面向对象

一、覆盖与重写

```
abstract class C {
    abstract void callme();

    void metoo() {
        System.out.println("在C的metoo()方法中");
    }
}

class D extends C {
    void callme() {
        System.out.println("在D的callme()方法中");
    }
}

public class Learn {
    public static void main(String args[]) {
        C c = new D(); // 创建类D的实例并将其赋值给类型为C的引用
        c.callme();    // 调用类D中重写的callme()方法
        c.metoo();     // 调用类C中的metoo()方法
    }
}
```

解释:

- 类 `C` 被声明为抽象类，具有一个抽象方法 `callme()` 和一个非抽象方法 `metoo()`。
- 类 `D` 扩展了类 `C`，并为抽象方法 `callme()` 提供了实现。
- 在 `learn` 类的 `main` 方法中，创建了类 `D` 的一个实例，并将其赋值给类型为 `C` 的引用变量。
- 当调用 `c.callme()` 时，它调用了类 `D` 中重写的 `callme()` 方法。
- 当调用 `c.metoo()` 时，它调用了类 `C` 中的 `metoo()` 方法，因为在类 `D` 中没有重写该方法。

二、封装的具体使用

```
class Person {
    private String name;
    private int high;

    public void setname(String a){
        if (a.length()>10) {
            System.out.println("错误！");
        }
        else{
            name=a;
        }
    }
```

```
    public String getName(){
        return name;
    }

    }
    public class Mypackage {
        public static void main (String args[]) {
            Person per=new Person();
            per.setname("aaabbbababbbbbbaaa");
            //注意经过private封装后，per.name=aaabbbababbbbbaaa是错误的

            }
        }
```

三、构造方法

1、如果没有定义构造方法，系统会给一个默认的构造方法。

```
class student{

}

    public class Mypackage {
        public static void main (String args[]) {
        student stu=new student();
        System.out.println(stu);
            }
        }
//它输出的是stu的地址
```

2、定义后系统就不提供了

3、如果定义一个带参构造方法，还要用无参数构造方法，就要再写一个无参数构造方法

4、构造方法没有返回值

5、构造方法必须和类名一样

```
class student {
    private String name;
    private int age;

    public student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public student() {
```

```
        this("", 0); // 调用有参构造方法，设置默认值
    }

    //@Override
     public String toString() {
         return "Name: " + name + ", Age: " + age;
    }//构造方法没有返回值，所以要加一个返回方法
}

public class Mypackage {
    public static void main(String args[]) {
        student stu = new student("张三", 18);
        student ent = new student();
        System.out.println(stu); // 输出 stu 对象的信息
        System.out.println(ent); // 输出 ent 对象的信息
    }
}
```

6、静态代码块

```
class Person {
  void say(){
    System.out.println("hello");
  }


  }

  public class exp {
      public static void main(String[] args) {
      Person p2=new Person();
      Person p1=new Person();
      p1.say();
      p2.say();
      p2=null;
      p2.say();
      }
  }
```

在前一个例子中，输出语句 `System.out.println(name);` 在 `Test` 类的静态代码块中。静态代码块在类加载时执行，并且只会执行一次。因此，当 `Test` 类被加载时，静态代码块会执行，并输出变量 `name` 的值 "World"。

7、引用

```
class Person {
  void say(){
    System.out.println("hello");
  }


  }
```

```
  public class exp {
      public static void main(String[] args) {
      Person p2=new Person();
      Person p1=new Person();
      p2.say();
      p1.say();
      p2=null;
      p2.say();
      }
  }
```

在这段代码中，创建了两个 `Person` 类的对象 `p1` 和 `p2`，然后分别调用了它们的 `say()` 方法。

但是，在第三个 `say()` 方法调用之前，将 `p2` 设置为 `null`。这意味着 `p2` 不再引用任何对象，因此再次调用 `p2.say()` 时会引发 `NullPointerException` 异常，因为你尝试调用一个空引用的方法。

四、继承

```
class staff{
  public void method(){
   System.out.println("I came from school.");
  }
}

class teacher extends staff {

}
public class exp {
    public static void main(String[] args) {
      teacher per=new teacher();
      per.method();
    }
}
```

因为teacher继承于staff，所以`class teacher`那是空的也是允许的。

五、final

1、final类

```
class person1{
  public void say() {
  System.out.println("hello");
  }
}

final class person2 extends person1 {
public void say(){
    System.out.println("hi^hi^");
}

}
```

```
public class Myclass {
    public static void main(String args[]) {
       person2 per=new person2();
       per.say();
    }
}
```

final类不可以被继承（太监类），可以继承别人。

2、final成员方法

```
class person1{
    String a,b;
    public final void hear(String a,String b){
this.a=a;
this.b=b;
    }
  public void say() {
  System.out.println("hello");
  }

}

class person2 extends person1 {
public final void say(){
    System.out.println("hi^hi^");
}

}

public class Myclass {
    public static void main(String args[]) {
       person2 per=new person2();
       per.say();
    }
}
```

final类不可以被别人覆写，但可以覆写别人。

3、final局部变量

```
class person1{


}

class person2 extends person1 {

}

public class Myclass {
    public static void main(String args[]) {
       person2 per=new person2();
```

```
        int num1=10;
        System.out.println(num1+"\n");
        num1=20;
        System.out.println(num1+"\n");

        final int num2=200;
        System.out.println(num2+"\n");
        //num2=100    //写法错误
        //num2=200    //写法错误

        final int num3;
        num3=30;//写法正确，只要保证一次赋值即可
    }
}
```

4、final成员变量

```
class person1{
    private final String name;//这里它必须手动赋值

    public person1(){
        name="张三";
    }

    public person1(String name){
        this.name=name;
    }//每个构造方法都要有手动赋值，不然可能会出现无法赋值的情况而报错

    public String getname(){
        return name;
    }//每个构造方法都要有手动赋值，不然可能会出现无法赋值的情况而报错

    //不能再次赋值，final修饰的成员变量只能赋值一次
// public void setname(String name){
//     this.name=name;
// }
}

class person2 extends person1 {

}

public class Myclass {
    public static void main(String args[]) {

    }
}
```

六、super

```
class Person {
    int age=18;
    }
```

```
    class student1 extends Person{
    int age=17;
    public int say(){
      return super.age;
    }
    }

public class example {
    public static void main(String[] args) {
        student1 stu=new student1();
        System.out.println(stu.say());
  }
}
```

这里输出的是18。

super指的是父类文件中的关键字

七、抽象类

```
abstract class animal1{
public void say1(){
  System.out.println("I'm animal1.");
}

abstract public void say2();//抽象方法必须被实现
}

abstract class animal2 extends animal1{

   public void say2(){
    System.out.println("I'm animal2.");
   }

   abstract public void say3();//抽象方法必须在抽象类中
  }

class animal3 extends animal2{

    public void say3(){
     System.out.println("I'm animal3.");
    }

   }

abstract class animal4 {

   }//抽象类可以为空

public class example {
    public static void main(String[] args) {
       animal3 ani=new animal3();
       ani.say1();
       ani.say2();
```

```
        ani.say3();
    }
}
```

八、接口

```
interface interfa {
    //接口中的抽象方法修饰符必须是public, abstract
    public abstract void method1();
    //public与abstract可以省略
    void method2();

    //接口默认方法就是接口的非抽象方法，不用覆写
    public default void methodDefault(){
        System.out.println("这是新添加的默认方法");
    }
}

//接口所有方法必须被实现
class interfaimpl implements interfa {
    public void method1(){
        System.out.println("方法实现1");
    }

    public void method2(){
        System.out.println("方法实现2");
    }
}



public class suminterfa {
    public static void main(String[] args) {
    interfaimpl in=new interfaimpl();
    in.method1();
    in.method2();
    in.methodDefault();
        }
}
```

九、多态

```
class fu {
    int mun=20;
    int age=24;
public void methodmulti(){
    System.out.println("父类方法");
}

public void methodspecial(){
    System.out.println("父类特有方法");
}
}
```

```
class zi extends fu {
    int age=26;
    public void methodmulti(){
        System.out.println("子类方法");
    }
    public void methodzi(){
        System.out.println("子类特有方法");
    }
}

class su extends zi {
    int age=22;
}

/*
 * 多态的格式：
 * 父类名称 对象名=new 子类名称();
 * or
 * 接口名称 对象名=new 实现类名称();
 */

 //只能往上找
 //如果子类有对应方法就调用该方法，没有就调用父类的方法
 //成员方法：编译看左边，运行看右边。
//成员变量：都看左边
public class Multi {
    public static void main(String[] args) {
        fu mul=new zi();
        mul.methodmulti();
        mul.methodspecial();

//fu中没有methodzi()，所以编译错误
        //mul.methodzi();错误方法
        System.out.println(mul.mun);
    }
}
```

十、Object类

```
/*
 * 常见的boolean equal()判断两个对象是否相等,String toString()返回字符串表示形式都属于Object类
 * 所有类都是Object类的子类
 */

class h{

}

public class obj {
    public static void main(String[] args) {
        h a=new h();
```

```
        System.out.println(a.toString());
    }
}
```

十一、内部类

1、成员内部类

```
/*
 * 成员内部类格式
 * 修饰符 class 外部类名称{
 * 修饰符 class 内部类名称{
 * }
 * }
 * 内用外，随意访问；外用内，要内部类对象。
 * 使用方式：
 * 1、间接方式：如下代码
 * 2、直接方式：
 * 外部类名称.内部类名称 对象名=new 外部类名称().new 内部类名称();
 */

class Body1 {

class Heart {
    void methodheart(){
        System.out.println("It's heart.");
    }
    }


void methodbody(){
    System.out.println("It's body");
    new Heart().methodheart();
}
}

public class in {
    public static void main(String[] args) {
        Body1 bo=new Body1();
        bo.methodbody();
    }

}
```

2、局部内部类

```
/*
 * 局部：只有当前方法才能使用它
 * 在一个类的方法中的一个类，就是局部内部类
 * 局部内部类声明处什么都不能写
 */

class outer {
```

```
    void methodouter(){
        class inner{
            int mun=10;
            public void methodinner(){
                System.out.println(mun);
            }
        }
        new inner().methodinner();
    }

}

 public class partin {
    public static void main(String[] args) {
        outer ou= new outer();
        ou.methodouter();
    }
}
```

3、匿名内部类

```
interface Inte {
void methodi();
}

public class pri {
    public static void main(String[] args) {
        Inte inte =new Inte() {
            public void methodi(){
System.out.println("匿名内部类实现的方法1");
            }
        };
        inte.methodi();

        new Inte() {
            public void methodi(){
                System.out.println("匿名内部类实现的方法2");
                            }
        }.methodi();;
    }
}
```