面向对象

何为类和对象?

你可以想象成是人类和你：你就是人类的一个特例，因此你是人类的一个实例，但人类并不是你，也不是我，同样也不是他，而是一个抽象的总和——当我们没有具体到哪一个实例（对象）的时候，光谈类是没有什么意义的。

怎样实现呢?

类的结构：**成员变量**和**成员方法**

```
public class People {
    //成员变量
    int age;
    String name;
    //成员方法
    public void say(){
        System.out.println("Hello Objects!");    //如果没有对象，这句话能否输出呢?
    }
}
```

新手常犯的错误：

```
public class People {
        System.out.println("Hello Objects!");    //这是个啥？请问该放到哪里呢?
}
```

在类里面始终记住：只有两个东西：**变量和方法**，**变量和方法**，**变量和方法**

面向对象的三大性质

封装

为啥要封装，首先理解不封装的坏处！

再提这几个修饰符：

修饰符	描述
public	公有的，任何类都可以调用它
protected	被保护的，它和它的子类可以调用它
package-private(default)	包私有，处于同一个包的可以调用它
private	私有，只有这个类才可以调用它

> 封装是一种可以使类中的字段私有并能通过公有方法来访问私有字段的技术。如果一个字段被声明为私有，它就不能在类的外部被访问，从而隐藏了类内部的字段。基于这个原因，封装有时也被称为数据隐藏。
>
> 封装可以被认为是一种能够保护代码和数据被定义在类外的其它代码任意访问的屏障。访问数据和代码由一个接口严格控制。 封装的主要好处是修改我们实现的代码而又不会破坏其他人使用我们的代码。封装的这个特性使我们的代码具有可维护性、灵活性以及扩展性。

- 搞对象，定义人**类**。

  类是对某一类事物的描述，是抽象的、概念上的定义；对象是实际存在的该类事物的个体，因而也称实例（Instance）。类和对象就如同概念和实物之间的关系一样，类就好比是一个模板，而对象就是该模板下的一个实例。

  - 定义属性

```
public class Person {
  /**
  *定义属性
  *属性名不能与JAVA关键词冲突
  */
    private String name;

    private double height;

    private char gender;

    private double weight;

    private int faceValue;

    private String mind;
}
```

  修饰符都是private的了，我咋使用捏?

  - 添加 getter / setter方法。设置 / 取得类的属性

```
public class Person {

    private String name;

    private double height;

    private char gender;

    private double weight;

    private int faceValue;

    private String mind;

    public String getName() {
        return name;
    }

    public void setName(String name) {
```

```
        this.name = name;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double height) {
        this.height = height;
    }

    public char getGender() {
        return gender;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    public int getFaceValue() {
        return faceValue;
    }

    public void setFaceValue(int faceValue) {
        this.faceValue = faceValue;
    }

    public String getMind() {
        return mind;
    }

    public void setMind(String mind) {
        this.mind = mind;
    }
}
```

为什么要用this?

this关键字

表示当前对象

一般用在一个方法里的参数名和对象的属性名相同的时候用来分辨你究竟要调用哪个参数。

- 对象实例化

```
public class Demo {
    public static void main(String[] args) {
        Person girl = new Person();
        girl.setFaceValue(6);
        girl.setName("myGirl");
        girl.setGender('女');
        girl.setHeight(160);
        girl.setWeight(50.1415926);
        girl.setMind("merry");

        System.out.println(girl.getName());
    }
}
```

这样写太麻烦了，要不停地调用对象的方法。

能否在创建对象实例时就把内部字段全部初始化为合适的值?

- 构造方法

```
public class Person{

     private String name;

    private double height;

    private char gender;

    private double weight;

    private int faceValue;

    private String mind;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double height) {
        this.height = height;
    }

    public char getGender() {
        return gender;
    }

    public void setGender(char gender) {
        this.gender = gender;
```

```
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    public int getFaceValue() {
        return faceValue;
    }

    public void setFaceValue(int faceValue) {
        this.faceValue = faceValue;
    }

    public String getMind() {
        return mind;
    }

    public void setMind(String mind) {
        this.mind = mind;
    }
  public Person(String name, double height, char gender, double weight, int faceValue, String
 mind) {
        this.name = name;
        this.height = height;
        this.gender = gender;
        this.weight = weight;
        this.faceValue = faceValue;
        this.mind = mind;
    }
 }
```

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，类似这样：

```
class Person {
    public Person() {
    }
}
```

要特别注意的是，如果我们自定义了一个构造方法，那么，编译器就不再自动创建默认构造方法，所以如果既要能使用带参数的构造方法，又想保留不带参数的构造方法，那么只能把两个构造方法都定义出来。——多构造方法。

继承

现在，假设需要定义一个Student类，字段如下：

```
class Student {
    private String name;
    private int age;
    private int score;   //相较于Person多了一项成员变量

    public String getName() {...}
    public void setName(String name) {...}
    public int getAge() {...}
    public void setAge(int age) {...}
    public int getScore() { … }        //多了两个getter, setter方法
    public void setScore(int score) { … }
}
```

```
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：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。只有Object特殊，它没有父类。

super

super关键字表示父类（超类）。子类引用父类的字段（前提是protected修饰的）时，可以用super.fieldName。

super()表示调用父类的构造方法。

大家需要注意的是：在java中，任何一个类的构造方法，第一行一定是调用父类的构造方法，所以如果不显式地调用，则必须要保证父类中有无参构造方法。

但是同样需要注意的是：子类不会继承任何父类的构造方法。子类默认的构造方法是编译器自动生成的，不是继承的。

向上转型（upcasting）& 向下转型（downcasting）

```
Student s = new Student();
Person p1 = new Person();
Person p2 = s; // upcasting, ok
Object o1 = p1; // upcasting, ok
Object o2 = s; // upcasting, ok
Student s1 = (Student) p2; // ok
Student s2 = (Student) p1; // runtime error! ClassCastException!
```

为了避免向下转型出错，我们可以使用 instanceof *操作符*，可以先判断一个实例究竟是不是某种类型：

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

注：instanceof实际上判断一个变量所指向的实例是否是指定类型，或者这个类型的子类。

覆写与重载

	区别	重载	覆写
1	英文表达	overloading	override
2	发生范围	发生在同一个类里面	发生在继承关系之中
3	定义	方法名相同 参数类型和个数不同	方法名相同 参数类型，个数，方法的返回值都要相同
4	权限	没有访问权限控制	被覆写的方法不能比父类中的原方法访问权限还要高

在继承关系中，子类如果定义了一个与父类方法签名以及返回值完全相同的方法，被称为覆写（Override）。

```
public class Main {
    public static void main(String[] args) {
    }
}

class Person {
    public void run() {}
}

public class Student extends Person {
    @Override // Compile error!
    public void run(String s) {}
}
```

```
public static void main(String[] args) {
        Person p = new Student();
        p.run(); // 应该打印Person.run还是Student.run?
    }
}
class Person {
    public void run() {
        System.out.println("Person.run");
    }
}

class Student extends Person {
    @Override
    public void run() {
        System.out.println("Student.run");
    }
}
```

覆写Object方法

因为所有的class最终都继承自Object，而Object定义了几个重要的方法：

- toString()：把instance输出为String；
- equals()：判断两个instance是否逻辑相等；
- hashCode()：计算一个instance的哈希值。

```
class Person {
    ...
    // 显示更有意义的字符串:
    @Override
    public String toString() {
        return "Person:name=" + name;
    }

    // 比较是否相等:
    @Override
    public boolean equals(Object o) {
        // 当且仅当o为Person类型:
        if (o instanceof Person) {
            Person p = (Person) o;
            // 并且name字段相同时，返回true:
            return this.name.equals(p.name);
        }
        return false;
    }

    // 计算hash:
    @Override
    public int hashCode() {
        return this.name.hashCode();
    }
}
```

调用super

在子类的覆写方法中，如果要调用父类的被覆写的方法，可以通过super来调用。例如：

```
class Person {
    protected String name;
    public String hello() {
        return "Hello, " + name;
    }
}

Student extends Person {
    @Override
    public String hello() {
        // 调用父类的hello()方法:
        return super.hello() + "!";
    }
}
```

final

继承可以允许子类覆写父类的方法。如果一个父类不允许子类对它的某个方法进行覆写，可以把该方法标记为final。用final修饰的方法不能被Override：

```
class Person {
    protected String name;
    public final String hello() {
        return "Hello, " + name;
    }
}

Student extends Person {
    // compile error: 不允许覆写
    @Override
    public String hello() {
    }
}
```

如果一个类不希望任何其他类继承自它，那么可以把这个类本身标记为final。用final修饰的类不能被继承：

```
final class Person {
    protected String name;
}

// compile error: 不允许继承自Person
Student extends Person {
}
```

对于一个类的实例字段，同样可以用final修饰。用final修饰的字段在初始化后不能被修改。例如：

```
class Person {
    public final String name = "Unamed";
}
```

对final字段重新赋值会报错：

```
Person p = new Person();
p.name = "New Name"; // compile error!
```

可以在构造方法中初始化final字段：

```
class Person {
    public final String name;
    public Person(String name) {
        this.name = name;
    }
}
```

这种方法更为常用，因为可以保证实例一旦创建，其final字段就不可修改。

over!