1. O que é herança?

Herança é um recurso da programação orientada a objetos que permite que uma classe aproveite características de outra classe.

A classe que fornece os atributos e métodos é chamada de **classe pai** ou **superclasse**.

A classe que herda é chamada de **classe filha** ou **subclasse**.

Exemplo simples:

```
class Animal {
  void dormir() {
    print('O animal está dormindo');
  }
}

class Cachorro extends Animal {
  void latir() {
    print('O cachorro está latindo');
  }
}

void main() {
  Cachorro cachorro = Cachorro();

  cachorro.dormir();
  cachorro.latir();
}
```

Saída:

```
O animal está dormindo
O cachorro está latindo
```

---

2. Palavra-chave `extends`

Em Dart, usamos `extends` para indicar que uma classe herda de outra.

```
class Pessoa {
  String nome = '';

  void apresentar() {
    print('Olá, meu nome é $nome');
  }
}

class Aluno extends Pessoa {
  String curso = '';

  void estudar() {
    print('$nome está estudando $curso');
  }
}
```

---

3. Exemplo completo

```
class Funcionario {
  String nome;
  double salario;

  Funcionario(this.nome, this.salario);

  void mostrarDados() {
    print('Nome: $nome');
    print('Salário: R\$ $salario');
  }
}

class Gerente extends Funcionario {
  String setor;

  Gerente(String nome, double salario, this.setor)
      : super(nome, salario);

  void mostrarSetor() {
    print('Setor: $setor');
  }
}

void main() {
  Gerente gerente = Gerente('Carlos', 5000, 'Financeiro');

  gerente.mostrarDados();
  gerente.mostrarSetor();
}
```

---

4. O que é `super`?

A palavra `super` é usada para acessar atributos, métodos ou construtores da classe pai.

Exemplo:

```
class Veiculo {
  String marca;

  Veiculo(this.marca);

  void ligar() {
    print('O veículo está ligado');
  }
}

class Carro extends Veiculo {
  String modelo;

  Carro(String marca, this.modelo) : super(marca);

  void mostrarInfo() {
```

```
    print('Marca: $marca');
    print('Modelo: $modelo');
  }
}
```

Aqui:

`super(marca)`

chama o construtor da classe `Veiculo`.

---

5. Sobrescrita de métodos

A classe filha pode alterar o comportamento de um método herdado.

Para isso, usamos `@override`.

```
class Animal {
  void emitirSom() {
    print('O animal faz um som');
  }
}

class Gato extends Animal {
  @override
  void emitirSom() {
    print('O gato mia');
  }
}

void main() {
  Gato gato = Gato();
  gato.emitirSom();
}
```

Saída:

```
O gato mia
```

---

6. Usando `super` com métodos

A classe filha pode chamar o método original da classe pai e depois adicionar algo novo.

```
class Pessoa {
  void apresentar() {
    print('Olá, eu sou uma pessoa');
  }
}

class Professor extends Pessoa {
  @override
```

```
  void apresentar() {
    super.apresentar();
    print('Eu também sou professor');
  }
}

void main() {
  Professor professor = Professor();
  professor.apresentar();
}
```

Saída:

```
Olá, eu sou uma pessoa
Eu também sou professor
```

---

7. Herança na prática

Imagine um sistema escolar.

```
class Usuario {
  String nome;
  String email;

  Usuario(this.nome, this.email);

  void login() {
    print('$nome fez login com o e-mail $email');
  }
}

class Aluno extends Usuario {
  String matricula;

  Aluno(String nome, String email, this.matricula)
      : super(nome, email);

  void acessarAula() {
    print('$nome acessou a aula');
  }
}

class Professor extends Usuario {
  String disciplina;

  Professor(String nome, String email, this.disciplina)
      : super(nome, email);

  void corrigirProva() {
    print('$nome está corrigindo provas de $disciplina');
  }
}

void main() {
  Aluno aluno = Aluno('Maria', 'maria@email.com', 'A123');
  Professor professor = Professor('João', 'joao@email.com',
'Matemática');
```

```
  aluno.login();
  aluno.acessarAula();

  professor.login();
  professor.corrigirProva();
}
```

---

8. Vantagens da herança

A herança ajuda a:

- Reaproveitar código;
- Organizar melhor o sistema;
- Evitar repetição;
- Criar classes mais especializadas;
- Facilitar manutenção.

---

9. Cuidados com herança

Nem tudo deve usar herança.

Use herança quando existir uma relação do tipo:

```
Aluno é um Usuário
Cachorro é um Animal
Carro é um Veículo
```

Evite usar herança apenas para reaproveitar código sem uma relação lógica.