5ª Aula Prática de PLC: Conceitos de Orientação a Objetos em Java

Roteiro

- Encapsulamento
- Métodos e Atributos Estáticos
- Herança e Polimorfismo
- Interfaces e Classes Abstratas

Encapsulamento

Na orientação a objetos, o uso da encapsulação e ocultamento da informação recomenda que a representação do estado de um objeto deve ser mantida oculta. Cada objeto deve ser manipulado exclusivamente através dos métodos públicos do objeto, dos quais apenas a assinatura deve ser revelada.

Dessa forma, detalhes internos sobre a operação do objeto não são conhecidos, permitindo que o usuário do objeto trabalhe em um nível mais alto de abstração, sem preocupação com os detalhes internos da classe.

Encapsulamento

Os modificadores de acesso/visibilidade podem estar presentes tanto para atributos como para métodos. Em princípio, três categorias de visibilidade podem ser definidas:

- private: o atributo ou método de um objeto dessa classe não pode ser acessado por nenhum outro objeto (nenhuma visibilidade externa);
- protected: o atributo ou método de um objeto dessa classe poderá ser acessado apenas por objetos de classes que sejam derivadas dessa através do mecanismo de herança e que estejam no mesmo pacote;
- public: o atributo ou método de um objeto dessa classe pode ser acessado por qualquer outro objeto (visibilidade externa total).

Encapsulamento

Relembrando a classe TV:

```
class TV{
        int canal, volume;
        boolean ligada;
        void ligarTV( );
        void ligarTV(boolean ligada);
        void ligarTV(int canal);
        TV(int canal, int volume){
                this.canal=canal;
                this.volume=volume;
                ligada=true;
        }
}
```

Encapsulamento

Onde:

```
void ligarTV( ){
        canal=3;
        volume=30;
        ligada=true;
}
void ligarTV(boolean ligada){
        canal=3;
        volume=30;
        this.ligada=ligada;
}

void ligarTV(int canal){
        this.canal=canal;
        volume=30;
        ligada=true;
}
```

Encapsulamento

Adicionando os modificadores:

```
public class TV{
        private int canal, volume;
        private boolean ligada;
        public void ligarTV( );
        public void ligarTV(boolean ligada);
        public void ligarTV(int canal);
        public TV(int canal, int volume);
}
```

Encapsulamento

Dependendo dos modificadores de acesso (private, por exemplo), alguns atributos podem ficar completamente inacessíveis. Então, para permitir o acesso a esses atributos de uma maneira controlada, a prática mais comum é criar dois tipos de métodos, um que retorna o valor (getter) e outro que modifica/seta o valor (setter). Por exemplo:

```
public int getVolume( ){
        return volume;
}
public void setVolume(int volume){
        this.volume=volume;
}
```

Métodos e Atributos Estáticos

Como já foi visto anteriormente, objetos são instâncias de classes. Dessa forma os valores dos atributos dos objetos de uma mesma classe são independentes entre si, ou seja, a modificação do valor de um atributo de uma determinada instância de uma classe não modifica o valor das outras instâncias.

Uma forma, então, de compartilhar o mesmo valor de um atributo entre as instâncias de uma classe é utilizando os atributos estáticos.

Um atributo estático não é um atributo de cada objeto, e sim um atributo da classe, a informação fica guardada pela classe, não é mais individual para cada instância da classe.

Métodos e Atributos Estáticos

Adicionando um atributo estático:

```
public class TV{
        private static int numeroTotalTVs=50000;
        private int canal, volume;
        private boolean ligada;
        public void ligarTV( );
        public void ligarTV(boolean ligada);
        public void ligarTV(int canal);
        public TV(int canal, int volume);
}
```

Métodos e Atributos Estáticos

Para acessar um atributo estático não é possível utilizar a palavra reservada this. Nesse caso, deve-se utilizar o nome da classe. Por exemplo:

```
public int getNumeroTotalTVs( ){
        return numeroTotalTVs;
}
public void setNumeroTotalTVs(int numeroTotalTVs){
        TV.numeroTotalTVs=numeroTotalTVs;
}
```

Métodos e Atributos Estáticos

Entretanto, utilizando essa abordagem, é necessária sempre uma instância da classe para acessar o atributo estático.

Para contornar esse problema, utiliza-se métodos estáticos (também conhecidos como métodos da classe) que podem ser chamados sem a necessidade de uma instância da classe.

```
public static int getNumeroTotalTVs( ){
        return numeroTotalTVs;
}
public static void setNumeroTotalTVs(int numeroTotalTVs){
        TV.numeroTotalTVs=numeroTotalTVs;
}
```

Métodos e Atributos Estáticos

Métodos que manipulem apenas atributos estáticos e/ou variáveis passadas como argumentos para os mesmos, ou seja, métodos que independem dos atributos armazenados em instâncias da classe e/ou de outros métodos não estáticos podem ser fortes candidatos a serem declarados como métodos estáticos.

Resumidamente: métodos estáticos podem apenas acessar atributos e outros métodos estáticos.

Exercícios Resolvidos

1. Crie uma classe Pilha que encapsula uma pilha de até 1000 números inteiros. Escolha a estrutura de dados mais adequada para implementar essa pilha. Para manipular essa classe, defina apenas as operações básicas de uma pilha como push, pop, top e isEmpty. A estrutura de dados utilizada deverá ficar completamente invisível externamente.

Herança e Polimorfismo

Herança ou generalização é o mecanismo pelo qual uma classe (subclasse) pode estender outra classe (superclasse), aproveitando seus comportamentos (métodos) e variáveis possíveis (atributos).

Já o Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse.

Herança e Polimorfismo

Relembrando a classe Pessoa:

```
public class Pessoa{
	protected String nome;
	protected int idade;
	protected char sexo;
	protected float altura, peso;
}
```

Herança e Polimorfismo

Exemplos de herança:

```
public class Eleitor extends Pessoa{
        private int zona, secao;
        private String numeroInscricao;
}

public class Motorista extends Pessoa{
        private String registro, placaCarro;
        private boolean definitiva;
}
```

Herança e Polimorfismo

O mecanismo de redefinição de métodos é utilizado para redefinir/sobrescrever métodos nas classes derivadas de forma que eles possam ter um comportamento diferente da sua classe ancestral. É imprescindível que o método reescrito tenha exatamente a mesma assinatura do método definido na superclasse.

Herança e Polimorfismo

```
public class SmartTV extends TV{
        private String[] aplicativos;
        public void iniciarNavegador( );
        public void ligarTV(boolean ligada){
                canal=101;      volume=45;
                this.ligada=ligada;
                this.iniciarNavegador( );
        }
        public SmartTV(int canal, int volume){
                super(canal, volume);
        }
}
```

Interfaces e Classes Abstratas

Interfaces, assim como o mecanismo de herança, possibilitam referenciar classes de uma mesma maneira, isto é, achar um fator/descendente comum.

Interfaces, ainda, podem ser visualizadas como uma espécie de contratos que definem métodos que uma classe deve implementar se for assinar esse contrato.

Uma interface pode definir uma série de métodos, mas nunca conter implementação deles. Ela só expõe o que o objeto deve fazer, e não como ele faz, nem o que ele tem. Como ele faz vai ser definido em uma implementação dessa interface.

Interfaces e Classes Abstratas

Exemplo de definição de uma interface:

```
public interface Printable{
        public void autoPrint( );
}
```

Interfaces e Classes Abstratas

Exemplo de implementação de uma interface:

```
public class Pessoa implements Printable{
        protected String nome;
        protected int idade;
        protected char sexo;
        protected float altura, peso;
        public void autoPrint( ){
                System.out.println(nome + " " + idade + " " + sexo);
        }
}
```

Interfaces e Classes Abstratas

Diferentemente das classes, uma interface pode herdar de mais de uma interface. É como um contrato que depende que outros contratos sejam fechados antes deste valer. Você não herda métodos e atributos, mas sim responsabilidades.

Interfaces e Classes Abstratas

Pode se dizer que uma classe abstrata idealiza um tipo, define apenas um rascunho. Diferentemente das interfaces, classes abstratas podem ter métodos implementados, assim como atributos. Entretanto, não pode haver instâncias dessas classes.

Redefinindo a classe Pessoa como uma classe abstrata:

```
public abstract class Pessoa{
	protected String nome;
	protected int idade;
	protected char sexo;
	protected float altura, peso;
}
```

Interfaces e Classes Abstratas

Em uma classe abstrata, é possível escrever que determinados métodos serão sempre implementados pelas classes derivadas. Isto é, é possível definir métodos abstratos.

Redefinindo os métodos da classe TV como abstratos:

```
public abstract class TV{
        private static int numeroTotalTVs=50000;
        private int canal, volume;
        private boolean ligada;
        public abstract void ligarTV( );
        public abstract void ligarTV(boolean ligada);
        public abstract void ligarTV(int canal);
}
```

Exercícios Resolvidos

2. Crie uma classe Equipamento com o atributo ligado (tipo boolean) e com os métodos liga e desliga. O método liga seta o atributo ligado para true, enquanto que o método desliga seta o atributo ligado para false. Crie um construtor para essa classe que inicialize o equipamento desligado.

Crie, agora, uma classe EquipamentoSonoro que herda as características de Equipamento e possua o atributo volume (tipo short que varia de 0 a 10). A classe ainda deve possuir métodos para ler e alterar o volume (getter e setter). Ao ligar algum EquipamentoSonoro através do método liga, seu volume é automaticamente ajustado para 5. Finalmente, crie um construtor para essa classe que inicialize o equipamento desligado e sem volume.

Exercícios Práticos

1.Parabéns, você foi contratado pela NASA para recriar o sistema de organização de Instrumentos Espaciais dele, pois o antigo foi hackeado por um ser desconhecido. Com a intenção de te ajudar, a NASA te ofereceu as seguintes informações:

Todo Instrumento Espacial da NASA possui ID (int) para identificá-lo e seu CUSTO (int) de fabricação

Toda Carga tem ID (int), NOME(String), QUANTIDADE ALOCADA (int) e INSTRUMENTO ESPACIAL alocado.

Todo Foguete é um Instrumento Espacial que devem possuir ALTURA (int), COMPRIMENTO (int), LARGURA (int), PILOTO (string), QUANTIDADE DE COMBUSTÍVEL EM LITROS (int) e DESTINO(string) da viagem, onde deverá ser permitido ter acesso a todos os dados e alterar somente o nome do piloto, a quantidade de combustível e o destino do foguete.

Exercícios Práticos

Além disso há dois tipos de foguetes:

a)Foguetes de Exploração que deve conter um ARRAY DE PASSAGEIROS (array de string), no qual deve ser possível adicionar ou remover um passageiro.

b)Foguetes de Carga que deve conter QUANTIDADE DE CARGAS POSSIVEIS (int), ARRAY DE CARGAS, onde se deve ter um método para inserir cargas, checando se a quantidade de cargas inseridas é <= do que a quantidade de cargas possíveis.

Exercícios Práticos

2. Defina uma classe Telefone que represente um número de telefone no formato internacional de uma agenda de contatos de um celular. Um telefone possui um tipo (Residencial, Celular, Comercial ou Fax), um código DDI, um código DDD e o número de telefone propriamente dito. Os atributos da classe Telefone devem ser mantidos ocultos externamente de acordo com o conceito do encapsulamento e ocultação da informação e manipulados, apenas, através dos seus respectivos métodos getters e setters.

Exercícios Práticos

a) Defina, também, um tipo enumerável TipoTelefone que represente os 4 possíveis tipos de telefone já mencionados acima.

b) Defina métodos getters e setters para os atributos da classe Telefone.

c) Defina quatro construtores sobrecarregados para a classe Telefone: o primeiro deve receber como parâmetro o tipo do telefone, os códigos DDI e DDD e o número do telefone; o segundo deve receber os mesmos parâmetros do primeiro construtor, exceto o código DDI; o terceiro deve receber os mesmos parâmetros do segundo construtor,

exceto o código DDD; finalmente, o quarto, e último, deve receber apenas como parâmetro o número do telefone. Implemente os três últimos construtores de forma que eles sempre chamem o primeiro construtor e inicialize, quando necessário, os parâmetros que faltam com os seguintes valores padrão: Celular para o tipo do telefone, 55 para o código DDI e 81 para o código DDD.

Exercícios Práticos

d) Sobreescreva os métodos equals e toString da classe Object (java.lang.Object) na classe Telefone para que seja possível comparar se duas instâncias dessa classe tem o mesmo conteúdo (através da palavra chave instanceof) e para transformar uma instância dessa mesma classe em uma String. Considere que a String que representará um objeto da classe Telefone será formada pelo junção do código DDI, do código DDD e do número

do telefone, nessa ordem.

e) Crie, também, uma aplicação simples (Pode ser em outra classe) que utilize ao menos uma vez cada um dos métodos definidos e sobrescritos nessa classe e, também, algum dos quatro construtores.

Exercícios Práticos

- Os exercícios práticos devem ser realizados individualmente e enviados por e-mail com o assunto “[IF686EC] EXERCÍCIOS PRÁTICOS 05” para monitoria-if686-ec-l@cin.ufpe.br até as 23:59 de sábado (09.12.2017).
- As resoluções dos exercícios devem estar em arquivos compactados diferentes, um arquivo por exercício, com os nomes no formato “Q[número do exercício].zip”. Onde cada arquivo desse poderá conter vários classes de Java no formato “[nome da classe].java”.