Programação II

Prof. Alcemir Rodrigues Santos

Encapsulamento e Visibilidade

Agenda

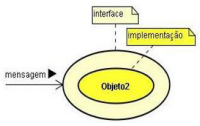
- Encapsulamento
- Visibilidade
- Exercícios

O que é encapsulamento ?

- O que você entende por encapsular?!
  - Ato de envolver em uma cápsula: Isolar!
- Encapsulamento é a característica da OO capaz de ocultar partes (dados e detalhes) de implementação interna de classes do mundo exterior.
- Isso torna-se necessário para que possamos controlar melhor as modificações em uma classe, em especial em seus atributos

O que é encapsulamento ?

- Cria o conceito de que um determinado objeto oferece um serviços, mas quem o usa não sabe como isso é realizado e deve se preocupar apenas com o resultado final
- A interface é o que liga o usuário com os métodos/atributos internos da classe



O que é encapsulamento ?

- Podemos dizer, nesse caso, que estamos aplicando a característica de encapsulamento ao objeto , pois ocultamos do objeto Pessoa os detalhes da implementação dos serviços oferecidos pelo Automóvel.

Encapsulamento

- Em uma classe Quadrado, quais métodos devem ser públicos e quais devem ser privados?
  - Transladar
  - Ampliar
  - Mover ponto
  - Girar
  - Adicionar ponto
  - Calcular área

Visibilidade

- Há basicamente três maneiras de acessar (utilizar) um atributo ou método:
  - Dentro da própria classe que o está definindo;
  - Através da instância (objeto) da classe que o definiu;
  - Através de um mecanismo chamado de herança.

Modificadores de Acesso

- Os são palavras-chave ou reservadas da linguagem Java cuja utilidade é permitir ou proibir o acesso aos atributos e/ou métodos das classes, pode ser:
  - Public
  - Private
  - Protected

Modificadores de Acesso

- **public:** garante que o atributo ou método da classe seja acessado ou executado a partir de qualquer outra classe.
- **private:** pode ser acessado, modificado ou executado apenas por métodos da mesma classe, sendo totalmente oculto ao programador (ou outros objetos do sistema) que for usar instâncias dessa classe.
- **protected:** funciona como o private, exceto que as classes filhas ou derivadas também terão acesso ao atributo ou método. Veremos mais sobre classes filhas na aula de Herança.

Modificadores de Acesso

- **Padrão:** não são palavras reservadas de modificadores de acesso.
- Os atributos e métodos são chamados de Padrão quando não possuem modificadores, ou seja, são os atributos e métodos declarados sem modificadores. Isso significa que podem ser acessados por todas as classes pertencentes a um mesmo pacote (pacotes são pastas onde estão inseridos os arquivos das classes, para ajudar a organizá-las).

Modificadores de Acesso

Visibilidade	Aplicado à	Permite Acesso	Palavra Reservada
Pública	Classes, Métodos e Atributos	A partir de qualquer classe	public
Privada	Métodos e Atributos	Apenas dentro da classe que os define	Private
Padrão	Classes, Métodos e Atributos	Apenas a classes do mesmo pacote	-
Protegida	Métodos e Atributos	Por classes do mesmo pacote através de herança ou de instanciação; e Por classes de outros pacotes apenas através de herança;	protected

Atividade

- Implemente classe que representa uma agenda, onde o usuário informa dia, mês e uma anotação para esta data. A agenda deve ter métodos para:
  - Realizar a anotação de eventos: anote(dia, mês, nota);
  - Verificar se a data informada é uma data correta, caso não seja, deve-se mudar a nota para "anotação não inserida devido a data invalida": validaData();
  - Mostrar a anotação, no formato 'dia/mês: nota': mostrarAnotacao();

Um exemplo completo

```
class Agenda {
        int dia;
        int mes;
        String anotacao;
}
```

Um exemplo completo

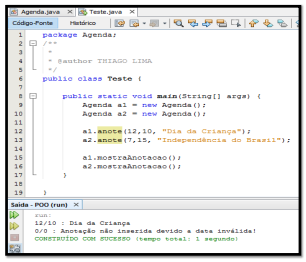
```
class Agenda {
        int dia;
        int mes;
        String anotacao;

        void anote(int d, int m, String nota){
                dia = d;
                mes = m;
                anotacao = nota;

                validaData();
        }

        void validaData(){
                if((dia < 1) || (dia> 31) || (mes< 1) || (mês > 12)){
                        dia=0;
                        mês=0;
                        anotacao = "anotação não inserida devido data
        inválida";
                        }
}

        void mostraAnotacao(){
                System.out.println(dia+"/"+mês+" : "+ anotacao);
        }
}
```

Um exemplo completo

```
class Agenda {
        int dia;
        int mes;
        String anotacao;

        void anote(int d, int m, String nota){
                dia = d;
                mes = m;
                anotacao = nota;

                validaData();
        }

        void validaData(){
                if((dia < 1) || (dia> 31) || (mes< 1) || (mês > 12))
                        dia=0;
                        mês=0;
                        anotacao = "anotação não inserida devido a
        data inválida";
                        }
        }

        void mostraAnotacao(){
                System.out.println(dia+"/"+mês+" : "+ anotacao);
        }
}
```

Atributos e métodos sem modificadores de acesso

Um exemplo completo



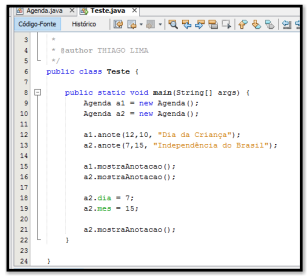
```
package Agenda;
/**
 *
 * @author THIAGO LIMA
 */
public class Teste {

    public static void main(String[] args) {
        Agenda a1 = new Agenda();
        Agenda a2 = new Agenda();

        a1.anote(12,10, "Dia da Criança");
        a2.anote(7,15, "Independência do Brasil");

        a1.mostraAnotacao();
        a2.mostraAnotacao();
    }

}
```

```
Saída - POO (run)
run:
12/10 : Dia da Criança
0/0 : Anotação não inserida devido a data inválida!
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Um exemplo completo

- Perfeito!!! Não utilizamos o encapsulamento na classe Agenda e tudo funcionou perfeitamente! Sem nenhuma via de acesso desprotegida, correto?
- Errado! Vamos identificar a falha da nossa codificação.
- Veja o que acontece se fizéssemos uma pequena modificação no método main().

Um exemplo completo



```
 *
 * @author THIAGO LIMA
 */
public class Teste {

    public static void main(String[] args) {
        Agenda a1 = new Agenda();
        Agenda a2 = new Agenda();

        a1.anote(12,10, "Dia da Criança");
        a2.anote(7,15, "Independência do Brasil");

        a1.mostraAnotacao();
        a2.mostraAnotacao();

        a2.dia = 7;
        a2.mes = 15;

        a2.mostraAnotacao();
    }

}
```

Um exemplo completo

```
Saída - POO (run)
run:
12/10 : Dia da Criança
0/0 : Anotação não inserida devido a data inválida!
7/15 : Anotação não inserida devido a data inválida!
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

- Veja que sua Agenda permitiu que você inserisse uma data inválida para uma anotação. Isso significa que seu código está susceptível a falhas. Em um programa simples como esse, isso não irá trazer nenhuma dor de cabeça. Agora, imagine em um programa real usado diariamente pelas pessoas.
- Qual é a solução?
- Respondendo: a solução é aplicar o encapsulamento.

Um exemplo completo

- Você pode estar se perguntando... Quando a classe Principal teve acesso aos atributos da classe Agenda?

O método main da classe Principal está acessando os atributos dia, mês e anotação da classe Agenda. E modificando as informações sem passar pelo teste da validação da data.

```
public class Principal {

    public static void main(String[] args) {

        Agenda agenda1 = new Agenda();
        Agenda agenda2 = new Agenda();

        agenda1.anote(12,10,"Dia da Criança");
        agenda2.anote(7,15,"Independência do Brasil");

        agenda1.mostraAnotacao();
        agenda2.mostraAnotacao();

        agenda2.dia = 7;
        agenda2.mes = 15;
        agenda2.anotacao = "Independência do Brasil";

        agenda2.mostraAnotacao();

    }
}
```

Aplicando o encapsulamento...

Atributos e métodos com modificadores de acesso

```
class Agenda {
        private int dia;
        private int mes;
        private String anotacao;

        public void anote(int d, int m, String nota){
        dia = d;
mes = m;
        anotacao = nota;
validaData();
        }

        private void validaData(){
if((dia < 1) || (dia> 31) || (mes< 1) || (mês > 12))
                dia=0;
                mês=0;
                anotacao = "anotação não inserida devido data in
                        válida";
        }
}

        public void mostraAnotacao(){
                System.out.println(dia+"/"+mês+" : "+ anotacao);
        }
}
```

Aplicando o encapsulamento...

Estes comandos causarão erros! Eles não são mais permitidos porque os atributos da classe Agenda agora são privados.

```
public class Principal {

    public static void main(String[] args) {

        Agenda agenda1 = new Agenda();
        Agenda agenda2 = new Agenda();

        agenda1.anote(12,10,"Dia da Criança");
        agenda2.anote(7,15,"Independência do Brasil");

        agenda1.mostraAnotacao();
        agenda2.mostraAnotacao();

        agenda2.dia = 7;  ERRO!
        agenda2.mes = 15; ERRO!
        agenda2.anotacao = "Independência do Brasil"; ERRO!

        agenda2.mostraAnotacao();

    }
}
```

Agora, o compilador já não aceita usar os comandos abaixo. Isso causaria erro! Ou seja, os atributos da classe Agenda agora são privados apenas para a uso da própria classe Agenda.

Aplicando o encapsulamento...

- Com essa modificação, só é possível inserir uma anotação na classe Agenda usando o método anote(). O método anote() garante que a data inserida para a anotação será validada com o método validaData(), que também é privado ao uso apenas da classe Agenda.
- SUGESTÃO: sugiro que você implemente este exemplo faça seus testes
- Bons Estudos!!

Atividade

- Modificar Agenda para utilizar as Classes “Date” ou “GregorianCalendar”