**6. Классы — наследники**

В языке программирования Java классы-наследники (или дочерние классы) создаются с помощью механизма наследования. Наследование позволяет одному классу (дочернему классу) наследовать поля и методы другого класса (родительского класса). Это позволяет избежать дублирования кода, улучшить его повторное использование и создать более специализированные классы на основе существующих.

Основные понятия:

1. Родительский (базовый) класс — это класс, который предоставляет общие свойства и поведение для дочерних классов.
2. Дочерний (потомок) класс — это класс, который наследует поведение и свойства родительского класса и может добавлять или изменять их.

**Синтаксис наследования**

Для того чтобы класс стал наследником другого класса, используется ключевое слово extends. Дочерний класс наследует все публичные (public) и защищённые (protected) члены родительского класса, но не приватные (private).

Пример базового синтаксиса:

class Parent {

// Поля

String name;

// Конструктор

public Parent(String name) {

this.name = name;

}

// Метод

public void greet() {

System.out.println("Hello from " + name);

}

}

class Child extends Parent {

// Дополнительное поле

int age;

// Конструктор дочернего класса

public Child(String name, int age) {

// Вызов конструктора родительского класса

super(name);

this.age = age;

}

// Переопределение метода

@Override

public void greet() {

super.greet(); // Вызов метода родительского класса

System.out.println("I am " + age + " years old.");

}

}

public class Main {

public static void main(String[] args) {

Child child = new Child("Alice", 10);

child.greet(); // Выведет информацию о дочернем классе и родительском

}

}

Объяснение:

* В классе Parent есть поле name, конструктор и метод greet(), который выводит приветствие.
* Класс Child наследует Parent с помощью extends. Он добавляет новое поле age и переопределяет метод greet() для расширения его функциональности. В методе greet() дочернего класса сначала вызывается метод родительского класса через super.greet(), а затем добавляется информация о возрасте.
* Ключевое слово super используется для вызова конструктора или метода родительского класса.

**1. Переопределение методов (Overriding)**

Переопределение методов — это процесс, при котором дочерний класс изменяет поведение метода, унаследованного от родительского класса. Это делается для того, чтобы адаптировать поведение метода под особенности дочернего класса.

Синтаксис переопределения метода:

@Override

public returnType methodName(parameters) {

// Реализация метода в дочернем классе

}

Пример:

class Animal {

public void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Dog barks");

}

}

class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal();

myAnimal.makeSound(); // Выведет: Animal makes a sound

Dog myDog = new Dog();

myDog.makeSound(); // Выведет: Dog barks

}

}

В этом примере метод makeSound() переопределён в классе Dog, чтобы выдавать звук "bark" вместо базового звука животного.

**2. Вызов конструктора родительского класса (super)**

Когда дочерний класс создаётся, он может вызвать конструктор родительского класса с помощью ключевого слова super. Это необходимо, если родительский класс не имеет конструктора по умолчанию или если нужно выполнить дополнительную инициализацию родительского класса.

Пример:

class Vehicle {

String type;

public Vehicle(String type) {

this.type = type;

}

}

class Car extends Vehicle {

int wheels;

public Car(String type, int wheels) {

// Вызов конструктора родительского класса

super(type);

this.wheels = wheels;

}

public void displayInfo() {

System.out.println("Vehicle type: " + type + ", Wheels: " + wheels);

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car("Sedan", 4);

myCar.displayInfo(); // Выведет: Vehicle type: Sedan, Wheels: 4

}

}

В этом примере класс Car вызывает конструктор класса Vehicle с помощью super(type), чтобы инициализировать поле type.

**3. Множественное наследование (через интерфейсы)**

В Java множественное наследование классов не поддерживается, то есть класс не может наследовать несколько других классов. Однако множественное наследование возможно через интерфейсы, которые позволяют классу реализовывать несколько интерфейсов.

Пример:

interface Engine {

void startEngine();

}

interface Wheels {

void rotateWheels();

}

class Car implements Engine, Wheels {

@Override

public void startEngine() {

System.out.println("Engine is started.");

}

@Override

public void rotateWheels() {

System.out.println("Wheels are rotating.");

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car();

myCar.startEngine();

myCar.rotateWheels();

}

}

Здесь класс Car реализует два интерфейса — Engine и Wheels, что позволяет ему использовать методы из обоих интерфейсов.

**4. Полиморфизм с наследованием**

Полиморфизм позволяет использовать объекты различных классов, которые могут реализовывать одинаковые методы. Благодаря наследованию, можно работать с объектами типа родительского класса, но в реальности это будут объекты дочерних классов.

Пример:

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Animal myAnimal = new Dog(); // Ссылаемся на объект типа Animal, но создаём объект типа Dog

myAnimal.makeSound(); // Выведет: Dog barks

}

}

В этом примере переменная типа Animal ссылается на объект типа Dog, и при вызове метода makeSound() выполняется метод, переопределённый в дочернем классе Dog.

Заключение

Наследование — это один из важнейших принципов объектно-ориентированного программирования в Java. Оно позволяет создавать иерархию классов, упрощает повторное использование кода, делает код более организованным и облегчает его поддержку. Наследование в Java поддерживает ключевые принципы, такие как переопределение методов и полиморфизм, что позволяет создавать гибкие и масштабируемые программы.