* 1. Identifique quaisquer trechos de código repetidos que possam ser extraídos para métodos privados. Realize:
     1. **private** **void** decrementQuality ...
     2. **private** **void** incrementQuality ...
  2. Execute o teste e confira a saída atual com a original
  3. Considere extrair o código longo na primeira estrutura if/else para um método privado. Realize:
     1. **private** **void** UpdateQualityForItemsThatAgeWell...
  4. Execute o teste e confira a saída atual com a original
  5. Considere extrair um segundo bloco de código longo de estrutura if/else para o método **private** **void** UpdateQualityForExpiredItems
  6. Execute o teste e confira a saída atual com a original
  7. Analise o código do método UpdateQualityForItemsThatAgeWell e refatore.
  8. Execute o teste e confira a saída atual com a original
  9. Comente sobre esta questão em 3 a 5 linhas.

Respostas:

1. 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.