Ciência da Computação

Programação Orientada a Objetos

Prof. Luciano Rodrigo Ferretto

Preparação para a aula

- Para essa aula iremos continuar trabalhando no nosso Sistema de Gestão de Bibliotecas. Caso você não o tenha funcionando, baixe-o do repositório Git da turma:
- Baixe as três classes (Livro, Biblioteca e Main)

https://github.com/luciano-ferretto/OrgAbsProg_OO_202501/tree/main/Aula08/SisBiblio

Abra o VSCode no diretório que contenha somente estas três classes somente.

Herança em Java

Herança

- **Herança** é o princípio que permite que uma classe (subclasse) herde os atributos e métodos de outra classe (superclasse), facilitando a reutilização de código, a organização hierárquica e a promoção de relações entre objetos.

- Quando você possui duas ou mais classes com atributos e métodos em comum, a herança facilita a criação de uma estrutura hierárquica, onde uma classe "pai" define os elementos comuns que serão herdados pelas classes "filhas".

Ciência da Computação

Herança

Na programação orientada a objetos, a relação de herança entre classes é a relação em que uma classe é do tipo **é uma**, e não do tipo **tem uma**. Esta é uma das confusões recorrentes na construção de programas em orientação a objetos.

Para ilustrar essa diferença, observe a Figura 1 e veja que, neste exemplo, temos um tipo de relação **é uma**, pois o objeto da classe `Cachorro`, assim como o objeto da classe `Gato`, é, por herança, um tipo de objeto da classe `Animal`.

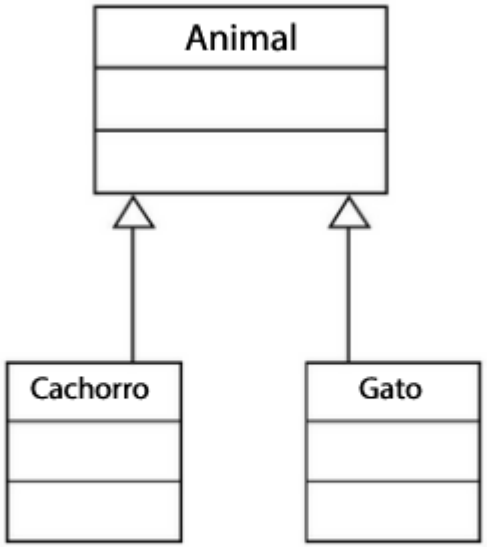


Figura 1. Representação de herança entre classes.

Compare, agora, a relação apresentada na Figura 2, que representa uma relação do tipo **tem uma**. Perceba que, neste caso, uma relação de herança entre as classes `Estado` e `Cidade` não faz sentido, visto que um estado possui cidades, mas uma cidade não é um estado.

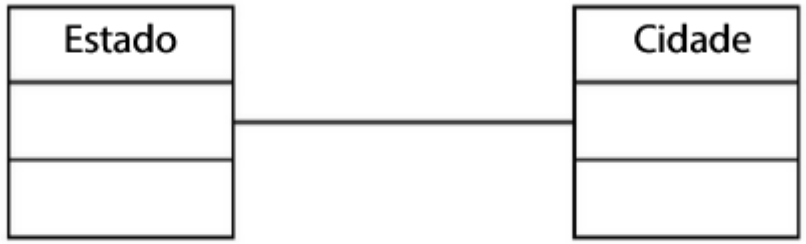


Figura 2. Representação da relação do tipo **tem uma** entre classes.

Fonte: SILVA, F. M.; LEITE, M. C. D.; OLIVEIRA, D. B. **Paradigmas de programação**. Porto Alegre: Sagah, 2019.

Computação

Herança em Java

- Em Java utilizamos a palavra-chave **extends** para indicar que uma Classe herda as informações de outra.

- No exemplo do próximo slide, a classe *LivroFisico* e a classe *LivroDigital* herdam os membros (atributos e métodos) da classe *Livro*.
- Podemos chamar a classe *Livro* de "**Classe Mãe**", "**Classe Pai**" ou "**Super Classe**".
- As classes *LivroFisico* e *LivroDigital* podem ser chamadas de "**Classes Filhas**" ou "**Sub-classes**"

Ciência da Computação

```java
public class Livro {
    private String titulo;
    private String autor;
    private int anoPublicacao;
    private int nPaginas;
```

```java
public class LivroFisico extends Livro {
    private int nExemplares;
    private String dimensoes;
```

```java
public class LivroDigital extends Livro {
    private double tamanhoArquivo;
    private String formatoArquivo;
```

Herança em Java

- Quando utilizamos a herança em Java, os objetos instanciados a partir de Sub-classes herdam todos os membros declarados na Super Classe.

- **Porém, é importante atentar para os modificadores de acesso.**

- No exemplo anterior, objetos do tipo *LivroFisico* e *LivroDigital* irão possuir os atributos *titulo*, *autor*, *anoPublicacao* e *nPaginas*, porém os métodos das Sub-classes só terão acesso à esses atributos através dos métodos **getters and setters**, pois os atributos estão com o modificador **"private"**.

Ciência da Computação

Classe Object

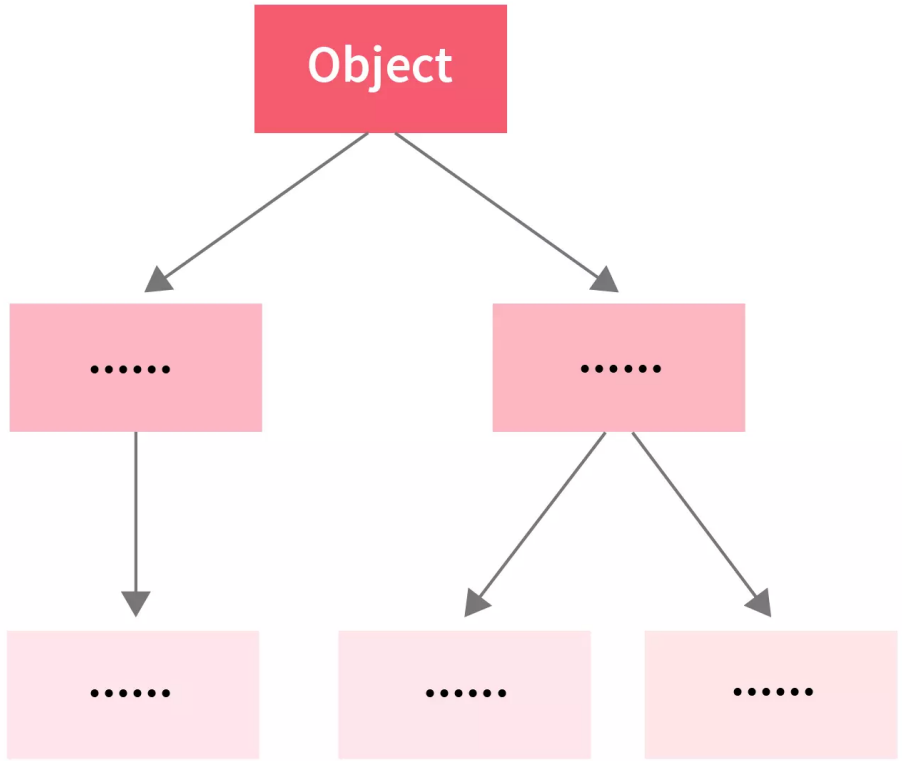
Classe Object

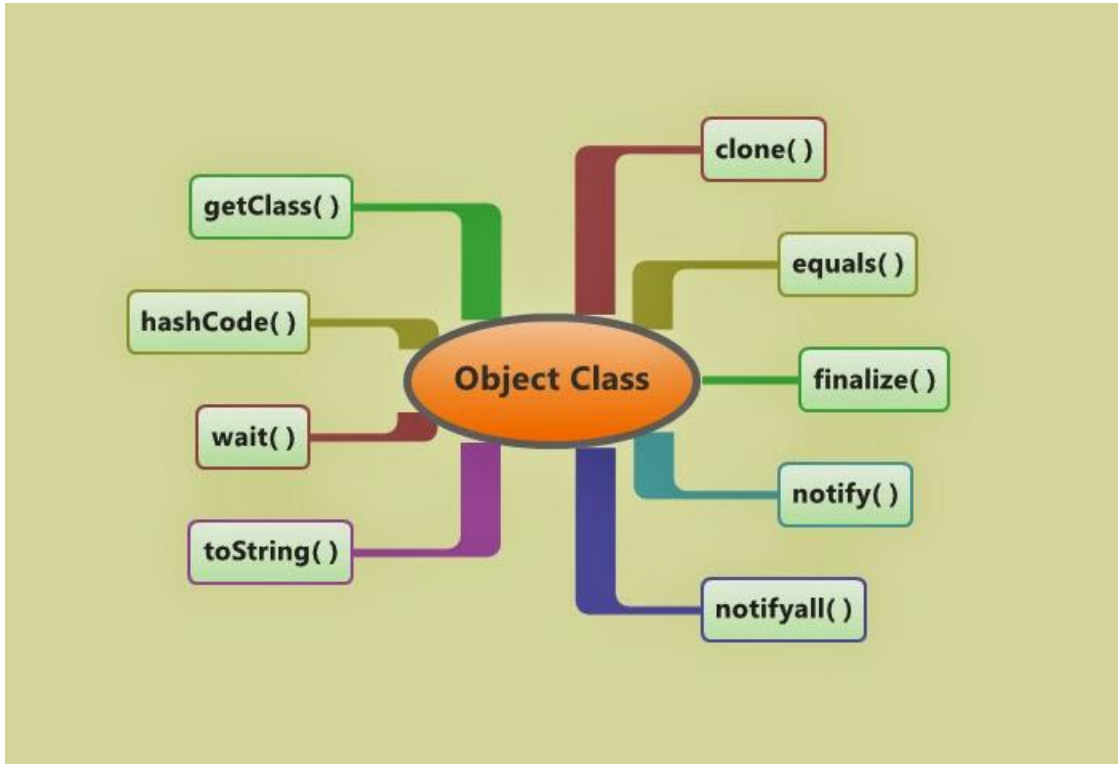
- A classe ***Object*** em Java é a classe mãe de todas as classes.

- Ela está localizada no pacote *java.lang* e é a superclasse **implícita** de qualquer outra classe criada no Java, ou seja, toda classe herda de Object mesmo que não seja explicitamente declarado.

- Isso significa que todos os objetos em Java possuem alguns métodos básicos que são herdados da classe Object.

Object em Java





Alguns Métodos da Classe Object

- **getClass():** Retorna um objeto da classe Class que contém informações sobre a classe do objeto atual.
- **hashCode():** Retorna um valor numérico (inteiro) que representa o código hash do objeto.
- **toString():** Retorna uma representação em forma de String do objeto. O método por padrão retorna o nome da classe seguido de um código hash hexadecimal do objeto

```java
package java.lang;
public class Object {
    public Object() {}
    public final native Class<?> getClass();
    public native int hashCode();
    public boolean equals(Object obj) {
        return (this == obj);
    }
    protected native Object clone() throws CloneNotSupportedException;
    public String toString() {
        return getClass().getName() + "@" + Integer.toHexString(hashCode());
    }
    public final native void notify();
    public final native void notifyAll();
    public final native void wait(long timeout) throws InterruptedException;
    public final void wait(long timeout, int nanos) throws InterruptedException {
        if (timeout < 0) {
            throw new IllegalArgumentException("timeout value is negative");
        }
        if (nanos < 0 || nanos > 999999) {
            throw new IllegalArgumentException("nanosecond timeout value out of range");
        }
        if (nanos > 0) {
            timeout++;
        }
        wait(timeout);
    }
    public final void wait() throws InterruptedException {
        wait(0);
    }
    protected void finalize() throws Throwable {}
}
```

Object em outras linguagens

- **C#:** A classe raiz em C# é a classe **Object**, assim como em Java. Todas as classes em C# derivam implicitamente ou explicitamente da classe **Object**.

- **Python**: Em Python, a classe raiz é chamada de **object**, com letra minúscula. Todas as classes em Python derivam implicitamente ou explicitamente da classe **object**.

- **Ruby**: A classe raiz em Ruby é chamada de **BasicObject**. Todas as outras classes em Ruby derivam implicitamente ou explicitamente da classe **Object**, que é uma subclasse de **BasicObject**.

- **C++**: Em C++, a classe raiz é a classe **Object**, embora essa classe não seja explicitamente definida na linguagem. Todas as classes em C++ derivam implicitamente ou explicitamente de uma classe de sistema chamada **type_info**, que por sua vez é derivada de uma classe chamada **Object**.

Polimorfismo em Java

Polimorfismo

- **Polimorfismo** é o princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas.

- Em termos simples, isso significa que diferentes objetos, de uma mesma Super Classe, podem responder de maneira única a uma mesma mensagem ou método.

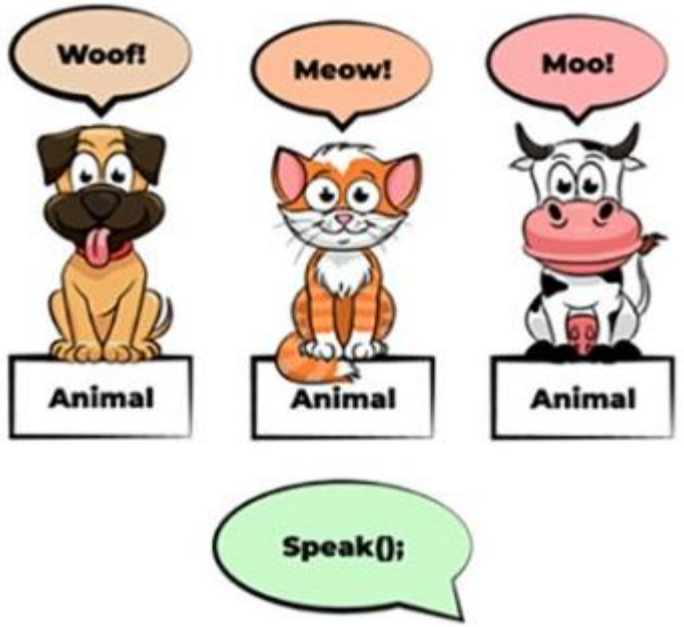
Ciência da Computação

Polimorfismo

O polimorfismo na programação orientada a objetos permite que uma ou mais classes derivadas de uma mesma superclasse possam invocar métodos que possuam uma mesma assinatura, mas com comportamentos diferenciados para cada classe derivada, utilizando, para isso, uma referência a um objeto da superclasse.

A definição de polimorfismo é mais um dos recursos da orientação a objetos que possibilita que um comportamento encontrado na realidade seja aplicado à programação. Na natureza, existem animais que são capazes de modificar sua forma ou comportamento para atender a determinada situação, e é isto que o polimorfismo possibilita na programação orientada a objetos.

Segundo Tucker e Noonan (2009, p. 323), "em linguagens orientadas a objetos, polimorfismo refere-se à ligação tardia de uma chamada a uma ou várias diferentes implementações de um método em uma hierarquia de herança".



Fonte: SILVA, F. M.; LEITE, M. C. D.; OLIVEIRA, D. B. **Paradigmas de programação**. Porto Alegre: Sagah, 2019.

Ciência da Computação

Polimorfismo – Tipos

Para Lima (2014), são dois os tipos mais recorrentes de polimorfismo na programação orientada a objetos:

a) polimorfismo estático ou sobrecarga de método;
b) polimorfismo dinâmico ou sobrescrita de método.

Fonte: SILVA, F. M.; LEITE, M. C. D.; OLIVEIRA, D. B. **Paradigmas de programação**. Porto Alegre: Sagah, 2019.

Ciência da Computação

Polimorfismo Estático – Sobrecarga de métodos

- O **polimorfismo estático**, também conhecido como sobrecarga de método ou polimorfismo de compile-time, é um conceito de POO em que uma classe pode ter vários métodos com o mesmo nome, mas com diferentes parâmetros.
- Isso permite que você chame o mesmo método de diferentes maneiras, dependendo dos tipos de argumentos passados.
- No polimorfismo estático, a decisão de qual método será chamado é tomada em tempo de compilação, com base na assinatura dos métodos (ou seja, nos tipos e na quantidade de parâmetros).

Ciência da Computação

Polimorfismo Estático – Sobrecarga de métodos


```java
public List<Livro> pesquisar(String titulo){
    return pesquisar(titulo, null);
}
public List<Livro> pesquisar(String titulo, String autor){
    List<Livro> livrosEncontrados = new ArrayList<>();

    for (Livro livro : acervo) {
        if (livro.getTitulo().toLowerCase().contains(titulo.toLowerCase())) {
            if (autor == null || livro.getAutor().toLowerCase().contains(autor.toLowerCase()))
                livrosEncontrados.add(livro);
        }
    }

    return livrosEncontrados;
}
```

Ciência da Computação

Polimorfismo Dinâmico – Sobrescrita de métodos

- O **polimorfismo dinâmico**, também conhecido como polimorfismo de tempo de execução ou **sobrescrita de método**, permite que objetos de diferentes classes, que compartilham uma relação de herança, respondam de maneira específica a chamadas de métodos comuns.

- No polimorfismo dinâmico, a decisão sobre qual método será executado é tomada em tempo de execução, com base no tipo real do objeto que está sendo referenciado, em vez de ser determinada em tempo de compilação, como acontece com o polimorfismo estático.

Ciência da Computação

Polimorfismo Dinâmico – Sobrescrita de métodos

- A sobrescrita de método é uma forma de implementar o polimorfismo dinâmico, na qual uma classe filha (subclasse) **redefine um método** previamente definido na classe pai (superclasse).

- A subclasse fornece sua própria implementação do método, alterando seu comportamento sem mudar a assinatura do método (nome, parâmetros e tipo de retorno).

- Isso é especialmente útil para permitir que objetos de diferentes subclasses se comportem de maneira específica, enquanto ainda podem ser tratados de maneira polimórfica.

Ciência da Computação

Polimorfismo Dinâmico – Sobrescrita de métodos


```java
@Override
public String toString() {
    String descricao = "Título: "
                + this.getTitulo() + " - Autor: "
                + this.getAutor() + " - Ano: "
                + this.getAnoPublicacao();
    return descricao;
}
```

```java
@Override
public String toString() {
    String descricao = "Título: "
            + this.getTitulo() + " - Autor: "
            + this.getAutor() + " - Ano: "
            + this.getAnoPublicacao();
    return "Livro Físico\n" + descricao
            + "\n Num. de Exemplares: " + this.nExemplares;
}
```

```java
@Override
public String toString() {
    String descricao = "Título: "
            + this.getTitulo() + " - Autor: "
            + this.getAutor() + " - Ano: "
            + this.getAnoPublicacao();
    return "Livro Digital\n" + descricao
            + "\n Formato do Arquivo: " + this.formatoArquivo;
}
```

Ciência da Computação

Polimorfismo Dinâmico – Sobrescrita de métodos

- Neste exemplo temos uma pesquisa que retornará objetos do tipo *Livro* que podem tanto ser instâncias da classe *LivroFisico* quanto da classe *LivroDigital*.
- Ao invocar o método *toString()* a JVM irá decidir quais instruções executar com base no tipo real do objeto instanciado.
- Ou seja, em tempo de compilação, **não** sabemos se será o método da classe *LivroFisico*, *LivroDigital*, *Livro* ou até mesmo da *Object* que será executado

```
private static void listar() {
    limparTela();
    List<Livro> livros = biblio.pesquisarTodos();
    livros.sort(Comparator.comparing(Livro::getTitulo));

    System.out.println("======== LISTA DE LIVROS =========");
    for (Livro livro : livros) {
        System.out.println(livro.toString());
        System.out.println("-----------------------------------");
    }
    aguardarEnter();
}
```