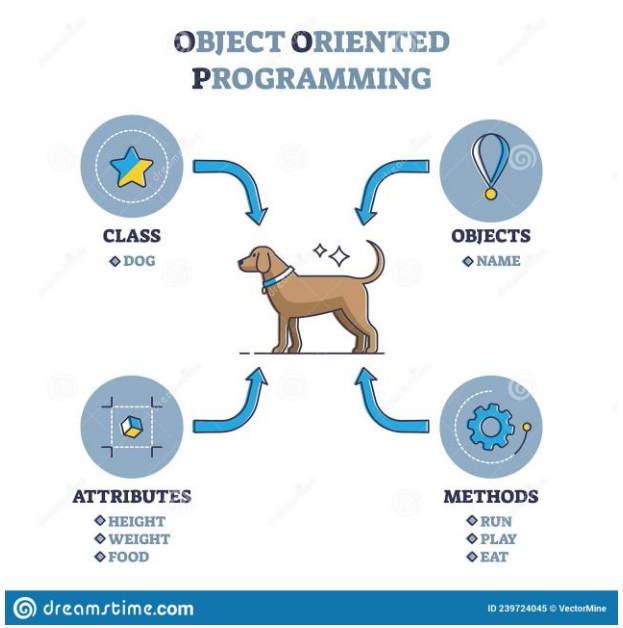
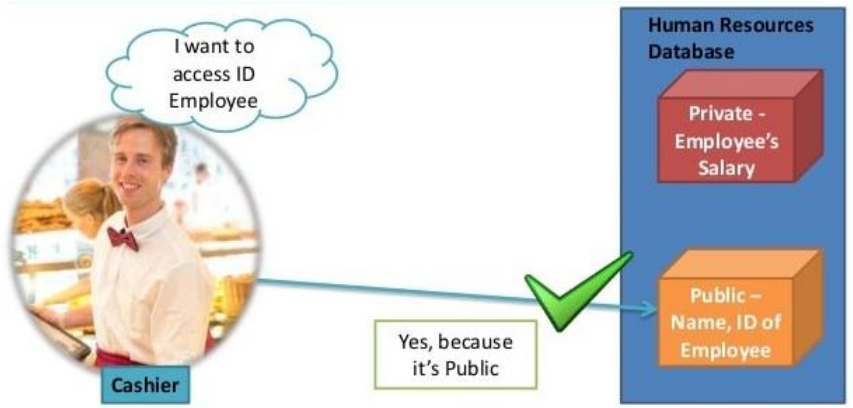
Ciência da Computação

Programação Orientada a Objetos

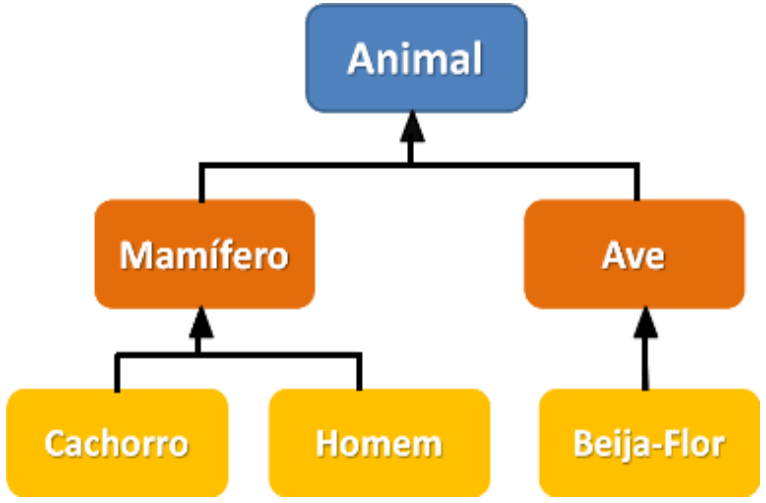
Prof. Luciano Rodrigo Ferretto

Vamos relembrar um pouco


OBJECT ORIENTED PROGRAMMING
CLASS
DOG
OBJECTS
NAME
ATTRIBUTES
HEIGHT
WEIGHT
FOOD
METHODS
RUN
PLAY
EAT
dreamstime.com
ID 239724045 © VectorMine


I want to access ID Employee
Cashier
Yes, because it's Public
Human Resources Database
Private - Employee's Salary
Public – Name, ID of Employee


1 Abstração
2 Encapsulamento
3 Herança
4 Polimorfismo


Animal
Mamífero
Ave
Cachorro
Homem
Beija-Flor


"Now Speak!"
"Quack"
"Woof"
"Meow"

Abstração

- **Abstração** é o processo de identificar e definir características essenciais de objetos do mundo real, representando essas características em forma de **classes** e **objetos**.

- Uma **classe** é uma estrutura que define atributos e métodos representando um tipo de objeto, fornecendo um modelo para criar instâncias desse tipo.
- Um **objeto** é uma instância de uma classe, caracterizado por seus atributos e comportamentos definidos na classe.
  - Ele pode interagir com outros objetos por meio de métodos e troca de dados.

Objeto é uma **instância** concreta de uma classe na POO.

Ciência da Computação

Encapsulamento

- **Encapsulamento** é a prática de ocultar os detalhes internos de um objeto, expondo apenas as operações relevantes para manipular esses detalhes. Isso promove a modularidade, a segurança e a manutenibilidade do código.

- Na maioria das linguagens de programação orientada a objetos, o encapsulamento é alcançado através da definição de **modificadores de acesso.**
  - **public, private**, **protected e default** (sem modificador)

Ciência da Computação

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

Ciência da Computação

Polimorfismo

- **Polimorfismo** é um princípio da programação orientada a objetos que permite que funções ou métodos assumam comportamentos diferentes, dependendo do contexto.

- Com o polimorfismo, é possível reutilizar e adaptar comportamentos em diferentes contextos, permitindo que classes e métodos sejam usados de maneira flexível, sem perder a consistência da interface ou lógica comum.

- Ele se divide em dois tipos principais:
  - Polimorfismo Estático
  - Polimorfismo Dinâmico

Ciência da Computação

Polimorfismo

- **Polimorfismo dinâmico** (em tempo de execução): Permite que objetos de sub-classes de uma mesma classe base invoquem métodos que têm a mesma assinatura (nome e parâmetros), mas que se comportam de maneira diferente, de acordo com o tipo específico do objeto instanciado.
  - Sobrescrita de métodos

- **Polimorfismo estático** (em tempo de compilação): Refere-se à capacidade de métodos com o mesmo nome, mas com assinaturas diferentes (número ou tipos de parâmetros), serem utilizados de formas distintas.
  - Sobrecarga de métodos.

Ciência da Computação

Polimorfismo - Estático x Dinâmico

Estático	Dinâmico
Também conhecido como **Sobrecarga de Método.**	Também conhecido como **Sobrescrita de Método.**
Ocorre quando há vários métodos com o mesmo nome, mas com diferentes assinaturas (diferentes tipos ou números de parâmetros) dentro da mesma classe.	Ocorre quando uma subclasse fornece uma implementação específica de um método que já é definido na sua superclasse.
A ligação (binding) das chamadas de método ao código correspondente é feita em tempo de compilação.	A ligação (binding) das chamadas de método ao código correspondente é feita em tempo de execução.
Permite que métodos com o mesmo nome (assinaturas diferentes) possam ser chamados, e a versão correta do método é invocada de acordo com os parâmetros fornecidos na chamada	Permite que o mesmo método (mesma assinatura) possa ser chamado em diferentes tipos de objetos, e a versão correta do método é invocada de acordo com o tipo real do objeto em tempo de execução.

Pilares da Programação Orientada a Objetos

- **Abstração**:
  - É o processo de identificar e definir características essenciais de objetos do mundo real, representando essas características em forma de **classes** e **objetos**.
- **Encapsulamento**:
  - É a prática de **ocultar os detalhes internos** de um objeto (classe), **expondo apenas as operações relevantes** para manipular esses detalhes. Isso promove a modularidade, a segurança e a manutenibilidade do código.
- **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.
- **Polimorfismo**:
  - Significa "muitas formas". O polimorfismo permite que objetos de diferentes classes sejam tratados como objetos de uma mesma classe porém com comportamentos diferentes. Isso facilita a criação de código mais flexível e extensível.

Ciência da
Computação

Vamos ver esses conceitos no Java...

"Imagine que estamos criando um sistema para um ***zoológico digital***. Precisamos cadastrar os animais, definir suas características e comportamentos, mas também garantir que certos comportamentos não mudem, e que existam regras gerais para algumas espécies..."

Abstract – Classes e métodos abstratos

- Uma classe abstrata **não pode ser instanciada, apenas herdada.**
- Sua principal função é servir como base para outras classes.
- Pode conter métodos abstratos e não abstratos.

- Métodos abstratos **não possuem implementação**, e devem ser **obrigatoriamente implementados (@override) nas subclasses.**
- Métodos abstratos são uma forma de garantir que todas as subclasses tenham um determinado comportamento definido

Abstract – Classes e métodos abstratos

```java
public abstract class Animal {
    String nome;
    Animal(String nome) {
        this.nome = nome;
    }
    abstract void fazerSom();
}
```

```java
public class Cachorro extends Animal {
    Cachorro(String nome) {
        super(nome);
    }
    @Override
    void fazerSom() {
        System.out.println("Au au!");
    }
}
```

```java
public class Gato extends Animal{
    Gato(String nome) {
        super(nome);
    }
    @Override
    void fazerSom() {
        System.out.println("Miauuuu!");
    }
}
```

Final – Classes, métodos e atributos finais

- Uma classe final **não pode ser herdada.**
- Um método final **não pode ser sobrescrito.**
- Um atributo (variável) final **não pode ter seu valor alterado.**

```java
public final class ExemploFinalClasse {
    // Conteúdo da classe
}

// A seguinte linha gera um erro, pois não é possível estender uma classe final
class SubClasse extends ExemploFinalClasse { }
```

```java
public class ExemploFinalMetodo {
    public final void metodoFinal() {
        System.out.println("Este método é final e não pode ser sobrescrito.");
    }
}
class subClasse extends ExemploFinalMetodo {
    //Essa sobrescrita gera um erro, pois o método da
    //super classe é final
    @Override
    public void metodoFinal() {
        System.out.println("Erro ao sobrescrever o método");
    }
}
```

```java
public class ExemploFinalVariavel {
    public static void main(String[] args) {
        final int constante = 10;
        constante = 20; // Isso gera um erro, pois a variável é final
        System.out.println(constante);
    }
}
```

Modificador **Static** em Java

Entendendo membros e métodos estáticos

O que é static?

- É um modificador usado para indicar que um membro pertence à classe, e não à instância
- Principais usos:
  - Variáveis (atributos)
  - Métodos

Variáveis Estáticas

```
public class Contador {
    static int total = 0;

    public Contador() {
        total++;
    }
}
```

- **total** é compartilhado entre todas as instâncias
- Ao criar vários objetos, a variável total é incrementada globalmente
- Não importa quantos objetos do tipo “Contador” existam, a variável **total** só existirá uma vez.

Métodos Estáticos

- Métodos estáticos **não** acessam atributos ou métodos de instância diretamente
- Podem ser chamados sem criar um objeto
- Para chamar o método, podemos usar o nome da Classe

```
public class Matematica {
    public static int somar(int a, int b) {
        return a + b;
    }
}
```

Método **main** é estático!

- O **main** precisa ser estático para que a JVM possa executar o programa sem criar um objeto da classe

```
public class Programa {
    public static void main(String[] args) {
        System.out.println("Olá, mundo!");
    }
}
```

Boas Práticas

- Use **static** para constantes e utilitários
- Evite acessar variáveis estáticas com objetos (use o nome da classe)
- Cuidado com estados globais → pode gerar dependências ruins