UAST
Unidade Acadêmica de Serra Talhada - PE
UFRPE
Desde 2006
BSi
Bacharelado em Sistemas de Informação

MPOO

Site: 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:

- Ao responder as perguntas desta lista informe, em cada questão, se você baseou sua resposta em alguma pesquisa ou se você respondeu a partir se seus próprios conhecimentos. Sendo assim use: "REFERÊNCIA: Elaboração própria" ou "REFERÊNCIA: citar local da pesquisa".

  Você sabe como referenciar uma fonte? Saiba mais sobre as normas da ABNT em
  - https://www.normasabnt.org/referencias-bibliograficas/
  - ABNT NBR 6023 (https://www.ufpe.br/documents/40070/1837975/ABNT+NBR+6023+2018+%281%29.pdf/3021f721-5be8-4e6d-951b-fa354dc490ed)

- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaIII.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** (Menu File -> Submenu New -> Opção Source Folder) para cada questão: questao1, questao2, e assim sucessivamente, contendo todas as respostas da lista.
- A lista envolve questões práticas e conceituais.
- Quando se questão envolver uma discussão teórica utilize um arquivo .txt (Menu File -> Submenu New -> Opção File), por exemplo, questao1.txt
- Quando se tratar de questão prática, na pasta de pacote utilize um arquivo .java (Menu File -> Submenu New -> Opção Class), por exemplo, Classe.java

Fique atento!

Em POO é possível utilizar o mecanismo de encapsulamento de variáveis para dar acesso às definições restritas de uma classe. Através do encapsulamento, atributos e métodos podem ser acessados.

Mas há um mito de que toda classe com atributo encapsulado como `private` tem de se disponibilizar seu get e set. Mas e se o atributo for de um controle interno? Por exemplo:

```
public class Usuario {
        private static int id_contador=0;
        int id;
        String login;
        String senha;
        String cpf;

        public Usuario(String login, String senha, String cpf) {
                this.id=++id_contador;
                this.login = login;
                this.senha = senha;
                this.cpf = cpf;
        }

        public int getId() {return id;}
        public void setId(int id) {     this.id = id;}

        public String getLogin() {return login;}
        public void setLogin(String login) {this.login = login;}

        public String getSenha() {return senha;}
        public void setSenha(String senha) {this.senha = senha;}

        public String getCpf() {return cpf;}
        public void setCpf(String cpf) {this.cpf = cpf;}
}
```

id_contador é variável de controle interno a classe Usuario.

Não há disponibilização de get e set! Não é de interesse da classe que outras classes tenham acesso a *id_contador*.

Tem como objetivo tualizar o `id` de um novo usuário como *auto-increment*.

Portanto:

"Modificadores de acesso são um conceito importante das linguagens orientadas a objetos. Eles permitem que os detalhes da implementação sejam ocultados em uma classe"

Curiosidade!

Em Java o encapsulamento **default** (quando não se explicita o modificador de acesso) é chamado de *package-private*.

Logo, *atributos poderão ser acessados por outras classes definidas em um mesmo pacote de maneira direta* (sem a necessidade de *getters* e *setters*). Vide a *source-folder* "curiosidade" no projeto **br.edu.mpoo.listaIII.SeuNomeSobrenome** disponibilizado.

Responda:

1) Para que serve os métodos *getters* e *setters*?

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

3) O que acontece quando em um projeto de software se define duas classes de mesmo nome em pacotes diferentes? Demonstre a criação de duas instâncias da classe Usuario, sendo uma a partir da definição no *package* pacoteA e outra a partir da definição do *package* pacoteB, pela App (do *package* app).

4)

A partir da codificação abaixo:

```
// ClasseA.java
package pacoteA;

public class ClasseA {
	public int varA1;
	private int varA2;

	public ClasseA(int varA1, int varA2) {
		this.varA1 = varA1;
		this.varA2 = varA2;
	}

	public int getVarA2() { return varA2; }
}

// ClasseB.java
package pacoteB;

public class ClasseB {
	int varB1;
	protected int varB2;

	public ClasseB(int varB1, int varB2) {
		this.varB1 = varB1;
		this.varB2 = varB2;
	}

	public int getVarB2() { return varB2; }
}

// App.java
package app;

import pacoteA.ClasseA;
import pacoteB.ClasseB;

public class App {
	public static void main(String[] args) {
		ClasseA objA = new ClasseA(1, 1);
		ClasseB objB = new ClasseB(1, 1);
		//?
	}
}
```

Qual seria a saída para `//?` assumindo as instruções abaixo:

Instrução `//?`	Saída
`System.out.println(objA.varA1);`	
`System.out.println(objA.varA2);`	
`System.out.println(objB.varB1);`	
`System.out.println(objB.varB2);`	
`System.out.println(objA.getVarA1());`	
`System.out.println(objA.getVarA2());`	
`System.out.println(objB.getVarB1());`	
`System.out.println(objB.getVarB2());`	
`System.out.println(new ClasseA(2, 2).varA1);`	
`System.out.println(new ClasseB(2, 3).getVarB2());`	

Desafio

5) **Desafio: Mito ou Verdade**. Podemos dizer que ao suprimir o modificador de acesso de um atributo (ou método) este é tido como `protected`, ou seja, seu acesso é direto apenas por classes que estejam definidas no mesmo pacote?
Diante deste questionamento, pesquise a diferença de restrição de acesso ao dado entre os modificadores de acesso `default` (private-package) e o `protected`.

Você Sabia?

Em Java há uma atividade que recupera a memória alocada dinamicamente de que o programa não precisa mais. Isso é o Garbage Collector.

Mas, diferentemente de outras linguagens, é uma atividade de baixa prioridade que é executada apenas quando o sistema está com a memória RAM comprometida em termos de espaço, ou seja, sem memória!

Para isso, procura objetos não utilizados: referência `null`.

Em C e C++, a coleta de lixo pode ser realizada pelo programador com a recuperação de memória alocada dinamicamente. Por exemplo:

```c
#include <stdio.h>
#include <alloc.h>
int main (void) {
   int *p;
   int a;
   ...
   p=(int *)malloc(a*sizeof(int));
   if (!p) {
       printf ("** Erro: Memoria Insuficiente**");
       exit;
   }
   ...
   free(p);
   ...
   return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
   int row = 5, col = 4;
   int *a = (int *)malloc(row * col * sizeof(int));
   int i, j;
   for (i = 0; i < row; i++)
      for (j = 0; j < col; j++)
         *(a + i*col + j) = i + j;
   printf("The array elements are:\n");
   for (i = 0; i < row; i++) {
      for (j = 0; j < col; j++) {
         printf("%d ", *(a + i*col + j));
      }
      printf("\n");
   }
   free(a);
}
```

6) Nas Listas de Exercícios I e II vimos que o método “construtor” realiza a alocação de uma área de memória. Mas:

   6.1) O que seria um “destrutor”?

   6.2) Como é realizada a liberação de memória em Java?

   6.3) Para que serve o método `finalize()` de Java?

7) Dado o código abaixo:

```java
//Usuario.java
public class Usuario {
    private String login;
    private int senha;

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

    public void destroyed(Usuario usuario){
        usuario=null;
        System.gc();
    }
}
//App.java
public class App {
    public static void main(String[] args) {
        Usuario user = new Usuario("Godofredo", 1234);
        user.destroyed(user);
    }
}
```

Responda:

   7.1) Houve liberação de memória do usuário "Godofredo"? Explique.

   7.2) No método `destroyed` é possível adicionar o comando `this=null;`? Explique.

8) Por que não se faz necessário invocar o coletor de lixo após a utilização de um objeto definido como variável local de um método? Por exemplo:

```
public static void exibirUsuario(Usuario usuario) {
        String dadosUsuario="Dados usuário: \n";
        dadosUsuario+= "id: " + usuario.id + "\n";
        dadosUsuario+= "Login: " + usuario.login + "\n";
        dadosUsuario+= "Senha: " + usuario.senha + "\n";
        dadosUsuario+= "CPF: " + usuario.cpf;
        JOptionPane.showMessageDialog(null, dadosUsuario);
        //Por que não fazer a liberação de memória abaixo:
        //usuario=null;
        //dadosUsuario=null;
        //System.gc();
}
```

9) Sobre Garbage Collector:

   9.1) Ainda que o programador não "chame" diretamente `System.gc()`, quando ele é executado?

   9.2) Faça uma aplicação Java ilustrando a alocação de MUITA memória e a ilustração do coletor de lixo em funcionamento.

Desafio

Você, aluno de MPOO, está experienciando situações-problemas do universo de desenvolvimento de software e começará a ser desafiado a solucionar problemas a partir de conhecimentos de Programação e Orientação a Objetos.

10) Crie um diagrama de classes, use case e a codificação Java para o seguinte problema:

Geralmente as frutas contêm casca e caroços. Crie os métodos retirarCaroco( ) - que retira os caroços um a um de uma fruta, caso haja caroços; retirarCasca () - que retira a casca de uma fruta, caso haja casca; e o método comerFruta( ) que retira a casca e os caroços e elimina uma fruta. Faça o devido uso de coletor de lixo para liberar a memória da fruta. Faça o devido uso de construtores e parâmetros para os métodos. Crie uma aplicação que ilustra diversos tipos de frutas: com caroço(s) e casca; sem caroço(s) e com casca; com caroço(s) e sem casca.

Analise e trate em sua solução: como diferenciar uma melância de um abacate?