Java基础内容——面向对象

- 基本概念
- 继承
- 重写和重载
- 多态
- 抽象类
- 接口
- 包
- 作用域

面向对象概念

- 常见的两种方法：面向对象和面向过程
- 解决问题的两种思路，把问题组织起来的方式不同
- 面向对象把特定数据组合和相关的方法整合到一起
- 没有哪个方法更好，根据具体需求去选择

class和instance区分（类和实体）

- 类：就是类型
- 实体：类中具体的一个
- 举例：
  - 书是类，《安卓开发》就是实体
  - 学习用品是类，书就是实体
  - 纸制品是类，书也是实体

> 哲学的内容就此结束

定义class

```
class Person {
    public String name;
    public int age;
}
```

- class name: Person
- field: name/age
- public是修饰字段

class例子练习

书： name、author、price

创建实例

```
Person ming = new Person();
ming.name = "ming";
ming.age = 12;
```

必须使用new

构造方法

```
class Person {
    public String name;
    public int age;

    Person(){}
    Person(String name, int age){
        this.name = name;
        this.age = age;
    }
}
```

构造方法使用

```
Person ming = new Person();
ming.name = "ming";
ming.age = 12;

Person hua = new Person("hua",11);
```

默认构造方法

- 如果一个类没有定义构造方法，编译器会自动为我们生成一个默认构造方法，它没有参数，也没有执行语句。
- 如果定义了自己的构造方法呢?

方法/成员函数

```
class Person {
    public String name;
    public int age;
    public void setName(String name)
    {
        this.name = name;
    }
    public String getName()
    {
        return  this.name;
    }
}
```

private使用和意义

- ？ field用public暴露给外部可能会破坏封装性
- ？ 直接操作field，容易造成逻辑混乱

private个人理解

- 减少直接操作的错误
- 便于功能升级

静态变量

- class普通字段叫实例字段，每个实例中占有不同的空间。
- class中用static修饰的字段叫静态字段，所有实例同享同一段空间。
- 建议只用className.fieldName访问静态字段。

静态方法

- 和上面一样，用static修饰的方法叫静态方法。
- 通过类名就可调用，也建议这么使用。
- 静态方法内部，无法访问this变量，也无法访问实例字段，它只能访问静态字段。
- 常用于工具类方法如Arrays.sort()、Math.acos()

方法重载(overload)

- 方法名相同，但各自的参数不同，称为方法重载（Overload）
- 方法重载的目的是，功能类似的方法使用同一名字，更容易记住和调用
- 注意和重写（Override）区分

重载举例

String类提供了多个重载方法indexOf()查找子串：

- int indexOf(int ch)：根据字符的Unicode码查找；
- int indexOf(String str)：根据字符串查找；
- int indexOf(int ch, int fromIndex)：根据字符查找，但指定起始位置；
- int indexOf(String str, int fromIndex)根据字符串查找，但指定起始位置。

重载规则

- 被重载的方法必须在同一个类中;
- 被重载的方法必须改变参数列表(参数个数或类型不一样);
- 被重载的方法可以改变返回类型;
- 被重载的方法可以改变访问修饰符;
- 被重载的方法可以声明新的或更广的检查异常;
- 无法以返回值类型作为重载函数的区分标准.

继承

继承是java面向对象编程技术的一块基石，因为它允许从现有类轻松创建新类。

> 注意：Java只允许一个class继承自一个类，因此，一个类有且仅有一个父类。

直接定义

```
class Student {
    private String name;
    private int age;
    private int score;  // new field

    public String getName() {...}
    public void setName(String name) {...}
    public int getAge() {...}
    public void setAge(int age) {...}
    public int getScore() { … }  // new field
    public void setScore(int score) { … }  // new field
    // other new functions
}
```

使用继承定义

```
class Student extends Person {
    // 不要重复name和age字段/方法,
    // 只需要定义新增score字段/方法:

    private int score;
    public int getScore() { … }
    public void setScore(int score) { … }
}
```

> 注意：子类自动获得了父类的所有字段，严禁定义与父类重名的字段！

术语

- Person称为超类（super class），父类（parent class），基类（base class）--Student称为子类（subclass），扩展类（extended class）

继承树

Student --》 Person --》 Object

> 除了Object，每个类都父类

protected

继承中，子类无法访问父类的private字段或者private方法。
为了让子类可以访问父类的字段，我们需要把private改为protected。

super

- super关键字表示父类（超类）.
- 常常在构建方法中需要用到

方法重写(Override)

当子类的某些方法的实现过程和父类不同时，有可能用到重写。
解决方法：

- 重新写个新方法 干脆，没有多级牵连
- 重写父类的同名方法 便于统一识别，多态

向上转型

```
Student s = new Student();
Person p = new Person();
Object obj = s;
```

这种把一个子类类型安全地变为父类类型的赋值，被称为向上转型（upcasting）。

向下转型

```
Person p1 = new Student(); // upcasting, ok
Person p2 = new Person();
Student s1 = (Student) p1; // ok
Student s2 = (Student) p2; // runtime error! ClassCastException!
```

因为p2的实际类型是Person，不能把父类变为子类，因为子类功能比父类多，多的功能无法凭空变出来。

预先判断 instanceof

```
Person p = new Student();
if (p instanceof Student) {
    // 只有判断成功才会向下转型:
    Student s = (Student) p; // 一定会成功
}
```

多态

- 多态:针对某个类型的方法调用，其真正执行的方法取决于运行时期实际类型的方法
- 作用：同一类指针，实现不同功能

重写(Overriding)和重载(Overloading)总结

方法的重写(Overriding)和重载(Overloading)是java多态性的不同表现。
(1)方法重载是一个类中定义了多个方法名相同,而他们的参数的数量不同或数量相同而类型和次序不同,则称为方法的重载(Overloading)。
(2)方法重写是在子类存在方法与父类的方法的名字相同,而且参数的个数与类型一样,返回值也一样的方法,就称为重写(Overriding)。
(3)方法重载是一个类的多态性表现, 而方法重写是子类与父类的一种多态性表现。

抽象类

使用场景：当父类中的某个方法没有实际实现，而子类需要，这正是抽象方法出场的时候。

- 添加abstract作为修饰
- 不需要实现该方法
  此时该类也应该加上abstract，变成抽象类，它再也不能被实例化。

> 这个类生来为了当“爸爸”，只能当父类，被其他类继承。

接口

接口是一组方法的实现规范，便于多方合作开发。

```
interface DataBaseManagement {
    public String[] getAllDatabase();
    public boolean addDatabase(String name);
}
```

- 明确地定义了方法的名称，参数，和返回值
- 没有任何实现过程的规范
- 使用implements关键字实现接口

包 package

类似C++中的namespace：

- 可避免名称冲突的问题
- 便于将class和功能分类管理、查找和使用
- 包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类

包规范

- 完整类名应该是 包名.类名，达到了解决名称冲突的效果
- 包没有父子关系。java.util和java.util.zip是不同的包，两者没有任何继承关系
- 所有Java文件对应的目录层次要和包的层次一致
- 包作用域：位于同一个包的类，可以访问包作用域的字段和方法。不用public、protected、private修饰的字段和方法就是包作用域。
- 一个.java文件只能包含一个public类，但可以包含多个非public类。如果有public类，文件名必须和public类的名字相同