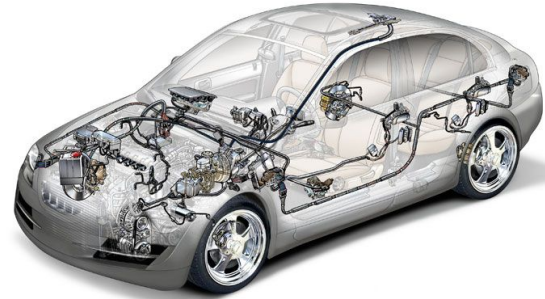

Paradigma orientado a objetos

Overview

- Conceitos básicos:
 - Modularização e abstração.
 - Objetos, classes, atributos e métodos.
 - Construtores, *getters* e *setters*.
- Características principais:
 - *Information hiding*/encapsulamento.
 - Modificadores de acesso.
 - Polimorfismo: *overloading* e *overriding*.
- Herança, composição, classes abstratas, interface e exceções.

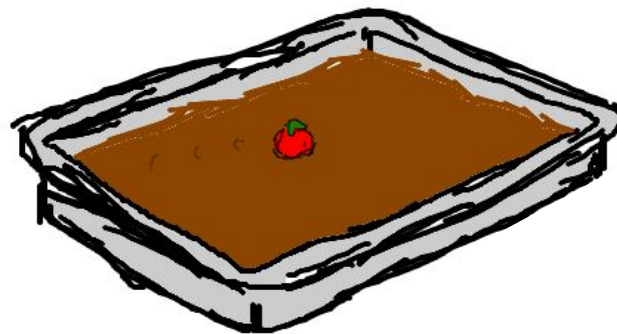
Características: cor,
tipo de pneu, tração,
placa...



Comportamentos:
resposta a aceleração,
resposta ao freio,
resposta a uma batida...

Conceitos básicos

- **Objeto:** agrupamento de características e comportamentos que representam um conceito.
- **Classe:** conceito que descreve a estruturação de objetos.
- **Atributo:** características.
- **Método:** comportamentos.



Modularização: geração de módulos gerenciáveis para desenvolvimento da solução geral.

Construtores, *getters* e *setters*

- **Construtor:** método especial que é responsável por inicializar os atributos do objeto que está sendo criado.
 - Construtor *default*: inicialização considerando valores *defaults*.
 - Um construtor pode reutilizar outro.
- *Information hiding*: princípio de proteção de determinadas informações → acesso/modificação consistente.
 - **Getters:** métodos que possibilitam o acesso aos atributos.
 - **Setters:** métodos que possibilitam a modificação dos atributos:

E o destrutor? **Garbage collection**, técnica utilizada por Java para desalocar objetos que não estão sendo mais utilizados pelo programa.

Exemplo carro:

```
public class Carro {  
    3 usages  
    private String cor;  
    3 usages  
    private String placa;  
  
    no usages  
    public Carro(String cor, String placa) {  
        this.cor = cor;  
        this.placa = placa;  
    }  
}
```

```
    public String getCor() {  
        return cor;  
    }  
    no usages  
    public String getPlaca() {  
        return placa;  
    }  
    no usages  
    public void setCor(String cor) {  
        this.cor = cor;  
    }  
    no usages  
    public void setPlaca(String placa) {  
        this.placa = placa;  
    }  
}
```

```
    public void acelerar() {  
        System.out.println("Acelerando");  
    }  
    no usages  
    public void freiar() {  
        System.out.println("Freiando");  
    }  
}
```

Visibilidade e pacotes

- Modificações de acesso:
 - **Public:** visibilidade em qualquer lugar.
 - **Protected:** visibilidade dentro do pacote ou nas subclasses da classe.
 - **Default:** visibilidade dentro do pacote.
 - **Private:** visibilidade somente dentro da classe.
- **Pacotes:** agrupamento de definições de classes relacionadas.
 - Maior nível de abstração.
 - Estruturação de sistemas de grande porte.

Coesão, encapsulamento e acoplamento

- **Coesão:** responsabilidade única.
 - Uma classe coesa faz bem uma única coisa.
 - Uma classe não pode ser duas coisas ao mesmo tempo.
- **Encapsulamento:** restrição de acesso às informações “secundárias”.
 - *Forte ligação com o os modificadores de acesso(public, private...)*
 - *Getters e setters* não são a definição de encapsulamento.
 - Lida com regras de negócio.
- **Acoplamento:** dependências entre módulos.
 - Dependências existem, mas devem ser reduzidas.

Herança e composição

- **Herança** possibilita uma hierarquia de classes que herdem propriedades e comportamentos e definem propriedades e/ou comportamentos → “é um”.
 - Preservação da semântica (comportamento).
 - Substituição: objetos da subclasse podem ser usados no lugar de objetos da superclasse.
 - Reuso de código e extensibilidade.
- **Composição**: extensibilidade com a utilização de uma classe dentro de outra classe → “tem um”.

Classes abstratas e interface

- Classe abstrata:
 - Objetos com implementações compartilhadas.
 - Definição de novas classes através da herança simples de código.
 - Ao menos um método abstrato e geralmente ao menos um concreto.
- Interface: padrão de serviço, todos os métodos disponíveis e suas assinaturas.
 - Objetos com implementações diferentes.
 - Definição de novas interfaces através da herança múltipla de assinatura.
 - Todos os métodos públicos e abstratos, sem atributos.

Exceções (*try-catch-finally*)

- Notificação e tratamento de erros:
 - Fornece informações extras sobre as falhas.
 - Distingue diferentes tipos de falhas.
- Fluxo de controle e código passados para chamadas acima.

Polimorfismo

- **Overloading**: sobrecarga.
- **Overriding**: redefinição e especialização.



```
int calculaSoma(int a){  
    return this.valor + a;  
}  
int calculaSoma(){  
    return this.valor + 5;  
}
```



```
public class Exemplo{  
    String mostraExemplo(){  
        return "Meu exemplo";  
    }  
}  
public class HerdaExemplo extends Exemplo{  
    @Override  
    String mostraExemplo(){  
        return "Sobrescrevi meu exemplo";  
    }  
}
```

Generics - início

- Permitem que as classes e métodos possam ser escritos para **trabalhar com qualquer tipo de objeto**.
- Os tipos genéricos são **definidos em tempo de compilação**
 - O compilador verifica o tipo de objeto que está sendo utilizado e gera o código apropriado em tempo de execução.
- Benefícios
 - Prevenir reescrita de código
 - Torna o código mais flexível
 - Mais fácil de manter
 - Segurança dos tipos de dados. Erros são detectados em tempo de compilação, antes da execução

Generics - exemplo

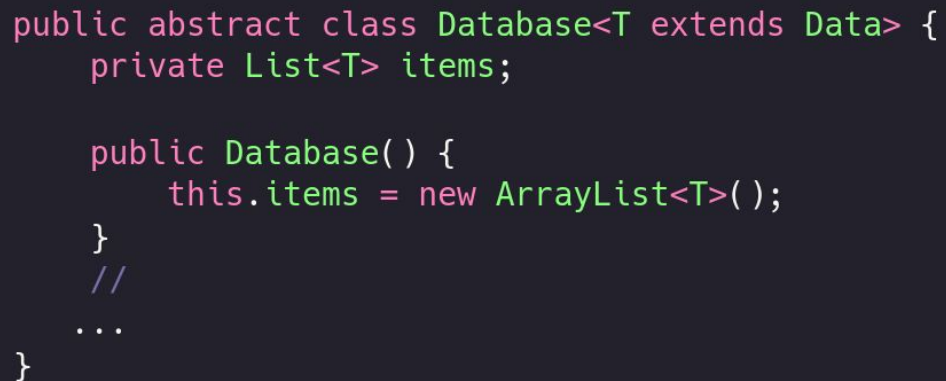
```
public static void imprimirVetorDouble(double[] vetor) {  
    for (double elemento: vetor) {  
        System.out.print(elemento + " ")  
    }  
    System.out.println();  
}
```

```
public static <T> void imprimirVetor(T[] vetor) {  
    for (T elemento : vetor) {  
        System.out.print(elemento + " ");  
    }  
    System.out.println();  
}
```

Generics - detalhes

- Sintaxe:
 - de tipo genérico é indicada pelo uso do sinal < > com um identificador de tipo genérico. Por exemplo, <T> ou <E>.
 - Podem herdar propriedades. Como T extends Data, sendo Data uma classe instanciada
- Curiosidades
 - Os collections da API do Java (ArrayList, HashMap, etc.) utilizam generics para permitir que armazenem objetos de qualquer tipo.
 - **Erasure**

Generics - exemplo



```
public abstract class Database<T extends Data> {  
    private List<T> items;  
  
    public Database() {  
        this.items = new ArrayList<T>();  
    }  
    //  
    ...  
}
```