Наследование

Наследование является важной опорой ООП (объектно-ориентированного программирования).

Это механизм в Java, с помощью которого одному классу разрешено наследовать свойства (поля и методы) другого класса.

важные термины:

Суперкласс: класс, функции которого наследуются, известен как суперкласс (также известный как базовый или родительский класс).

Подкласс: класс, который наследует другой класс, известен как подкласс (также известный как производный, расширенный или дочерний класс). Подкласс может добавлять свои собственные поля и методы в дополнение к полям и методам суперкласса.

Повторное использование: Наследование поддерживает концепцию «повторного использования», т. е. когда мы хотим создать новый класс, а уже есть класс, который включает часть кода, который нам нужен, мы можем получить наш новый класс из существующего класса. Делая это, мы повторно используем поля и методы существующего класса.

Правила наследования

Правило 1. Наследуем только один класс.

Java не поддерживает наследование нескольких классов. Один класс - один родитель.

Обратите внимание - нельзя наследовать самого себя!

Правило 2. Наследуется все кроме приватных переменных и методов.

Выше мы говорили, что класс-наследник будет иметь доступ ко всем переменным и методам родителя. Это не совсем так.

На самом деле, все методы и переменные, помеченные модификатором private, не доступны классу-наследнику.

Правило 3. Переделать метод класса-родителя.

Представим, что мы наследуем класс, но нам нравится не все, что мы унаследовали. Допустим мы хотим, чтобы определенный--- метод работал не так, как в родителе.

Для того, чтобы переопределить метод класса-родителя, пишем над ним @Override:

Правило 4. Вызываем методы родителя через ключевое слово super.

Представим, что Вы хотите изменить метод родительского класса совсем чуть-чуть - буквально дописать пару строк. Тогда в своем методе мы можем вызвать родительский метод с помощью ключевого слова super.

Например, у нас есть класс Dog с методом voice():

// Example1.java

public class Dog {

public void voice(){

System.out.println("Dog voice!");

}

}

Кроме того, у нас есть класс Cat, который наследует класс Dog и имеет свой метод voice(). В нем мы вызываем родительский метод через ключевое слово super:

// Example1.java

class Cat extends Dog {

@Override

public void voice()

{

super.voice();

System.out.println("Cat voice!");

}

}

Теперь, вызовем метод voice() из класса Cat:

// Example1.java

public class Test {

public static void main(String[] args) {

Cat cat = new Cat();

cat.voice();

}

}

-> сделать выводы

Правило 5. Запрещаем наследование.

Если Вы не хотите, чтобы кто-то наследовал Ваш класс, поставьте перед ним модификатор final. Например:

-> добавить final в класс Dog

---

При создании объекта класса Cat создается копия всех полей суперкласса и выделяется память в этом объекте.

Поэтому, используя объект подкласса, мы также можем получить доступ к членам суперкласса.

вопрос??? Как вы думаете сколько обьектов будет создано?

ответ: при наследовании создается только объект подкласса, а не суперкласса.

---

еще пример (если нет Ломбок, используйте обычные Гет и Сет)

доказываем, что создается только один класс

!!! -> изучить код

// Example2.java

@Data

public class Fruit {

public String[] vitamins;

public int sugar;

public Fruit() {

// Print statement

System.out.println("Super class constructor");

// Displaying object hashcode of super class

System.out.println("Super class object hashcode :" + this.hashCode());

System.out.println(this.getClass().getName());

}

@Override

public String toString() {

return " vitamins = " + Arrays.toString(vitamins) +

", sugar = " + sugar;

}

}

// Example2.java

@Data

public class Ananas extends Fruit {

private int cellulose;

public Ananas() {

// Print statement

System.out.println("Subclass constructor invoked");

// Displaying object hashcode of subclass

System.out.println("Sub class object hashcode :" + this.hashCode());

System.out.println(this.hashCode() + " " + super.hashCode());

System.out.println(this.getClass().getName() + " " + super.getClass().getName());

}

@Override

public String toString() {

return "Ananas{cellulose = " + cellulose + super.toString() +"}";

}

}

// Example2.java

public class InheritancePractice {

public static void main(String[] args) {

Ananas ananas = new Ananas();

ananas.setCellulose(20);

ananas.setSugar(10);

ananas.setVitamins(new String[]{"A", "B"});

System.out.println(ananas);

}

}

Объяснение:

хэш-код объекта суперкласса (Fruit) и хэш-код объекта подкласса (Ananas) одинаковы, поэтому создается только один объект.

Этот объект относится к классу Ananas (подклассу), так как когда мы пытаемся напечатать имя класса, в котором создан объект, он печатает Ananas, который является подклассом.

---

когда вызывается конструктор, создание объекта не обязательно!

!!! - не надо спешить, если время лекции на исходе, можно эту тему продолжить на следующей.

Задайте вопросы студентам по пройденному материалу, выясните точно ли всем понятно!

ТИПЫ НАСЛЕДОВАНИЯ

1. Одиночное наследование.

При одиночном наследовании подклассы наследуют свойства одного суперкласса.

// Example3.java

class one {

public void printOne()

{

System.out.println("one");

}

}

// Example3.java

class two extends one {

public void printFor() { System.out.println("for"); }

}

2. Многоуровневое наследование:

в многоуровневом наследовании производный класс будет наследовать базовый класс, и как, и производный класс, он также будет выступать в качестве базового класса для другого класса.

// Example4.java

class one {

public void printOne()

{

System.out.println("One");

}

}

// Example4.java

class two extends one {

public void printFor() { System.out.println("for"); }

}

// Example4.java

class three extends two {

public void printOne()

{

System.out.println("One");

}

}

3. Иерархическое наследование.

При иерархическом наследовании один класс служит суперклассом (базовым классом) для более чем одного подкласса.

class A {

public void printA() { System.out.println("Class A"); }

}

class B extends A {

public void printB() { System.out.println("Class B"); }

}

class C extends A {

public void printC() { System.out.println("Class C"); }

}

class D extends A {

public void printD() { System.out.println("Class D"); }

}

public class Test {

public static void main(String[] args)

{

B b = new B();

b.printA();

b.printB();

C c = new C();

c.printA();

c.printC();

D d = new D();

d.printA();

d.printD();

}

}

4. Множественное наследование (через интерфейсы).

При множественном наследовании один класс может иметь более одного суперкласса и наследовать функции от всех родительских классов. Обратите внимание, что Java не поддерживает множественное наследование с классами.

В java мы можем добиться множественного наследования только через интерфейсы.

!!! -> пример не тестировался

// Example6.java

public interface One {

public void printOne();

}

// Example6.java

public interface Two {

public void printTwo();

}

// Example6.java

public interface Three extends one, two {

public void printTree();

}

// Example6.java

class Child implements Three {

@Override

public void printOne()

{

System.out.println("One");

}

public void printTwo() {

System.out.println("Two");

}

}

// Example6.java

public class Main {

public static void main(String[] args)

{

Child child = new child();

child.printOne();

child.printTwo();

child.printTree();

}

}

Важные факты о наследовании в Java

Суперкласс по умолчанию : кроме класса Object, у которого нет суперкласса, каждый класс имеет один и только один прямой суперкласс (одиночное наследование). В отсутствие какого-либо другого явного суперкласса каждый класс неявно является подклассом класса Object.

Суперкласс может быть только один: Суперкласс может иметь любое количество подклассов. Но подкласс может иметь только один суперкласс. Это связано с тем, что Java не поддерживает множественное наследование с классами. Хотя с интерфейсами множественное наследование поддерживается java.

Наследование конструкторов: подкласс наследует все члены (поля, методы и вложенные классы) от своего суперкласса. Конструкторы не являются членами, поэтому они не наследуются подклассами, но конструктор суперкласса может быть вызван из подкласса.

Наследование закрытых членов: подкласс не наследует закрытые члены своего родительского класса. Однако, если у суперкласса есть общедоступные или защищенные методы (например, геттеры и сеттеры) для доступа к его закрытым полям, они также могут использоваться подклассом.

---

Слишком длинно, в 4 часа не уложиться. - это будет отрезано

Перегрузка overloading в Java

Перегрузка позволяет различным методам иметь одно и то же имя, но разные сигнатуры, где сигнатура может отличаться количеством входных параметров или типом входных параметров, или и тем, и другим.

Перегрузка связана с полиморфизмом времени компиляции (так же может называеться статическим).

// Example7.java

public class Sum {

public int sum(int x, int y)

{

return (x + y);

}

public int sum(int x, int y, int z)

{

return (x + y + z);

}

public double sum(double x, double y)

{

return (x + y);

}

public static void main(String args[])

{

Sum s = new Sum();

System.out.println(s.sum(1, 2));

System.out.println(s.sum(1, 2, 3));

System.out.println(s.sum(1.3, 2.0));

}

}

---

Переопределение overriding в Java

В любом объектно-ориентированном языке программирования переопределение — это функция, которая позволяет подклассу или дочернему классу предоставлять конкретную реализацию метода, который уже предоставлен одним из его суперклассов или родительских классов.

Переопределение метода — это один из способов достижения полиморфизма во время выполнения в Java.

// Example8.java

class Parent {

void show()

{

System.out.println("Parent's show()");

}

}

// Example8.java

class Child extends Parent {

@Override

void show()

{

System.out.println("Child's show()");

}

}

// Example8.java

class Main {

public static void main(String[] args)

{

Parent parent = new Parent();

parent.show();

Parent child = new Child();

child.show();

}

}

---

ABSTRACT CLASS

Наблюдение 1: В Java, экземпляр абстрактного класса не может быть создан, хотя у нас могут быть ссылки на тип абстрактного класса.

// Example9.java

public class Main {

public static void main(String args[])

{

AbstractClass caller = new Caller();

caller.print();

}

}

// Example9.java

public abstract class AbstractClass {

public abstract void print();

}

// Example9.java

public class Caller extends AbstractClass {

void print()

{

System.out.println("Caller print() called");

}

}

Наблюдение 2: абстрактный класс в Java может содержать конструкторы.

А конструктор абстрактного класса вызывается при создании экземпляра унаследованного класса.

// Example10.java

public class Main {

public static void main(String args[]) {

Caller caller = new Caller();

caller.print();

}

}

// Example10.java

public abstract class AbstractClass {

AbstractClass() {

System.out.println("AbstractClass Constructor Called");

}

abstract void print();

}

// Example10.java

public class Caller extends AbstractClass {

Caller() {

System.out.println("Caller Constructor Called");

}

public void print() {

System.out.println("Caller print() called");

}

}

Наблюдение 3: в Java у нас может быть абстрактный класс без какого-либо абстрактного метода.

Это позволяет нам создавать классы, экземпляры которых нельзя создать, но которые можно только наследовать

// Example11.java

public class Main {

public static void main(String args[]) {

Caller caller = new Caller();

caller.print();

}

}

// Example11.java

public abstract class AbstractClass {

void print() {

System.out.println("Function of AbstractClass class is called");

}

}

// Example11.java

public class Caller extends AbstractClass {

// This class only inherits the Base class methods and properties

}

Наблюдение 4: Абстрактные классы также могут иметь final методы (методы, которые нельзя переопределить)

Наблюдение 5: Подобно интерфейсу, мы можем определить статические методы в абстрактном классе, которые можно вызывать без объекта.

// Example12.java

public abstract class Helper {

public static void print() {

System.out.println("Print from class Helper");

}

}

// Example12.java

public class HelperTest extends Helper {

public static void main(String[] args) {

Helper.print();

}

}

Наблюдение 6: мы можем использовать ключевое слово abstract для объявления классов верхнего уровня (внешний класс), а также внутренних классов как абстрактных.

// Example13.java

public abstract class ClassOne {

public abstract class ClassTwo {

abstract void printAbstract();

}

}

// Example13.java

public class ClassThird extends ClassOne {

class ClassFour extends ClassTwo {

public void printAbstract() { System.out.println("Inside abstract method implementation"); }

}

}

// Example13.java

public class Main {

public static void main(String args[]) {

ClassThird outer = new ClassThird();

ClassThird.ClassFour inner = outer.new ClassFour();

inner.myAbstractMethod();

}

}

Наблюдение 7: если класс содержит хотя бы один абстрактный метод, то обязательно мы должны объявить класс абстрактным, иначе мы получим ошибку времени компиляции.

// Example14.java

public abstract class Demo {

abstract void ParentAbstract();

}

// Example14.java

class Child extends ParentAbstract {

public void print() {

System.out.print("Hello child");

}

}

// Example14.java

class Test {

public static void main(String[] args) {

Child child = new Child();

child.print();

}

}

Наблюдение 8: Если дочерний класс не может обеспечить реализацию всех абстрактных методов родительского класса, тогда мы должны объявить этот дочерний класс абстрактным, чтобы дочерний класс следующего уровня обеспечивал реализацию оставшегося абстрактного метода.

// Example15.java

abstract class ParentAbstract {

abstract void methodOne();

abstract void methodTwo();

abstract void methodThird();

}

// Example15.java

abstract class FirstChild extends ParentAbstract {

public void methodOne() {

System.out.println("Inside methodOne");

}

}

// Example15.java

class SecondChild extends FirstChild {

public void methodTwo() {

System.out.println("Inside methodTwo");

}

public void methodThird() {

System.out.println("Inside methodThird");

}

}

// Example15.java

class Test {

public static void main(String[] args) {

SecondChild secondChild = new SecondChild();

secondChild.methodOne();

secondChild.methodTwo();

secondChild.methodThird();

}

}

---

ПОЛИМОРФИЗМ

полиморфизм означает наличие множества форм.

Реальная иллюстрация: полиморфизм

Человек одновременно может иметь разные характеристики. Мужчина одновременно является отцом, мужем, работником.

тип 1: полиморфизм времени компиляции, также известен как статический полиморфизм.

Достигается перегрузкой функций или перегрузкой операторов, Java поддерживает только первый вариант.

// Example16.java

class Calculator {

public static int multiply(int a, int b) {

return a \* b;

}

public static double multiply(double a, double b) {

return a \* b;

}

}

// Example16.java

class Test {

public static void main(String[] args) {

System.out.println(Calculator.multiply(2, 2));

System.out.println(Calculator.multiply(2.2, 2.2));

}

}

Тип 2: Полиморфизм времени выполнения

Полиморфизма достигается переопределением метода.

// Example17.java

class Parent {

void print() {

System.out.println("parent class");

}

}

// Example17.java

class ChildOne extends Parent {

void print() { System.out.println("childOne class"); }

}

// Example17.java

class childTwo extends Parent {

void print() {

System.out.println("childTwo class");

}

}

// Example17.java

class Test {

public static void main(String[] args) {

Parent parent;

parent = new ChildOne();

parent.print();

parent = new ChildTwo();

parent.print();

}

}

Объяснение вывода:

В этом примере, когда создается объект наследника, вызывается метод внутри наследника. Это связано с тем, что метод родителя переопределяется наследником. Поскольку метод переопределен, этот метод имеет более высокий приоритет.

Переопределение в Java

Переопределение метода — это один из способов достижения полиморфизма во время выполнения в Java.

Версия выполняемого метода будет определяться объектом, который используется для его вызова.

Если для вызова метода используется объект родительского класса, то будет выполняться версия в родительском классе, но если для вызова метода используется объект подкласса, то будет выполняться версия в дочернем классе.

// Example18.java

class Parent {

void show() {

System.out.println("Parent's show()");

}

}

// Example18.java

class Child extends Parent {

@Override

void show() {

System.out.println("Child's show()");

}

}

// Example18.java

class Main {

public static void main(String[] args) {

Parent parent = new Parent();

parent.show();

Parent child = new Child();

child.show();

}

}

Правила переопределения метода:

---

1 Переопределение и модификаторы доступа: модификатор доступа для переопределяющего метода может разрешить больший, но не меньший доступ, чем переопределенный метод.

// Example19.java

class Parent {

private void methodOne() {

System.out.println("From parent methodOnem1()");

}

protected void methodTwo() {

System.out.println("From parent methodTwo()");

}

}

// Example19.java

class Child extends Parent {

private void methodOne() {

System.out.println("From child methodOne()");

}

@Override

public void methodTwo() {

System.out.println("From child methodTwo()");

}

}

// Example19.java

class Main {

public static void main(String[] args) {

Parent parent = new Parent();

parent.methodTwo();

Parent child = new Child();

child.methodTwo();

}

}

---

2 final методы не могут быть переопределены: если мы не хотим, чтобы метод был переопределен, мы объявляем его как final.

// Example20.java

class Parent {

final void show() {}

}

// Example20.java

class Child extends Parent {

void show() {}

}

---

3 Статические методы не могут быть переопределены.

Когда вы определяете статический метод с той же сигнатурой, что и статический метод в базовом классе, это называется сокрытием метода.

// Example21.java

class Parent {

public static void mthodOne() {

System.out.println("From parent static mthodOne()");

}

public void methodTwo() {

System.out.println("From parent non-static methodTwo()");

}

}

// Example21.java

class Child extends Parent {

static void mthodOne() {

System.out.println("From child static mthodOne()");

}

@Override

public void methodTwo() {

System.out.println("From child non-static methodTwo()");

}

}

// Example21.java

class Main {

public static void main(String[] args) {

Parent child = new Child();

child.mthodOne();

child.methodTwo();

}

}

---

4 private методы не могут быть переопределены, поскольку они связаны во время компиляции.

---

5 Переопределяющий метод должен иметь тот же тип возвращаемого значения (или подтип):

начиная с Java 5.0 можно иметь другой тип возвращаемого значения для переопределяющего метода в дочернем классе, но дочерний тип возвращаемого значения должен быть подтипом родительского возвращаемого типа.

---

6 Вызов переопределенного метода из подкласса: мы можем вызвать метод родительского класса в переопределяющем методе, используя ключевое слово super.

// Example22.java

class Parent {

void show() {

System.out.println("Parent's show()");

}

}

// Example22.java

class Child extends Parent {

@Override

void show() {

super.show();

System.out.println("Child's show()");

}

}

// Example22.java

class Main {

public static void main(String[] args) {

Parent child = new Child();

cahld.show();

}

}

---

7 Переопределение и конструктор: мы не можем переопределить конструктор, поскольку родительский и дочерний классы никогда не могут иметь конструктор с одинаковым именем (имя конструктора всегда должно совпадать с именем класса).