UAST
Unidade Acadêmica de Serra Talhada - PE
UFRPE
Desde 2006

Bacharelado em Sistemas de Informação

MPOO

Sites: https://sites.google.com/site/profricodemery/mpoo

http://ava.ufrpe.br/

https://sigs.ufrpe.br/sigaa/ava/index.jsf

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS III

Leia atentamente as instruções gerais:

- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaIII.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** para cada questão: questao1, questao2, e assim sucessivamente, contendo todas as respostas da lista.
- Quando a questão envolver uma discussão teórica utilize um arquivo .txt (Menu File -> Submenu New -> Opção File), por exemplo, questao1.txt
- A lista envolve questões práticas e conceituais, então deverão ser entregues no AVA tanto os códigos-fonte (projeto completo) quanto às demais respostas. Em caso de imagens e digramas, você poderá salvar o arquivo também na pasta correspondente do projeto.
- A entrega da lista compõe sua frequência e avaliação na disciplina.

Responda:

1) Em diversas situações é preciso fazer o uso de informações `static` e `final`. Qual a diferença entre essas palavras-chaves em um sistema desenvolvido em Java?

2) Quando se tem os pacotes a e b, o que fazer para que classes de "b" possam utilizar classes do pacote a?

3) A partir da codificação abaixo:

```
1  package util;
2  import javax.swing.JOptionPane;
3
4  public class Mensagem {
5
6      public static final String MENSAGEM_FALHA = "Falha no sistema";
7      protected static final String MENSAGEM_SUCESSO= "Operação realizada com sucesso";
8      private static final String MENSAGEM_ERRO = "O sistema será finalizado";
9      static final String MENSAGEM = "Bem vindo ao sistema";
10
11     public static String exibirMensagemFalha(){
12         return "Falha";
13     }
14
15     public static void exibirMensagem(String mensagem){
16         JOptionPane.showMessageDialog(null, mensagem);
17     }
18 }
```

3.1) Apresente uma aplicação Java que utiliza `exibirMensagemFalha()`, `exibirMensagem(String mensagem)` e as diferentes mensagens *MENSAGEM_FALHA, MENSAGEM_SUCESSO, MENSAGEM_ERRO* e *MENSAGEM*.

3.2) Apresente o diagrama de classes de 3.1)

4) A partir da descrição do problema abaixo, apresente o devido diagrama de use case:

[RF01] - Devolver produto	
Ator Principal	Cliente
Atores Secundários	Funcionário e Caixa
Resumo	Este caso de uso descreve as etapas necessárias para que um cliente devolva um produto comprado.
Pré-condições	É necessário existir um produto.
Pós-condições	Escolher opção de devolução
Fluxo Principal	
Ações do Ator	**Ações do sistema**
1. Solicitar a devolução de um produto apresentando o produto.	
	2. Executar a Devolução de um produto
	3. Executar Caso de uso Atualizar estoque
	4. Efetuar troca ou ressarcir o valor do produto
Restrições /validações	
1. O produto só poderá ser devolvido pelo cliente que realizou a compra	
2. O produto só poderá ser devolvido se não apresentar avaria.	
Fluxo Alternativo I – devolução	
Ações do Ator	**Ações do sistema**
	1. Executar caso de uso Efetuar Troca
Fluxo Alternativo II – ressarcimento	
Ações do Ator	**Ações do sistema**
	1. Ressarcir ao cliente o valor pago pelo produto (não é considerado taxas de envio)

[RF02] – Atualizar estoque	
Ator Principal	Vendedor
Resumo	Este caso de uso descreve as etapas necessárias para atualizar o estoque
Pré-condições	É necessário haver uma demanda de atualização
Pós-condições	Verificar situação do produto
Fluxo Principal	
Ações do Ator	**Ações do sistema**
1. Atualizar a base com um produto	
	2. Executar a atualização de estoque
Restrições /validações	
1. Existência de produto requisitado para troca	

[RF03] – Efetuar Troca	
Ator Principal	Vendedor
Resumo	Este caso de uso descreve as etapas necessárias para efetuar a troca de um produto
Pré-condições	É necessário existir uma solicitação de troca de produto
Pós-condições	É necessário dar baixa do produto trocado no estoque
Fluxo Principal	
Ações do Ator	**Ações do sistema**
1. Efetuar a troca de um produto por outro	
	2. Executar o caso de uso Baixar Estoque
Restrições /validações	
1. Existência de produto requisitado para troca	

[RF04] – Ressarcir cliente	
Ator Principal	Caixa
Resumo	Este caso de uso descreve o ressarcimento de um cliente
Pré-condições	É necessário existir uma solicitação de ressarcimento
Fluxo Principal	
Ações do Ator	**Ações do sistema**
1. Realizar a devolução do ressarcimento de valores a cliente.	
	2. Executar o ressarcimento de valor a cliente
Restrições /validações	
1. Existência de uma conta bancária	
Fluxo Alternativo I – ressarcimento em conta	
Ações do Ator	**Ações do sistema**
	1. Executar a transferência de valor em conta bancária de um cliente

Fique atento!

Método construtor é um método que inicializa os atributos da classe. O nome do método construtor deverá ser o mesmo nome da classe.

```
public class Pessoa{
    String nome;
    int rg;

    public Pessoa (String n, int rg){
        this.nome = n;
        this.rg = rg;
    }
}
```

Em **Herança**, quando uma superclasse define um método construtor, logo o método **default deixa de existir!** Mas se houver a necessidade da existência de um "método construtor default" então se deve declará-lo!

```
public class Pessoa{
        String nome;
        int rg;

        public Pessoa (){}

        public Pessoa (String n, int rg){
                this.nome = n;
                this.rg = rg;
        }
}
```

Em **Herança**, subclasses podem definir pelo menos um construtor herdado para não ocorrer em erro de sintaxe. Entretanto, sugere-se que subclasses definam construtores para todos os herdados.

```
public class Usuario extends Pessoa{
        String login;
        String senha;

        public Usuario() {}

        public Usuario(String login, String senha) {
                super();
                this.login = login;
                this.senha = senha;
        }

        public Usuario(String nome, int rg, String login, String senha) {
                super(nome, rg);
                this.login = login;
                this.senha = senha;
        }
 }
```

```
public Pessoa (){}
```

```
public Pessoa (String n, int rg){
        this.nome = n;
        this.rg = rg;
}
```

5) Preencha as lacunas:

   5.1) Se a classe Pessoa herda da classe Animal, a classe Pessoa é chamada de _____ e a classe Animal é chamada de _____.

   5.2) O conceito de herança permite a _____, que economiza tempo no desenvolvimento e estimula a utilização de programas previamente testados.

   5.3) Quando uma classe é utilizada com o mecanismo de herança, ela se torna uma superclasse que fornece _____ e _____ para outras classes ou se torna uma subclasse.

   5.4) O relacionamento “é um” entre as classes representa o conceito de _____, enquanto o relacionamento “tem um” entre classes representa _____.

6) A partir da descrição do sistema apresenta na questão 4), implemente em Java as definições para os **atores** do sistema. Não é preciso implementar o corpo dos comportamentos, apenas suas definições. Faça o devido uso de Herança.

7) Analise o código abaixo e aponte mudanças necessárias. As correções devem ser justificadas.

```java
package questao7;

public class Conta{
    private int num;
    private double saldo;

    public int Conta (int n, double saldo) {
        num = n;
    }

    public void debito (double valor) {
        this.saldo-=valor;
    }

    public void credito (double valor) {
        this.saldo+=valor;
    }
}
```

```java
package questao7;

public class Poupanca extends Conta{
    public Poupanca (int num){
        super (num, saldo);
    }

    public void debito (double valor) {
        this.saldo-=valor;
    }

    public void rendeJuros(){
        this.saldo+=saldo*taxa/100;
    }
}
```

8) Crie o diagrama de classes e use case e implemente em Java os seguintes problemas:

8.1) O sistema de um Supermercado possui Funcionários, onde cada um possui matricula, RG, nome, função, senha. Quando o funcionário possui a função "gerente", ele poderá dar um desconto no total de uma compra. Possui Produtos, onde cada um possui nome, código e preço. Uma Compra possui um total, uma nota e descrição. Possui uma operação totalizar onde acrescenta ao total da compra o preço de um produto informado. A cada produto registrado na compra a descrição (nome do produto) e o seu valor é registrado na nota da compra (onde terá um detalhamento de todos os produtos comprados e o valor total da compra).
Sugestão:

- A operação registrar repassa o nome e o preço à nota após cada produto registrado.
- A operação resumir compra exibe toda a nota da compra, incluindo o total da compra.

8.2) Crie uma classe Data que forneça a data em múltiplos formatos. Use construtores sobrecarregados para criar objetos Data inicializados com datas em diferentes formatos de apresentação.