Universidade Estadual de Campinas – UNICAMP
Faculdade de Tecnologia – FT

Programação Orientada a Objetos II
SI400
Atividade 2: Exercícios de Fixação – Java

Samuel Lima Martins - 173820

Limeira, SP

Exercícios de desenvolvimento de código:

1. Variáveis Privadas: Escreva uma classe Java com variáveis privadas representando o nome e a idade de uma pessoa. Forneça métodos públicos para definir e obter esses valores.

```java
public class Person {
    private String name;
    private int age;

    public void setNome(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int Age) {
        this.Age = age;
    }

    public int getAge() {
        return age;
    }
}
```

2. Modificadores de acesso: Crie uma classe com métodos e variáveis privadas, protegidas, padrão e públicas. Instancie esta classe em outro pacote e tente acessar cada tipo de método e variável.

```java
public class Acessos {

    public String publico = "Publica";

    private String privado = "Privada";
```

```
    protected String protegido = "Protegida";

    String padrao = "Padrão";

    /*Métodos*/

    public String acessoPublico() {
        return "Acessivel";
    }

    protected String acessoProtegido() {
        return "Acessivel";
    }

    String acessoPadrao() {
        return "Acessivel";
    }

    private String acessoPrivado() {
        return "Acessivel";
    }
}

public class Main {

    public static void main(String[] args) {
        Acessos a = new Acessos();

        // Variáveis
        System.out.println("Público: " + a.publico);
        System.out.println("Protegido: " + a.protegido);
        // System.out.println("Privado: " + a.privado);
        System.out.println("Padrão: " + a.padrao);

        // Métodos
        System.out.println("\nMétodos:");
        System.out.println("Público: " + a.acessoPublico());
        System.out.println("Protegido: " + a.acessoProtegido());
        // System.out.println("Privado: " + a.acessoPrivado());
        System.out.println("Padrão: " + a.acessoPadrao());
    }
}
```

3. Encapsulamento do Construtor: Projete uma classe para uma conta bancária com variáveis privadas para saldo e número da conta. Crie um construtor que inicialize o saldo. Fornece métodos para depositar, sacar e verificar saldo.

```java
public class Conta {
    private double saldo;
    private String numeroDaConta;

    public ContaCorrente(String numeroDaConta, double saldoInicial) {
        this.numeroDaConta = numeroDaConta;
        this.saldo = saldoInicial;
    }

    public void depositar(double valor) {
        saldo += valor;
    }

    public void sacar(double valor) {
        saldo -= valor;
    }

    public double verificar() {
        return saldo;
    }
}
```

4. Lógica Getter e Setter: Implemente uma classe que representa uma temperatura em Celsius. Use variáveis privadas para armazenar a temperatura e forneça métodos para defini-la em Celsius e obtê-la em Fahrenheit.

```java
public class Temperatura {
    private double temperaturaCelsius;

    public void setTemperaturaCelsius(double temperaturaCelsius) {
        this.temperaturaCelsius = temperaturaCelsius;
    }
```

```
    public double getTemperaturaCelsius() {
        return temperaturaCelsius;
    }

    public double getTemperaturaFahrenheit() {
        return (temperaturaCelsius * 9/5) + 32;
    }
}
```

5. Classe Imutável: Projete uma classe imutável representando um ponto 2D (x, y). Uma vez criado o ponto, ele não deverá ser modificável. No entanto, suas coordenadas devem ser acessíveis por métodos getters públicos.

Java

```java
public final class Ponto2D {
    private final int Eixox;
    private final int Eixoy;

    public Ponto2D(int x, int y) {
        this.Eixox = x;
        this.Eixoy = y;
    }

    public int getX() {
        return Eixox;
    }

    public int getY() {
        return Eixoy;
    }
}
```

6. Acesso privado ao pacote: Crie um pacote contendo várias classes. Defina uma classe com acesso privado ao pacote e outra com acesso público. Tente acessar a classe privada de fora do pacote.

```java
package pacotePrivado;

class ClassePrivada {
    public void mensagem() {
        System.out.println("Classe privada ao pacote.");
    }
}

public class ClassePublica {
    public void mensagemPublica() {
        System.out.println("Classe pública.");
    }
}
```

Exercícios Teóricos:

7. Definição de encapsulamento: Defina encapsulamento em Java e explique por que ele é importante na programação orientada a objetos.

Encapsulamento ou acessibilidade é o conceito que define a disponibilidade de acesso da classe para com os atributos, é a forma como os elementos da classe podem ser vistos e utilizados por outras classes.

Sobre a importância do uso do encapsulamento em Java, podemos destacar algumas vantagens da implementação, uma delas é ocultar certos detalhes de implementação, reduzindo o tamanho do código, tornando-os mais legíveis e minimizando erros. Outra vantagem diz respeito a segurança, uma vez que o uso de encapsulamento possibilita maior controle sobre a disponibilidade de dados para as classes.

8. Explicação dos modificadores de acesso: Descreva os quatro modificadores de acesso (privado, padrão, protegido, público) em Java. Forneça exemplos de onde cada um deve ser usado.

R:

Private - limita o acesso à própria classe em que está definido. Nenhuma outra classe, nem mesmo subclasses ou classes no mesmo pacote, pode acessar.

Package - se define por não estar declarado explicitamente, é a ausência da definição e Ele permite que o membro seja acessível apenas dentro do mesmo pacote.

Protegido - permite que o membro seja acessível dentro da mesma classe, como no private, mas dessa vez também é acessível a outras classes no mesmo pacote.

Public- maior nível de acesso, pode ser acessada por qualquer classe do programa.

9. Benefícios do encapsulamento: Discuta os benefícios do encapsulamento, concentrando-se na capacidade de manutenção, reutilização e segurança do código.

R:
Sobre segurança, discernir a respeito na questão 7, já sobre a reutilização de código podemos citar a característica de que com o controle de acesso da classe e as características de possuir métodos, basta incluir uma classe específica em um outro programa e invocar seu método e ele funcionará além do fato de não ter de repetir a declaração das variáveis por exemplo apenas instanciando o objeto da classe. E por fim, sobre a manutenção do código o fato de ter um código mais enxuto e com poucas repetições torna a correção do erro facilmente replicável para toda a base de código.

10. Encapsulamento vs. Abstração: Explique a diferença entre encapsulamento e abstração e como eles contribuem para a construção de sistemas de software eficazes.

R:
A abstração é o conceito de criar uma classe com apenas aquilo que for essencial a ela e a partir dessa substância essencial o desenvolvedor pode focar no que é imediato para a aplicação se preocupando com o subjacente apenas quando a hora chegar, então nesse momento poderá expecializar a classe. Já o encapsulamento se foca em controlar o acesso de classes a variáveis entre si e com seus próprios atributos também.
Mesmo não se assemelhando tanto na implementação esse dois conceitos se aproximam em seus objetivos e resultados, tornar o software modular e flexível.

11. Objetivo do getter e do setter: Descreva o propósito dos métodos getter e setter. Discuta cenários onde eles são úteis e quando podem ser evitados.

R:
O objetivo do getter é dar acesso a um atributo de uma classe, já o do setter é alterar esse atributo, Portanto, quando discutimos quando são uteis podemos citar casos em que é necessário controle de acesso a atributos, além de casos onde é necessário manter a consistência dos dados, já em casos em que a preocupação com o acesso, ou quando não for necessário alterar atributos, ou quando o atributo é derivado de outro, não é necessária a utilização de getters e setters.

12. Encapsulamento e ocultação de informações: Explique como o encapsulamento facilita a ocultação de informações e por que ele é crucial para gerenciar a complexidade no desenvolvimento de software.

R:
Sobre a ocultação de informações, a mesma se dá com a utilização do modificador de acesso private, esse estabelece acesso ao atributo somente por meio da sua classe, mantendo a regra mais restrita podemos restringir a quantidade de meios possíveis de se utilizar esse atributo e manter os bugs mais distantes, mantendo as regras bem claras e a complexidade diminuída.