

Aula 08

Abstração e encapsulamento

Programação III

Prof. Augusto César Oliveira

augusto.oliveira@unicap.br

Na aula passada...

- Compreender o conceito de objetos;
- Definir o conceito de classes como modelos para criar objetos;
- Diferenciar atributos e métodos em uma classe;
- Explicar como os atributos e métodos são utilizados para modelar o comportamento de um objeto;

Na aula passada...

- Demonstrar como criar instâncias de objetos a partir de uma classe;
- Identificar o construtor padrão e construtores personalizados;
- Explicar como os construtores são utilizados para inicializar os atributos de um objeto.

O objetivo da aula de hoje...

- Compreender o conceito de abstração na programação orientada a objetos;
- Aplicar o princípio de encapsulamento, ocultando detalhes de implementação de classes;
- Fornecer acesso controlado aos atributos por meio de métodos getter e setter;
- Criar classes imutáveis com facilidade através de "records".

Programação orientada a objetos

- Vimos que com OO é possível **representar o mundo real com objetos**.
- Os objetos possuem:
 - **Atributos** (estado/características);
 - **Métodos** (comportamento/operações).
- Mas como definir **quais os atributos e métodos** devem compor um objeto?

1. Abstração

Abstração e encapsulamento

O que é abstração?

- É o processo de **extrair** de "**algo**" o que ele possa ter de mais interessante.
- Em POO, se refere à capacidade de **simplificar** e **modelar objetos** do mundo real em **representações de software**.
- Ou seja, a abstração permite que você se concentre nos **aspectos mais relevantes e essenciais** de um objeto, **ignorando os detalhes irrelevantes**.

O que você vê aqui?



Um homem e uma mulher?

- Pode até servir de **abstração** do que seria uma **homem** e uma **mulher** com o intuito do **sinalizar um banheiro**.
- Mas será que serve, por exemplo, como **abstração** para representar o **homem** e uma **mulher** em uma **aula de anatomia humana**?

Modelagem de objetos

- É a **atividade** em que se **define** os **objetos**, seus **atributos** e **métodos** que serão necessários para o domínio de uma **aplicação**.
- **Abstração** é o **conceito chave** para **modelar objetos**.

Modelando o objeto "TV"



Modelando o objeto "TV"

Atributos?

Métodos?

Modelando o objeto "TV"

Atributos?

1. Marca;
2. Polegadas;
3. Resolução;
4. Tecnologia da tela (CRT, Plasma, LCD, LED);
5. Possui WI-FI;
6. Possui HDMI;
7. Quantidade de portas HDMI.

Métodos?

Modelando o objeto "TV"

Atributos?

1. Marca;
2. Polegadas;
3. Resolução;
4. Tecnologia da tela (CRT, Plasma, LCD, LED);
5. Possui WI-FI;
6. Possui HDMI;
7. Quantidade de portas HDMI.

Métodos?

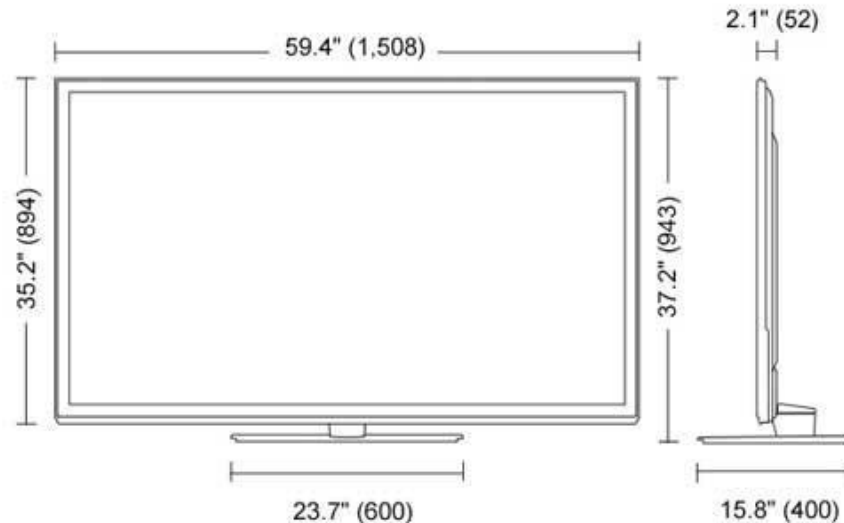
1. Ligar;
2. Desligar;
3. Aumentar volume;
4. Abaixar volume;
5. Mudar de canal;
6. Acessar menu.

Modelando o objeto "TV"

- E se você estiver desenvolvendo um **sistema que calcula o tamanho de caixas de papelão** para o **empacotamento** de um **produto**?
- ...será que a marca ou a quantidade de portas HDMI são **atributos relevantes**?

Modelando o objeto "TV"

Mais útil são as informações das **dimensões da TV:**
altura, largura, profundidade



Modelagem de objetos

- Para modelar objetos precisamos **entender o CONTEXTO** do sistema... **qual a finalidade?**
- **Entendido o contexto...** usamos nossa capacidade de **ABSTRAÇÃO** para **modelar os atributos e métodos dos objetos.**
- **Modelagem de objetos** é um trabalho **CRIATIVO!**
- **Não existe fórmula universal.**

Reduzindo a complexidade

- Quando **modelamos um objeto**, é necessário **entender todos os seus detalhes** para definir os **atributos** e **implementar os métodos**.
- **Quanto mais detalhes**, maior a complexidade.
- Será que **todos os detalhes** são realmente importantes?

2. Encapsulamento

Abstração e encapsulamento

O que é encapsulamento?

- É uma técnica utilizada **para não expor os detalhes internos de um objeto** para aqueles que somente irão utilizá-lo.
- **Vantagens:**
 - Reduz a complexidade para quem quer somente utilizar um objeto;
 - Reduz o acoplamento, que por consequência evita efeitos colaterais.

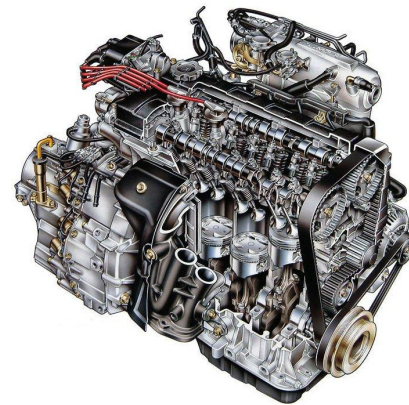
O que preciso para dirigir um carro?



1. **Ignição:** para ligar o carro;
2. **Câmbio:** para alterar força e velocidade;
3. **Acelerador:** para alterar velocidade do carro;
4. **Freio:** para breicar o carro.

Porém...

- Um carro não é somente ignição, marcha, acelerador e freio.
- E se para dirigir fosse necessário **saber todos os detalhes internos do carro?**



Estratégia de encapsulamento

- Os **engenheiros** que projetaram o carro **ENCAPSULARAM** todos os seus detalhes.
- Só é exposto aquilo que importa aos usuários do carro para evitar complexidade sem necessidade.

Como encapsular objetos em Java?

- Usar **modificadores de acesso privados** para os atributos.
 - **Atributos privados não podem ser acessados fora da classe do objeto;**
 - Evitamos que quem use a classe possa saber os detalhes internos do objeto.
- Implementar métodos **"get"** e **"set"** para cada um dos atributos.
 - Forma que permitimos alguém **obter** ou **modificar** o valor de um atributo
- **NOTA DE RODAPÉ:** na realidade encapsulamento é um conceito um mais amplo, mas trabalharemos somente nesse nível por enquanto.

Exemplo: classe ContaBancaria

modificadores de acesso privados

```
{ private String numero;  
  private double saldo;
```

```
String getNumero() {  
    return numero;  
}
```

```
void setNumero(String numero) {  
    this.numero = numero;  
}
```

```
double getSaldo() {  
    return saldo;  
}
```

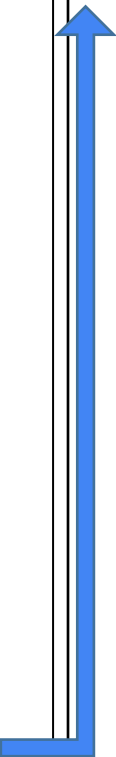
```
void setSaldo(double saldo) {  
    this.saldo = saldo;  
}
```

métodos get e set



Exemplo: classe ContaBancaria

```
public class ContaBancaria {  
    private String numero;  
    private double saldo;  
  
    public ContaBancaria(String numero,  
double saldo) {  
        this.numero = numero;  
        this.saldo = saldo;  
    }  
  
    void creditar(double valor) {  
        saldo = saldo + valor;  
    }  
  
    void debitar(double valor) {  
        saldo = saldo - valor;  
    }  
}
```



```
String getNumero() {  
    return numero;  
}  
  
void setNumero(String numero) {  
    this.numero = numero;  
}  
  
double getSaldo() {  
    return saldo;  
}  
  
void setSaldo(double saldo) {  
    this.saldo = saldo;  
}
```

Mas, é necessário mesmo todos esses gets e sets?

- Existe uma **longa discussão** na comunidade Java sobre gets e sets.
- Alguns argumentam que na maioria das vezes os gets e sets simplesmente **permitem o acesso aos atributos** e **nunca realizam nenhuma validação** ou outro tipo de **procedimento** e por isso **seriam desnecessários**.
- Os gets e sets ainda são um padrão na comunidade Java ([POJO/Beans](#)) e **difícilmente vão desaparecer**, mas as **novas versões da plataforma criaram um alternativa**.

3. Records

Abstração e encapsulamento

O que são "records"?

- É uma **espécie de tipo agregado de dados** heterogêneos imutáveis;
- Para facilitar a compreensão, podemos entender que um record:
 - Nos **permite criar um tipo** (assim como as classes);
 - Com **atributos** que são, automaticamente **privados**;
 - Possuem métodos de acesso a eles (**sem precisar criar gets**);
 - São **imutáveis** (depois que você “instancia” um record, **não pode mudar os valores dos atributos**).

Trabalhando com "records"

- Assim como as classes, um **"record"** precisa **estar dentro de um arquivo .java** que tenha exatamente o **mesmo nome do record**.
- Records **surgiram na versão 16** do Java.
- Ainda não temos certeza se, no futuro, os **records** se tornarão **amplamente adotados** e **substituirão** os **POJOs/Beans**.

Exemplo: records

```
public record Usuario(String login, String senha) {}
```

```
public class ClassePrincipal {  
    public static void main(String[] args) {  
        Usuario bruno = new Usuario("brunocartaxo", "123456");  
        System.out.println(bruno.login());  
        System.out.println(bruno.senha());  
    }  
}
```



Exemplo: records

Declaração do record

```
public record Usuario(String login, String senha) {}
```

Lista de atributos

```
public class ClassePrincipal {  
    public static void main(String[] args) {  
        Usuario bruno = new Usuario("brunocartaxo", "123456");  
        System.out.println(bruno.login());  
        System.out.println(bruno.senha());  
    }  
}
```

Métodos de acesso "gerados automaticamente"

Como seria uma classe que "funcionasse" como um record?

```
1 public class Usuario {  
2     private final String login;  
3     private final String senha;  
4  
5     public Usuario(String login, String senha) {  
6         this.login = login;  
7         this.senha = senha;  
8     }  
9  
10    public String login() {  
11        return this.login;  
12    }  
13  
14    public String senha() {  
15        return this.senha;  
16    }  
17 }
```

Atributos "final" garantem que uma vez "setados" no construtor, os valores não mudarão mais

4.

Considerações finais

Abstração e encapsulamento

O que aprendemos hoje?

- O conceito de abstração na programação orientada a objetos;
- Aplicar o princípio de encapsulamento, ocultando detalhes de implementação de classes;
- Fornecer acesso controlado aos atributos por meio de métodos getter e setter;
- Criar classes imutáveis com facilidade através de "records".

Próxima aula...

-

5.

Exercício de fixação

Teams



Abstração e encapsulamento

- **Link da atividade:** [clique aqui](#).



ATIVIDADE

Universidade Católica de Pernambuco

Professor: Augusto César Oliveira

Disciplina: Programação III / POO

Aluno(a): _____ data: __/__/__

Aula 08 - Abstração e encapsulamento

1. Desenvolva uma classe chamada "ContaBancaria" para um sistema bancário. Ela deve conter os seguintes atributos privados:
 - a. Nome do titular da conta.
 - b. Número da conta.
 - c. Saldo.

Implemente um construtor para a classe que recebe o nome do titular e o número da conta como parâmetros. O saldo inicial deve ser definido como zero.

Aula 08

Abstração e encapsulamento

Programação III

Prof. Augusto César Oliveira

augusto.oliveira@unicap.br