类

- 把某一类有相同特征(属性)和行为(方法)抽象出来的概念,叫做类.类是不占用内存的

对象

- 通过 `new` 关键字来创建一个新的对象.对象占用内存

变量

- 成员变量：在类中方法之外定义的变量,叫做成员变量
- 局部变量：在方法体,代码块中定义的变量,叫做局部变量
- 成员变量和局部变量的区别:
    1. ***定义位置***:成员变量在类里面,方法体外面;局部变量定义在方法体,代码块里面
    2. ***默认值***:成员变量如果没有赋值,则会默认值,和数组一样;局部变量必须有默认;
    3. ***生命周期***:成员变量随着对象的创建而创建,随着对象被垃圾回收而消失;随着方法进栈而诞生,随着方法出栈而消失
    4. ***适用范围***:成员变量整个类中都可使用;局部变量只有在方法体和代码块中执行
    5. ***存储位置***:成员变量储存在堆内存中;局部变量储存在栈内存中;

面向对象的三大特征

1. 封装

访问修饰符

- private: 本类
- default: 本类, 同包
- protected: 本类, 同包, 子类
- public: 全部

getter/setter 方法

略，不要改写set/get方法，让其有些附加功能。

构造方法

重载

1. 同一类
2. 方法名一致
3. 方法的参数列表不一样(类型,顺序,数量)
4. 方法只有返回值不一样无法重载
5. 重载多用于构造方法上

初始化顺序

```
静态属性初始化 ->静态方法初始化 -> 普通属性初始化 -> 普通方法快初始化 -> 构造函数初始化
```

Demo01: 方法的重载

```
  public void show(int num) {}
  public void show(float f) {}
   ...
  public void show(int num, String str) {}
  public void show(String str, int num) {}
```

Demo02: 初始化顺序

```
public class Demo02 {
    // 静态属性,调用了静态方法:getStaticField()
    private static final String staticField = getStaticField();

    // 静态块
    static {
        System.out.println(staticField);
        System.out.println("静态方法块初始化");
    }

    // 普通属性
    private final String field = getField();

    // 普通方法快
    {
        System.out.println(field);
    }

    // 构造函数
    public Test03() {
        System.out.println("构造函数初始化");
    }

    // 静态方法
    public static String getStaticField() {
        return "Static Field Initial";
    }

    //静态方法
    public static String getField() {
        return "Field Initial";
    }
```

```
    // 主函数
    public static void main(String[] argc) {
        new Demo02();
    }
}
```

输出如下: `Static Field Initial` 静态方法块初始化 `Field Initial` 构造函数初始化

2. 继承

子类/父类

- 子类只能继承父类中的非私有、非final、非static的属性和方法
- 构造方法不能被继承
- 一个子类只能有一个父类，一个父类可以有多个子类
- 继承关系满足： "is a"
- 若一个 B 类继承 A 类
    1. B 是 A 的子类（派生类）
    2. A 是 B 的超类（父类，基类）

重写

- 子类**重写**父类方法时：
    1. 方法的访问修饰符对比父类的方法的访问修饰符范围只能放大
    2. 方法的返回值类型与父类的中方法的返回值完全一致
    3. 方法的参数列表与父类中方法的参数列表完全一致
    4. 方法的方法名与父类方法的方法名完全一致

this 与 super

- this ：表示当前类对象
    - 可以调用本类中的属性，方法，构造方法。当调用构造方法时，只能位于构造函数的第一行。
- super ：出现在有继承关系的子类中
    - 可以调用父类中的属性，方法，构造方法。当调用构造方法时，只能位于构造函数的第一行。
    - 如果子类中构造方法中没有super()父类的构造方法 ,那么默认第一行会添加 super()语句调用父类中的构造方法;
    - 也可以调用有参的构造函数
    - 当父类没有无参构造方法时,子类有无参构造,如果没有 super 父类的有参构造，将会默认 super()父类的无参构造,但由于父类没有无参构造,所以程序将会程序无法编译，所以这时必须要手动 super()父类的有参构造

Demo01: 构造方法

```
public class Student {
    String name;
    String gender;
```

```
    public Student() {
        // this 调用构造方法只能在构造方法第一行
        this("乔治", "男");
    }
    public Student(String name, String gender) {
        this.name = name;
        this.gender = gender;
    }
    public void speak() {
        System.out.println("speak!");
    }
    public void showInfo(String name) {
        System.out.println(this.name);
        this.speak();
    }
    public static void main(String[] args) {
        Student student = new Student();
        student.showInfo("");
    }
}
```

输出： 乔治 speak!

Demo02: super

```
public class Person {
    String name;
    String gender;
    public Person() {
        System.out.println("无参构造");
    }
    public Person(String name, String gender) {
        this.name = name;
        this.gender = gender;
    }
    public void eat() {
        System.out.println("吃---");
    }
}
public class Teacher extends Person {
    public Teacher() {
        super("", "");
    }
    public Teacher(String name, String gender) {
        super(name, gender);
    }
    public void showInfo() {
        System.out.println(super.name);
        System.out.println(super.gender);
        super.eat();
    }
```

```
    public static void main(String[] args) {
        Teacher teacher = new Teacher();
        teacher.showInfo();
    }
}
```

Demo03: this

> 一个简单的链式调用,体现了this是指向了当前对象.

```
public class Apple {
    int i = 0;

    Apple eatApple() {
        i++;
        return this;
    }

    public static void main(String[] args) {
        Apple apple = new Apple();
        apple.eatApple().eatApple().eatApple()
                .eatApple().eatApple().eatApple()
                .eatApple().eatApple();
        System.out.println("吃了 " + apple.i + " 次苹果");
    }
}
```

输出: 吃了 8 次苹果

3. 多态

- 提高代码的复用性,扩展性
- 解耦
- 实现多态的特点:
    1. 有[[接口与抽象类|继承和实现]]关系
    2. 子类必须重写了父类/父接口中的方法
    3. 父类的引用指向了子类的对象 如:List<String> list = new ArrayList<>()>
    4. 多态是向上转型的.子类的访问修饰符只能相比父类中的扩大.如果想调用子类中的方法,必须向下转型.

Demo: 多态

```
// 父类
public class Vehicle {
    String color;
    String wheel;
    public Vehicle() {
```

```
    }
    public void addOil() {
        System.out.println("加油");
    }
}
// 子类bus
public class Bus extends Vehicle {
    public Bus() {
        super.wheel = "Bus有八个轮子";
    }
}
// 子类car
public class Car extends Vehicle {
    public Car() {
        super.wheel = "Car有四个轮子";
    }
    // 重写了父类的addOil()方法
    @Override
    public void addOil() {
        System.out.println("Car 加油了");
    }
}
// 工厂类
// 其实也就是根据参数 返回一个Car或者Bus对象
// return 一个 new Bus(); 或者一个 new Car();
public class VehicleFactory {
    static Vehicle getVehicle(String type) {
        if (type.equals("car")) {
            return new Car();
        } else if (type.equals("bus")) {
            return new Bus();
        } else {
            return null;
        }
    }
}
// 测试类
public class Test {
    // 这个方法主要就是打印出对象中轮子的信息,是一个字符串.
    void run(Vehicle vehicle) {
        System.out.println(vehicle.wheel);
    }
    // main方法
    public static void main(String[] args) {
    // 我们调用了一个静态方法getVehicle(),传了一个"car"字符串
    // 讲道理,return new Car()
    // 其实下面这句话 和 new Car();是完全一样的效果
    Vehicle vehicle = VehicleFactory.getVehicle("car");

    // 实例化测试类
    Test test = new Test();
    // 调用的test对象中的run()方法
    test.run(vehicle);
    // 调用Vehicle vehicle的addOil()方法
```

```
    vehicle.addOil();
    }
}
```

输出: Car有四个轮子 Car 加油了

> 对于多态呢,你看,我在别的类需要一个车的参数,但是我不知道是什么车,我就可以把他们的父类作为参数.如果说只是Car car = new Car(); 一个同样功能的方法我就要再重载一个Bus类的参数,但是如果我们有了多态,我们就可以只写一个方法,就可以实现接收不同的参数,返回值也是同理.私以为,多态,这个东西对统一参数和返回值上有着重大的作用.

重写与重载注意事项:

> [[== 与 equals|重写 equals() 方法一定要重写 hashCode()方法]]

- 方法的重写:方法的重写是子类对父类中非私有,非 final,非 static 的方法的重新实现. 返回值,方法名,参数列表必须完全一致, 且子类方法访问修饰符只能对比父类的只能扩大访问范围,不能缩小.
- 方法的重载:在同一类中,方法名相同,参数列表不同,返回值可以不同.同一类中,故访问范围可以扩大或缩小
- 区别:1. 位置不同 2. 参数列表不同 3. 返回值不同 4. 访问范围不同

关键字

static

> 可修饰：方法，成员变量

1. 类在加载是被加载
2. 只会加载一次,并且内存中只有一块空间保存
3. static可以修饰方法和成员变量
    1. 1. static 修饰的方法叫静态方法(类方法),该方法属于这个类,通过 **类名.静态方法名(实参);**
    2. 没有被 static 修饰的叫做成员方法,成员方法的调用方式为: **对象.方法名(实参)**
    3. 被staic修饰的变量叫静态变量
4. 静态块:
    1. 静态代码块主要用于类的初始化，它只执行一次。
    2. 静态代码块里的变量都是局部变量，只在块内有效。
    3. 静态代码块只能访问类的静态成员，而不允许访问实例成员。
    4. 一个类中可以有多个静态块，按顺序执行。

Demo01: 静态变量/静态块

```
class Demo01 {
 // 成员变量(类比C的"全局变量")
    static int a = 0;
    static {
      // 局部变量
        a = 0;
      // 定义int整型变量b,只能在静态块中使用,无法被静态块外部调用
      int b = 1;
```

```
    }
}
```

Demo02: 静态块

```
class Demo02 {
    static {
        System.out.println("1234");
    }
    public static void main(String[] args){
    }
}
```

输出: "1234"

Demo03: 静态变量只有一块存储空间

```
public class Demo03 {
    static String name;
        int age;
    public Demo03(String name, int age) {
        Demo03.name = name;
        this.age = age;
    }
    public static void main(String[] args) {
        Demo03 d3 = new Demo03("晓峰", 20);
        Demo03 d3 = new Demo03("圆圆", 16);
        System.out.println(name);
        System.out.println(d3.age);
        System.out.println(name);
        System.out.println(d3.age);
    }
}
```

输出： 圆圆 2 圆圆 16

final

> 可修饰：类，变量（成员变量/局部变量），方法

- 被final修饰的类不能被继承（如String类）
- 被final修饰的成员变量必须给初始值，一旦赋值不能更改.
- 被final修饰的局部变量不必有初始值，一旦赋值不能更改.
- 被final修饰的方法不能被重写.