Aula: Herança e Abstração

Objetivo da Aula

Compreender os conceitos de **herança** e **abstração** na POO, como eles funcionam em Java, e como utilizá-los para criar sistemas organizados e extensíveis.

O Que é Herança?

Herança é um mecanismo que permite que uma classe (subclasse) reutilize ou sobrescreva comportamentos e atributos de outra classe (superclasse). Ela promove o reaproveitamento de código e facilita a manutenção.

Características:

1. Uma classe filha **herda** atributos e métodos de sua classe pai.
2. Permite **especialização**: a subclasse pode adicionar ou modificar comportamentos.
3. Utiliza a palavra-chave extends.

Exemplo de Herança

```java
class Animal {
    String nome;

    public void dormir() {
        System.out.println(nome + " está dormindo.");
    }
}

class Cachorro extends Animal {
    public void latir() {
        System.out.println(nome + " está latindo: Au au!");
    }
}

public class TesteHeranca {
    public static void main(String[] args) {
        Cachorro cachorro = new Cachorro();
        cachorro.nome = "Rex";
        cachorro.dormir();
        cachorro.latir();
    }
}
```

Saída:

```
Rex está dormindo.
Rex está latindo: Au au!
```

Ilustração:

- Classe pai: Animal
    - Atributos: nome
    - Métodos: dormir()
- Classe filha: Cachorro
    - Herda nome e dormir()
    - Adiciona o método latir()

Vantagens da Herança

1. **Reaproveitamento de Código**: Evita repetição ao reutilizar atributos e métodos.
2. **Organização**: Facilita a criação de hierarquias lógicas.
3. **Flexibilidade**: Permite a sobrescrita de métodos para comportamentos específicos.

O Que é Abstração?

Abstração é o processo de esconder os detalhes de implementação de um objeto, expondo apenas sua funcionalidade essencial. Em Java, isso é feito por meio de **classes abstratas** e **interfaces**.

Classe Abstrata

Uma **classe abstrata** é uma classe que não pode ser instanciada diretamente. Serve como modelo para outras classes e pode conter:

- Métodos concretos (implementados).
- Métodos abstratos (sem implementação).

Exemplo:

```java
abstract class Animal {
    String nome;

    public abstract void emitirSom();

    public void dormir() {
        System.out.println(nome + " está dormindo.");
    }
}

class Cachorro extends Animal {
    @Override
    public void emitirSom() {
        System.out.println(nome + " está latindo: Au au!");
    }
}

class Gato extends Animal {
    @Override
    public void emitirSom() {
        System.out.println(nome + " está miando: Miau!");
    }
}

public class TesteAbstracao {
    public static void main(String[] args) {
        Animal cachorro = new Cachorro();
        cachorro.nome = "Rex";
        cachorro.emitirSom();
        cachorro.dormir();

        Animal gato = new Gato();
        gato.nome = "Mingau";
        gato.emitirSom();
        gato.dormir();
    }
}
```

Saída:

```
Rex está latindo: Au au!
Rex está dormindo.
Mingau está miando: Miau!
Mingau está dormindo.
```

Ilustração:

- Classe abstrata: Animal
    - Atributo: nome
    - Método concreto: dormir()
    - Método abstrato: emitirSom()
- Classes concretas: Cachorro, Gato
    - Implementam o método emitirSom().

Diferenças entre Herança e Abstração

Aspecto	Herança	Abstração
Conceito	Reutilização de atributos e métodos	Esconde detalhes, expondo funcionalidade
Implementação	Utiliza classes concretas	Utiliza classes abstratas ou interfaces
Exemplo	Classe Cachorro herda de Animal	Classe Animal é abstrata

---

Exercício Prático

Crie um sistema de veículos que inclua:

1. Uma classe abstrata Veiculo com os métodos:
    - mover() (abstrato).
    - abastecer() (implementado).
2. Classes concretas Carro e Moto que herdam de Veiculo e implementam mover().
3. Implemente uma classe TesteVeiculo para testar o polimorfismo com os diferentes tipos de veículos.

```java
abstract class Veiculo {
    public abstract void mover();

    public void abastecer() {
        System.out.println("Veículo abastecido.");
    }
}

class Carro extends Veiculo {
    @Override
    public void mover() {
        System.out.println("Carro está se movendo.");
    }
}

class Moto extends Veiculo {
    @Override
    public void mover() {
        System.out.println("Moto está se movendo.");
    }
}

public class TesteVeiculo {
    public static void main(String[] args) {
        Veiculo carro = new Carro();
        Veiculo moto = new Moto();

        carro.mover();
        carro.abastecer();

        moto.mover();
        moto.abastecer();
    }
}
```

Conclusão

A **herança** e a **abstração** são pilares essenciais da POO que permitem criar sistemas organizados e extensíveis. Enquanto a herança promove o reaproveitamento de código, a abstração garante que possamos focar nas funcionalidades principais, escondendo os detalhes de implementação. Estas práticas são fundamentais para criar sistemas robustos, escaláveis e fáceis de manter.