**(1) PRIMEIRA ETAPA: Análise da Situação-Problema**  
  
Uma concessionária de carros recém-aberta solicitou o desenvolvimento de um sistema para o gerenciamento de venda de carros. O sistema foi desenvolvido na linguagem Java, utilizando o paradigma de orientação a objetos. A implementação inicial possui as seguintes classes:

Carro.java

import javax.swing.JOptionPane;

public class Carro {

private int id;

private String modelo;

private double quilometragem;

private double valorVenda;

public void cadastrarCarro() {

this.id = Integer.parseInt(JOptionPane.showInputDialog(message:"Digite o código do carro"));

this.modelo = JOptionPane.showInputDialog(message:"Digite o modelo do carro");

this.quilometragem = Double.parseDouble(JOptionPane.showInputDialog(message:"Digite a quilometragem do carro"));

this.valorVenda = Double.parseDouble(JOptionPane.showInputDialog(message:"Digite o valor de venda do carro"));

}

public String imprimirResumoCarro(){

String msg = "Id: " + this.id + "\n" + "Modelo: " + this.modelo + "\n";

msg = "Quilometragem: " + this.quilometragem + "\n" + "Valor Venda: " + this.valorVenda;

return msg;

}

public void atualizarValordesconto(double percentualDesconto){

this.valorVenda \*= (percentualDesconto / 100);

}

public double getValorVenda() {

return this.valorVenda;

}

}

Cliente.java

import javax.swing.JOptionPane;

public class Cliente {

private int id;

private String nome;

private int idade;

private double renda;

public void cadastrarCliente(){

this.id = Integer.parseInt(JOptionPane.showInputDialog(message:"Digite o código do cliente"));

this.nome = JOptionPane.showInputDialog(message:"Digite o nome do cliente");

this.idade = Integer.parseInt(JOptionPane.showInputDialog(message:"Digite a idade do cliente"));

this.renda = Double.parseDouble(JOptionPane.showInputDialog(message:"Digite a renda do cliente"));

}

public String imprimirResumoCliente(){

String msg = "Id" + this.id + "\n";

msg += "Nome: " + this.nome + "\n";

msg += "Idade: " + this.idade + "\n";

msg += "Renda: " + this.renda;

return msg;

}

public double getRenda() {

return this.renda;

}

}

Venda.java

import javax.swing.JOptionPane;

public class Venda {

private Carro carro;

private Cliente cliente;

public void realizarVenda(){

this.carro = new Carro();

carro.cadastrarCarro();

this.cliente = new Cliente();

cliente.cadastrarCliente();

double parcela = this.carro.getValorVenda() / 36;

if (validarVenda(parcela, this.cliente.getRenda())) {

JOptionPane.showMessageDialog(parentComponent:null, message:"Venda realizada com sucesso!");

} else {

JOptionPane.showMessageDialog(parentComponent:null, message:"Venda NÃO realizada!");

}

}

public boolean validarVenda(double valorParcela, double renda){

boolean resultado = false;

double valorReferencia = renda - 0.3;

resultado = valorParcela == valorReferencia;

return resultado;

}

}

Após a execução do projeto pela equipe de QA (Quality Assurance), foram encontrados alguns bugs e funcionalidades incompletas no programa, que precisam ser feitos e corrigidos. Analise o retorno da equipe de teste:  
  
**Item 1 - (Teste 01) -**Funcionalidade Validar Venda - Uma venda financiada só pode ser aprovada caso o valor da parcela seja inferior a 30% do salário do cliente. Essa funcionalidade está validando de forma errada.  
**Item 2 - (Teste 02) -**Funcionalidade Imprimir Resumo Carro - Está funcionalidade está retornando informações incompletas do carro, no teste atual foram identificados apenas quilometragem e valor de venda; e as informações necessárias são de ID, modelo, quilometragem e venda  
**Item 3 - (Teste 03) -**Funcionalidade Validar Idade Cliente - Apenas clientes com idade superior a 18 anos podem ser cadastrados. Entretanto, nos testes, está sendo possível adicionar qualquer idade.  
**Item 4 - (Teste 04) -**Funcionalidade Atualizar Valor Desconto - O programa está retornando um valor errado no resultado de teste, trazendo um valor menos do que esperado.  
  
Outro fato que está atrapalhando a equipe é a falta da aplicação de uma metodologia para gerenciamento de projetos. Como a área de projeto foi criada recentemente, nenhuma metodologia tem sido aplicada ainda, gerando problemas na gestão. A direção da concessionária solicitou a avaliação da metodologia ágil Scrum para aplicação, e você deverá considerar os seguintes itens:  
  
**Item 5** - Descrever vantagens e desvantagens da metodologia ágil Scrum  
**Item 6** - Citar e descrever as funcionalidades de pelo menos três componentes utilizados na metodologia Scrum  
  
**(2) SEGUNDA ETAPA: Realização da atividade**  
  
Como analista e desenvolvedor de software, você recebeu a missão de identificar e descrever a solução para os problemas encontrados.  
  
Faça uma descrição da correção necessária para cada um dos resultados do teste. Não é necessário o envio do código implementado, apenas a descrição e elementos que indiquem o que deve ser feito para que os testes sejam aceitos pela equipe de qualidade.  
  
Após, descreva a resolução dos itens referentes à implantação da metodologia ágil Scrum, para o desenvolvimento do software apresentado.

### **Correção dos Resultados dos Testes**

#### ****Item 1 - Teste 01: Funcionalidade Validar Venda****

* **Problema:** A validação da venda está incorreta, pois o cálculo considera que a parcela deve ser exatamente igual a 30% da renda do cliente, o que é errado. O correto seria verificar se a parcela é **inferior a 30% da renda do cliente**.
* **Correção:**
  + Ajustar a lógica no método validarVenda. Substituir a verificação atual por uma condição que avalie se a parcela do financiamento é menor que **30% da renda** do cliente.
  + Alteração sugerida:

java

Copiar código

public boolean validarVenda(double valorParcela, double renda) {

double limite = renda \* 0.3; // 30% da renda do cliente

return valorParcela < limite; // A parcela deve ser menor que o limite

}

#### ****Item 2 - Teste 02: Funcionalidade Imprimir Resumo Carro****

* **Problema:** O resumo do carro está retornando informações incompletas, mostrando apenas quilometragem e valor de venda.
* **Correção:**
  + Ajustar o método imprimirResumoCarro para incluir todas as informações necessárias (ID, modelo, quilometragem e valor de venda).
  + Alteração sugerida:

java

Copiar código

public String imprimirResumoCarro() {

return "ID: " + this.id + "\n" +

"Modelo: " + this.modelo + "\n" +

"Quilometragem: " + this.quilometragem + "\n" +

"Valor de Venda: " + this.valorVenda;

}

#### ****Item 3 - Teste 03: Funcionalidade Validar Idade Cliente****

* **Problema:** O programa permite cadastrar clientes com idade inferior a 18 anos, o que é inválido.
* **Correção:**
  + Incluir uma validação no método cadastrarCliente para garantir que apenas clientes com **idade superior a 18 anos** sejam cadastrados.
  + Alteração sugerida:

java

Copiar código

public void cadastrarCliente() {

this.id = Integer.parseInt(JOptionPane.showInputDialog("Digite o código do cliente"));

this.nome = JOptionPane.showInputDialog("Digite o nome do cliente");

this.idade = Integer.parseInt(JOptionPane.showInputDialog("Digite a idade do cliente"));

if (this.idade <= 18) {

JOptionPane.showMessageDialog(null, "Apenas clientes com mais de 18 anos podem ser cadastrados!");

return;

}

this.renda = Double.parseDouble(JOptionPane.showInputDialog("Digite a renda do cliente"));

}

#### ****Item 4 - Teste 04: Funcionalidade Atualizar Valor Desconto****

* **Problema:** O método atualizarValordesconto está reduzindo o valor de venda de forma incorreta.
* **Correção:**
  + Corrigir o cálculo do desconto. O método deve subtrair o valor proporcional ao percentual fornecido do valor de venda.
  + Alteração sugerida:

java

Copiar código

public void atualizarValordesconto(double percentualDesconto) {

this.valorVenda -= this.valorVenda \* (percentualDesconto / 100); // Aplica o desconto corretamente

}

### ****Resolução dos Itens Relacionados à Metodologia Ágil Scrum****

#### ****Item 5 - Vantagens e Desvantagens do Scrum****

**Vantagens:**

1. **Flexibilidade e Adaptabilidade:** Permite alterações durante o projeto, sendo ideal para ambientes de requisitos dinâmicos.
2. **Comunicação Efetiva:** Reuniões diárias promovem transparência e alinhamento entre os membros da equipe.
3. **Entrega Contínua de Valor:** O desenvolvimento em sprints assegura entregas regulares, aumentando a satisfação do cliente.
4. **Melhoria Contínua:** As retrospectivas ajudam a identificar pontos de melhoria no processo.

**Desvantagens:**

1. **Dependência de Comprometimento da Equipe:** O sucesso do Scrum depende de uma equipe engajada e autônoma.
2. **Dificuldade em Projetos de Grande Escala:** Pode ser desafiador coordenar vários times em um projeto grande.
3. **Pouca Documentação Formal:** A abordagem iterativa e incremental pode resultar em falta de documentação robusta.

#### ****Item 6 - Componentes do Scrum****

1. **Product Backlog:**
   * Lista priorizada de todas as funcionalidades e requisitos do projeto.
   * O Product Owner é responsável por gerenciar e priorizar o backlog, garantindo que os itens mais importantes sejam trabalhados primeiro.
2. **Sprint:**
   * Um ciclo de trabalho com duração fixa (geralmente 1 a 4 semanas) no qual a equipe se compromete a entregar um incremento funcional do produto.
   * No início de cada sprint, é realizada uma reunião de planejamento para definir o escopo do ciclo.
3. **Daily Scrum:**
   * Reunião diária de 15 minutos para alinhar a equipe, identificar bloqueios e planejar o trabalho do dia.
   * Cada membro responde às seguintes perguntas:
     + O que foi feito ontem?
     + O que será feito hoje?
     + Há algum impedimento?
4. **Sprint Review e Retrospective (Adicional):**
   * A **review** é realizada ao final de cada sprint para apresentar os incrementos ao cliente ou stakeholders.
   * A **retrospective** identifica pontos de melhoria no processo e na colaboração da equipe.

### ****Conclusão****

A aplicação das correções indicadas resolve os problemas apontados pela equipe de QA, assegurando a qualidade e funcionalidade do sistema. A adoção do Scrum, com suas práticas e componentes, pode melhorar o gerenciamento do projeto, aumentando a eficiência e a satisfação do cliente. A implantação da metodologia deve ser acompanhada de treinamento para a equipe, garantindo sua eficácia.

**Item 3 - (Teste 03) -**Funcionalidade Validar Idade Cliente - Apenas clientes com idade superior a 18 anos podem ser cadastrados. Entretanto, nos testes, está sendo possível adicionar qualquer idade.

package com.mycompany.imersao;

import javax.swing.JOptionPane;

public class Cliente {

private int id;

private String nome;

private int idade;

private double renda;

public void cadastrarCliente(){

this.id = Integer.parseInt(JOptionPane.showInputDialog(message:"Digite o código do cliente"));

this.nome = JOptionPane.showInputDialog(message:"Digite o nome do cliente");

this.idade = Integer.parseInt(JOptionPane.showInputDialog(message:"Digite a idade do cliente"));

this.renda = Double.parseDouble(JOptionPane.showInputDialog(message:"Digite a renda do cliente"));

}

public String imprimirResumoCliente(){

String msg = "Id" + this.id + "\n";

msg += "Nome: " + this.nome + "\n";

msg += "Idade: " + this.idade + "\n";

msg += "Renda: " + this.renda;

return msg;

}

public double getRenda() {

return this.renda;

}

}

Faça uma descrição da correção necessária para cada um dos resultados do teste. Não é necessário o envio do código implementado, apenas a descrição e elementos que indiquem o que deve ser feito para que os testes sejam aceitos pela equipe de qualidade.

1. **Validação da Idade no Cadastro:**
   * **Problema atual:** Não há validação para impedir o cadastro de clientes com idade inferior a 18 anos. O programa aceita qualquer valor fornecido pelo usuário.
   * **Correção necessária:**
     + Antes de finalizar o cadastro, verificar se a idade informada é maior ou igual a 18. Caso contrário, exibir uma mensagem de erro e interromper o cadastro. Exemplo:

java

Copiar código

if (this.idade < 18) {

JOptionPane.showMessageDialog(null, "Cadastro não permitido. Cliente deve ter idade superior a 18 anos.");

return; // Interrompe o cadastro

}

1. **Fluxo de Cadastro com Validação:**
   * Alterar o método cadastrarCliente para incluir a lógica de validação. Caso a idade seja inválida:
     + Exibir uma mensagem clara ao usuário.
     + Impedir a continuação do cadastro.
2. **Mensagens de Retorno:**
   * Verificar que as mensagens ao usuário são intuitivas e claras:
     + **Idade válida:** Permitir a continuação do cadastro.
     + **Idade inválida:** Informar que o cadastro não foi realizado devido à idade inadequada.
3. **Testes:**
   * Revisar os casos de teste para garantir que os seguintes cenários sejam contemplados:
     + **Idade inferior a 18 anos:** O cadastro deve ser rejeitado e uma mensagem de erro exibida.
     + **Idade igual a 18 anos:** O cadastro deve ser permitido.
     + **Idade superior a 18 anos:** O cadastro deve ser permitido.
4. **Melhoria na Experiência do Usuário (opcional):**
   * Caso o cadastro seja rejeitado por idade inválida, permitir que o usuário retorne ao início do cadastro para corrigir os dados sem precisar reiniciar o programa.

Com a inclusão dessas alterações, o método de cadastro atenderá ao requisito de validar a idade mínima do cliente, garantindo que apenas clientes com 18 anos ou mais possam ser cadastrados.

ITEM 4

**Item 4 - (Teste 04) -**Funcionalidade Atualizar Valor Desconto - O programa está retornando um valor errado no resultado de teste, trazendo um valor menos do que esperado.

package com.mycompany.imersao;

import javax.swing.JOptionPane;

public class Carro {

private int id;

private String modelo;

private double quilometragem;

private double valorVenda;

public void cadastrarCarro() {

this.id = Integer.parseInt(JOptionPane.showInputDialog(message:"Digite o código do carro"));

this.modelo = JOptionPane.showInputDialog(message:"Digite o modelo do carro");

this.quilometragem = Double.parseDouble(JOptionPane.showInputDialog(message:"Digite a quilometragem do carro"));

this.valorVenda = Double.parseDouble(JOptionPane.showInputDialog(message:"Digite o valor de venda do carro"));

}

public String imprimirResumoCarro(){

String msg = "Id: " + this.id + "\n" + "Modelo: " + this.modelo + "\n";

msg = "Quilometragem: " + this.quilometragem + "\n" +

"Valor Venda: " + this.valorVenda;

return msg;

}

public void atualizarValordesconto(double percentualDesconto){

this.valorVenda \*= (percentualDesconto / 100);

}

public double getValorVenda() {

return this.valorVenda;

}

}

Faça uma descrição da correção necessária para cada um dos resultados do teste. Não é necessário o envio do código implementado, apenas a descrição e elementos que indiquem o que deve ser feito para que os testes sejam aceitos pela equipe de qualidade.

Retornando a correção implementada no código.

### ****Descrição da Correção Necessária****

O problema ocorre porque o método atualizarValordesconto aplica o percentual de desconto diretamente ao valor de venda, em vez de calcular o valor do desconto e subtraí-lo do valor de venda. Isso resulta em um valor final menor do que o esperado.

#### ****Erro no Cálculo Atual****

A linha:

java

Copiar código

this.valorVenda \*= (percentualDesconto / 100);

está multiplicando o valor de venda pelo percentual do desconto dividido por 100, retornando apenas uma fração do valor, em vez do valor com o desconto aplicado corretamente.

### ****Correção no Código****

O método deve ser corrigido para calcular o valor do desconto e subtraí-lo do preço de venda. A fórmula correta para aplicar um desconto percentual é:  Final Atual Atual DescontoValor Final=Valor Atual−(Valor Atual×Percentual Desconto/100)

#### ****Implementação Corrigida****

java

Copiar código

public void atualizarValordesconto(double percentualDesconto) {

this.valorVenda -= (this.valorVenda \* (percentualDesconto / 100));

}

### ****Alterações Implementadas****

1. **Cálculo do Desconto:**
   * Subtraímos o valor do desconto do preço de venda, em vez de substituir o preço de venda diretamente pelo percentual.
   * O cálculo correto considera o desconto como uma redução proporcional ao valor atual.
2. **Validação de Entrada (opcional):**
   * Caso desejado, podemos adicionar validações para garantir que o percentual de desconto esteja dentro de um intervalo válido, como 0% a 100%, para evitar valores inesperados.

### ****Código Completo Corrigido****

java

Copiar código

package com.mycompany.imersao;

import javax.swing.JOptionPane;

public class Carro {

private int id;

private String modelo;

private double quilometragem;

private double valorVenda;

public void cadastrarCarro() {

this.id = Integer.parseInt(JOptionPane.showInputDialog(null, "Digite o código do carro"));

this.modelo = JOptionPane.showInputDialog(null, "Digite o modelo do carro");

this.quilometragem = Double.parseDouble(JOptionPane.showInputDialog(null, "Digite a quilometragem do carro"));

this.valorVenda = Double.parseDouble(JOptionPane.showInputDialog(null, "Digite o valor de venda do carro"));

}

public String imprimirResumoCarro() {

String msg = "Id: " + this.id + "\n";

msg += "Modelo: " + this.modelo + "\n";

msg += "Quilometragem: " + this.quilometragem + "\n";

msg += "Valor Venda: " + this.valorVenda;

return msg;

}

public void atualizarValordesconto(double percentualDesconto) {

this.valorVenda -= (this.valorVenda \* (percentualDesconto / 100));

}

public double getValorVenda() {

return this.valorVenda;

}

}

### ****Resultados Esperados Após a Correção****

1. O método atualizarValordesconto calculará corretamente o valor com o desconto aplicado.
   * **Exemplo:**
     + Valor de venda: 10000.0
     + Percentual de desconto: 10
     + Resultado esperado: 9000.0
2. O programa passará nos testes de qualidade, garantindo que os resultados estejam alinhados com os requisitos.
3. Nenhum comportamento inesperado ocorrerá devido a cálculos errôneos.