1. **运行JAVA程序**

**（1）在doc窗口下运行Java文件（已安装JDK）**

①java的编译：Javac 文件名.java，产生一个.class文件（字节文件）。

注意：一个类只能产生一个class文件。

如果JAVA文件放在一个包里，需要加入包的储存路径。

例如：JAVA文件放在文件夹com文件夹中的bit文件夹中，则首先在程序内导入包，导入包的格式为（package com.bit;）

然后再编译，编译命为javac com\bit\类.java，若程序中文字较多，则应用（javac -encoding UTF-8 com\bit\ 文件名.java）。

②运行生成的class文件

命令为（java 文件名），如果有包，则命令为（java com.bit.类）

注意：若要查看java反汇编，则用命令（javap -c 文件名进行访问）

1. **在集成环境上运行**

①在Eclipse下运行

②在Idea下运行。

注意：个人更建议使用Idea，因为IntelliJ本身就自带了众多的功能（如：GitHub的集成）。当然，在Eclipse你也可以通过选择不同版本的插件来获取到足够的功能，只是需要自己来配置这些插件。具体情况可以去网上查阅。

1. **JAVA的语言基础**

（1）**先看一个例子，现有一个名为(HelloWord.java)文件，其内部代码为：**

public class HelloWord {

public static void main(String[] args) {

System.out.println("HelloWord");

}

}

在该段程序中：

public：访问修饰限定符 ,还有 private 、 protected 、 default（默认访问权限）。 其中访问权限由大到小的顺序为： private< default（默认访问权限）< protected<public.

class：修饰类

HelloBit：类名称,《阿里巴巴规约手册》中要求用大驼峰命名法。该名称需和文件名一致。

其它的命名规范如下：

函数，用变量的命名（小驼峰），类名（大驼峰），包名（小写）

main():主函数

static：静态类型关键字。

Println:其中后缀ln表示换行，若只有Print则为不换行打印，其中也可用printf()。

**（2）标识符与关键字**

**1.标识符**

①标识符不能为关键字，变量名不能以数字开头。

②Java语言严格区分大小写的，例如：republic和Republic是两个完全不同的标识符。

③由于Java使用的unicode字符集，因此标识符可以用各种语言（如汉文，日文，韩文等都可以）

2.**关键字**

用于修饰访问限定：private 、 protected 、 default、public

用于定义类，函数，变量修饰的：abstract(抽象）、final（修饰常量）、static（静态）、synchronized（处理线上安全）

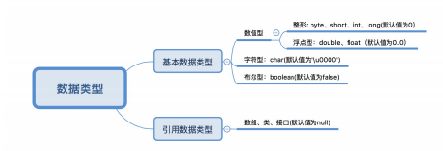
用于定义类与类之间的关系：extends、implements（用于接口的实现）

用于建立实例，引用实例的关键字：new、this、super、instanceof

用于异常处理的关键字：try、catch、finally、throw、throws

用于包的关键字：package（导入文件包）、import（导入工具包）

1. **数据类型**



各数据类型的取值范围



其中大的数据类型转为小的需强转，如：

Long num1=26637448;

Int num2=(int) num1;

1. **Java中逻辑控制和方法的使用**

在Java中，程序有三种分支结构：顺序结构、分支结构和循环结构。

**（1）分支语句**

1.if语句

if语句⼀共有两种形式：

①if(布尔表达式){

//条件满⾜时执⾏代码

}

else{

//条件不满⾜时执⾏代码

}

②if(布尔表达式){

//条件满⾜时执⾏代码

}

else if(布尔表达式){

//条件满⾜时执⾏代码

}

else{

//条件都不满⾜时执⾏代码

}

1. switch语句

语法：switch(数字|枚举|字符|字符串){

case 内容1 : {

内容满⾜时执⾏语句;

[break;]

}

case 内容2 : {

内容满⾜时执⾏语句;

[break;]

}

default:{

内容都不满⾜时执⾏语句; [break;]

}

}

注意：switch（）括号中的参数不能用float、double、long、long long。但字符串和枚举可作为switch的参数

**switch开关语句，若case之后没有break，则会满足到case之后的所有语句一直执行到break或全部结束**

1. **循环语句**
2. while循环语句

while(循环结束条件判断){

循环语句;

修改修改循环结束判断;

}

1. do-while循环语句

do{

循环语句; 修改循环结束判断;

}while(循环结束条件判断);

注意：其中循环体放do里面，while中的结束条件相当于for循环中的第二条件。

While与do-while相比较，while最大特点是如果条件不成立，一次也不执行，而do-while至少执行一次。因此在学习中用的do-while语句很少。

1. for循环

for循环语法：

for(循环初始化;循环结束判断;修改循环结束条件){

循环体;

}

**一般情况下，若知道循环次数用for循环，不知道次数用while循环；**

注意;若for循环中第二条件不写，那就意味着没有循环结束条件，即为死循环。

例如：打印任意个数的乘法口诀表

import java.util.Scanner;

public class Test{

public static void Multiplication(int n){

for(int i=1;i<=n;i ++){

for(int j=1;j<=i;j ++){

System.out.print(i+"\*"+j+"="+i\*j+"\t");

}

System.out.println();

}

}

public static void main(String[] args){

Scanner input = new Scanner(System.in);

int i = input.nextInt();

Multiplication(i);

}

}

注意：此乘法口诀运用了方法的使用，在main函数中是对Multiplication函数的调用。

Continue和break都可以控制循环语句，continue是执⾏到此语句时会跳过当前循环的剩余部分，返回循环判断。而break是退出整个循环。

**（3）方法重载（overload）**

函数名相同，参数列表不同，返回值不做要求。

例如：

public class Test{

public static int add(int x,int y){

return x+y;

} public static int add(int x,int y,int z){

return x+y+z;

}

public static void main(String[] args) {

System.out.println(add(5,5));

System.out.println(add(5,5,55));

}

}

**（4）方法递归（指一个方法自己调用自己）**

递归⽅法的特点：

①⽅法必须有递归的结束条件

②⽅法在每次递归处理的时候⼀定要作出⼀些变更

例如：汉莫塔问题

public class Test {

public static void move(char pos1, char pos2) {

System.out.println(pos1 + "===>" + pos2);

}

public static void hanioc(int n, char pos1, char pos2, char pos3) {

if (n == 1) {

move(pos1, pos3);

return;

} else {

hanioc(n - 1, pos1, pos3, pos2);

move(pos1, pos3);

hanioc(n - 1, pos2, pos1, pos3);

}

}

public static void main(String[] args) {

hanioc(1, 'A', 'B', 'C');

System.out.println("============");

hanioc(2, 'A', 'B', 'C');

System.out.println("============");

hanioc(3, 'A', 'B', 'C');

hanioc(64, 'A', 'B', 'C');

}

}

1. **Java中数组的定义与使用**

**（1）一维数组**

**1.动态初始化数组**

**例如：**public class ArrayDemo{

public static void main(String[] args) {

int[] x = new int[3] ; // 开辟了⼀个⻓度为3的数组

System.out.println(x.length) ;

x[0] = 1 ; // 数组第⼀个元素

x[1] = 2 ; // 数组第⼆个元素

x[2] = 3 ; // 数组第三个元素

for (int i = 0; i<x.length;i++){

System.out.println(x[i]) ; // 通过循环控制索引下标更改

}

}

}

注意：数组属于引用数据类型，因此在使用之前一定要为其开辟空间。

从以上程序可以看出：

数组长度：array.length;

定义数组;int[] array=new int[];

打印数组的两种方法：

①for(int i=0;i<array.length;i++){

System.out.println(array[i]);

}

②for(int i :array）{

System.out.println(i);

}

③System.Out.println(Arrays.toString(array)).使用该方法是需导入Arrays工具包（import java.until.Arrays)

第二种方法不能拿到数组的下表，但第一种可以。

浅拷贝:当两个引用同时指向一块内存的情况下，可以看作为浅拷贝；

**2.静态初始化数组**

**例如:**public class ArrayDemo{

public static void main(String[] args) {

int[] x = {1,2,5,55,555,223,45545,666465,6443} ; // 静态初始化定义数组 System.out.println(x.length) ;

for (int i = 0; i<x.length;i++){

System.out.println(x[i]);

}

}

**（2）二维数组**

**1.初始化方式：**

①动态初始化

数据类型[][] 对象数组=new 数据类型[行个数][列个数]

②静态初始化

数据类型[][] 对象数组 = new 数据类型[][]{{值，值，..},{值，值，..},..} ;

1. 打印方式;

①for（int x=0;x<data.length;x++){

for(int y=0;y<data[x].length;y++){

System.out.println(array[i][j]+" "）

}

}

②System.out.println(Arrays.deepToString(array)）。

**（3）JAVA对数组的支持**

Arrays.binarySearch（array，5）二分查找法;

Arrays.sort(array) 数组排序；

**（4）数组的拷贝方式**

for循环;

for(int i=0;i<array.length;i++){

System.out.println(array[i]);

}

clone（克隆）

array2=array.clone()

System.Arraycopy(原数组名，原数组开始下标，拷贝后的数组名，拷贝后数组开始的下标)（拷贝速度最快)；

int i=Arrays.copy（数组名，开始下标）（有返回值需要定义一个数组去接收）

注意：其中这四种拷贝对于基本类型都是深拷贝，对于引用类型都是浅拷贝。

**（5）可变参数编程**

public class Test{

public static int add(int...array){

int sum=0;

int i=0;

for(i=0 ;i < array.length; i++){

sum+=array[i];

}

return sum;

}

public static void main(String[] args){

int a=add(10,20);

int b=add(10,20,30);

int c=add(10,20,30,40);

int[]array={10,20,30};

System.out.println(a);

System.out.println(b);

System.out.println(c);

System.out.println(add(array));

System.out.println(add(new int[]{10,20,30,40}));//匿名数组

}

}

1. **面向对象编程-类与对象**

**（1）面向对象的三大特征**:封装、继承、多态

**（2）一、类的组成：**属性（变量，描述每个对象的具体特点)方法（操作的行为）

具体语法：class 类名称 {

属性1；

属性2；

属性n...;

⽅法1（）{}

⽅法2（）{}

⽅法n（）{}...

}

**二、有了类就能产生对象，产生对象的方法：**

对象的产生需要两步;

1. 为对象开辟内存
2. 调用合适的构造函数,构造函数不止一个。构造函数可以发生重载。

对象产生的格式：类名称 对象名称=new 类名称（）；

**三、有了对象需要对对象初始化，对象的初始化方式有三种，分别为：**

1、提供一些列的get和set方法(public)：

2、通过构造函数，构造方法遵循的原则是：

一.⽅方法名称必须与类名称相同

二.构造⽅方法没有返回值类型声明

三.每⼀个类中⼀一定⾄至少存在⼀一个构造⽅方法（没有明确定义，则系统⾃自动⽣生成⼀一个⽆无参构造）。

3、静态代码块 实例代码块。

**例如：get和set方法：**

class Person {

private String name;

private int age;

public void getPersonInfo() {

System.out.println ( "姓名：" + name + "年龄：" + age );

}

public void setAge(int age){

this.age=age;

}

public int getAge(){

return age;

}

public void setName(String name){

this.name=name;

}

public String getName(){

return name;

}

}

public class Test{

public static void main(String[] args) {

Person person =new Person();

person.setName ( "bit" );

person.setAge(10);

System.out.println ( person.getName());

System.out.println ( person.getAge());

}

}

**通过构造函数**

class Person {

private String name;

private int age;

public int data = 10;

public static int count = 0;//静态用类明调例如（int n=Person.count）方法区

public Person(){

this(name,age)//调用其他构造函数

System.out.println("Person.init()");//不带参数的构造函数

}

public Person(String name,int age) { //有参构造函数

this.name = name;

this.age = age;

System.out.println("Person.init(String,int)");

}

public static void function() { //静态函数只能用类名调用System.out.println("Person.static.function()");

}

}

public class Test{

public static void main(String[] args) {

int n=Person.count;

System.out.println (n );//静态不属于类，只能用类名调用

Person.function();//用类名调用静态函数

this.function();//调用类中不是静态的函数

Person person =new Person();//实例化对象

System.out.println (person );//打印无参构造函数

Person person2 =new Person("bit",10); //实例化对象

System.out.println (person2 );//有参构造函数打印

}

}

实例代码块和静态代码块

class Person {

private String name;

private int age;

public int data = 10;

public static int count=0;

static{ //静态代码块只能用静态数据，不调用可打印

count=100;

System.out.println("static{}");

}

{//实例代码块（不能访问静态数据）

this.data=99;

System.out.println("instance{}");

}

}

public class Test2{

public static void main(String[] args) {

Person person=new Person();

System.out.println (person );

}

}

**从以上代码分析可知对象的初始化顺序为：**

**静态代码块——实例代码块——构造函数**

**被static所修饰的不依赖于对象，需要类名调用，且先被执行，并且只会被初始化一次。**

1. **this关键字：**
2. this调用本类属性 this调用本类方法 this表示当前对象的引用
3. 使⽤用this调⽤用构造⽅方法时请注意：

一、this调⽤用构造⽅方法的语句句必须在构造⽅方法⾸⾏

二、 使⽤用this调⽤用构造⽅方法时，请留留有出⼝；

（**4）内部类**

内部类：所谓的内部类就是在一个类的内部进行其他类结构的嵌套操作。

内部类的分类：静态内部类、本地内部类、匿名内部类、实例内部类。

内部类存在的分析：

1. 内部类⽅法可以访问该类定义所在作⽤域中的数据，包括被 private 修饰的私有数据；

2. 内部类可以对同⼀包中的其他类隐藏起来；

3. 内部类可以实现 java 单继承的缺陷。

内部类与外部类的关系;

1.对于⾮静态内部类，内部类的创建依赖外部类的实例对象，在没有外部类实例之前是⽆法创建内部类的

2.内部类是⼀个相对独⽴的实体，与外部类不是is-a关系

3.内部类可以直接访问外部类的元素(包含私有域)，但是外部类不可以直接访问内部类的元素

4.外部类可以通过内部类引⽤间接访问内部类元素

内部类的访问方法：

Outerclass.Innerlass innerclass=Outerclass. New Innerclass();

**（5）继承的定义与使用**

**i**mport java.util.Scanner;

class A{ //定义父类,只有名字无年龄

private String name;

public String setName(String name){

this.name=name;

return name;

}

}

class B{ //定义父类B，只有年龄无名字

private int age;

public int SetAge(int age){

this.age=age;

return age;

}

}

class Out{ //定义外部类

private String school;

private class Int1 extends A{ //定义内部类Int1

private int age;

Scanner scanner = new Scanner(System.in);

String name = scanner.nextLine ( );

public Int1 (int age ){ //内部类构造函数

this .age=age;

}

public int getAge() {

return age;

}

public String getName(){

return super.setName (name); //内部类调用父类（即调用A类）

}

} //内部类Int2完

private class Int2 extends B{ //定义内部类Int2

private String name;

Scanner input = new Scanner(System.in);

int age= input.nextInt ();

public Int2(String name){ //内部类构造函数

this.name=name;

}

public String getName(){

return name;

}

public int getAge(){

return super.SetAge (age ); //内部类调用父类（即调用B类）

}

} //内部类Int2完

public Out(String school){ //外部类构造函数

this.school=school;

}

Scanner input = new Scanner(System.in);

int age= input.nextInt ();

Scanner scanner = new Scanner(System.in);

String name = scanner.nextLine ( );

public String person1() { //外部类方法

Int1 person1 = new Int1 ( age ); //外部类调用内部类 Int1

return ("姓名："+person1.getName ()+" \t"+"年龄："+person1.getAge ()+" \t"+"学校："+school+"\t");

}

public String person2(){ //外部类方法

Int2 person2=new Int2(name); //外部类调用内部类 Int1

return ("姓名："+person2.getName ()+" \t"+"年龄："+person2.getAge ()+" \t"+school+" \t");

}

}

public class Test {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String school= scanner.nextLine ( );

Out people=new Out(school);

System.out.println (people.person1 () );

System.out.println (people.person2 () );

}

}

继承的定义：继承的主要作用在于，在已有的基础上继续进行功能的扩充。

在Java中用extends实现继承，被继承者成为（基类、父类或超类），比如以上程序的A类和B类称为基类；继承A类和B类的称为（子类、派生类）；

派生类继承了基类，继承除构造函数外的其他东西。派生类需要帮助基类构造。派生类对象实例化之前需要先实例化基类对象。

被 final 所修饰的类属于密封类不能被继承。被private所修饰的类也不能被子类所继承。

Java可实现多继承：

正确形式： class A{}；

class B extends A{}；

class C extends B{}；

错误形式：class A{}；

class B{}；

class C extends A,B{}；

多态：基类引用 引用派生类对象，并且基类和派生类有同名的覆盖(重写)方法

继承总结：

1. 继承的语法以及继承的⽬的（扩展已有类的功能，使代码重⽤）

2. ⼦类对象的实例化流程：不管如何操作，⼀定要先实例化⽗类对象。

3. 不允许多重继承，只允许多层继承。

super关键字：

1. super调用基类构造函数；
2. Super调用基类属性；
3. Super调用基类成员方法。

overload：重载：函数名相同 参数列表不同 返回值不要求；

override:重写：函数名相同 参数列表相同 返回值相同。

invokespecial （ 构造函数）；invokevirtual （ 普通方法 虚函数）、invokestatic （静态方法）

1. **抽象类与接口**
2. **抽象类**

抽象类：是在普通类的基础上扩充了⼀些抽象⽅法⽽已，所谓的抽象⽅法指的是只声明⽽未实现的⽅法（即没有⽅法体）。所有抽象⽅法要求使⽤abstract关键字来定义，并且抽象⽅法所在的类也⼀定要使⽤abstract关键字来定义，表示抽象类。

1. **抽象类中包含有抽象⽅法，⽽抽象⽅法不包含⽅法体，即没有具体实现。因此抽象类不能直接产⽣实例化对象。**
2. **对于抽象类的使⽤原则：**
3. 所有的抽象类必须有⼦类。

2.抽象类的⼦类必须覆写抽象类的所有抽象⽅法（⼦类不是抽象类）【⽅法覆写⼀定要考虑权限问

题，权限尽量都⽤public】

3.抽象类的对象可以通过对象多态性利⽤⼦类为其实例化

4.private与abstract不能同时使⽤。

**（4）**1.抽象类⼀定不能使⽤fifinal声明，因为使⽤fifinal声明的类不允许有⼦类；⽽抽象类必须有⼦类；相应的，抽象⽅法也不能使⽤private定义，因为抽象⽅法必须要能被覆写

1. 抽象类也分为内部抽象类和外部抽象类。内部抽象类中也可以使⽤static定义来描述外部抽象

**例如：模板设计模式**

以下引例是对抽象类以及模板谁模式的应用：

网上购物过程：

1.浏览商品

2.选择商品

3.客服咨询

4.结算

5.支付

6.配送

7.查看订单

import java.util.Scanner;

abstract class OnlineShopping{

public final void BrowseGoods(){

System.out.println ("1.浏览商品" );

}

public final void SelectGoods(){

System.out.println ("2.选择商品" );

}

public abstract void CallService();

public final void ComputerPrice(){

System.out.println ("4.结算" );

}

public abstract void OuderPay();

public abstract void SendService();

public boolean isCallService(){

return true;

}

final void process(){

BrowseGoods();

SelectGoods();

if(isCallService()) {

CallService ();

}

ComputerPrice();

OuderPay();

SendService();

}

}

class Skycat extends OnlineShopping {

@Override

public void CallService() {

System.out.println ( "3.您好，天猫客服为你服务！" );

}

@Override

public void OuderPay() {

System.out.println ( "5.微信支付" );

}

@Override

public void SendService() {

System.out.println ( "6.圆通配送！" );

}

public String Answer() {

System.out.println ( "是否需要客服服务？ 是：y 不是：n" );

Scanner scanner = new Scanner ( System.in );

String str = scanner.nextLine ( );

return str;

}

public boolean isCallService() {

String str = Answer ( );

return true;

}

else return false;

}

}

public class 购物过程 {

public static void main(String[] args) {

Skycat sktcat=new Skycat();

sktcat.process ();

}

}

**（5）接口的定义与使用**

接口的定义：接⼝就是抽象⽅法和全局常量的集合，在Java中接口使⽤interface关键字定义。

定义接口使用关键字interface 接口的名称使用 I 来表示；

接口中的数据成员默认为 （public static final），接口中的方法默认为 （public abstract）。

接口内不能有已经实现的方法，接口不能进行实例化。IMessage imessage = new IMessage();

接口可以继承多个接口，此时的接口拥有了继承的接口里面的抽象方法。

如果一个抽象类继承了接口，那么抽象类当中可以不实现接口的方法。但是，如果再有一个普通类继承了此抽象类，那么普通类一定要实现接口和抽象类里面的抽象方法。

当⼀个⼦类即需要实现接⼝⼜需要继承抽象类时，请先使⽤extends继承⼀个抽象类，⽽后使⽤implements实现多个接口。

⼀个抽象类可以使⽤implements实现多个接⼝，但是接⼝不能继承抽象类。

接口的工厂模式（简单工厂模式、工厂方法模式、抽象工厂模式、代理设计模式）

此处以抽象工厂模式举例，其他见博客：

public interface Computer {

void printComputer();

}

class MacbookProcComputer implements Computer{

@Override

public void printComputer() {

System.out.println ("这是一个苹果电脑！" );

}

}

class SurfacebookComputer implements Computer{

@Override

public void printComputer() {

System.out.println ("这是一个外星人笔记本电脑！" )；

}

}

interface OperateingSystem{

void printSystem();

}

class MacSystem implements OperateingSystem{

@Override

public void printSystem() {

System.out.println ("这是一个苹果系统！" );

}

}

class SurfaceSystem implements OperateingSystem{

@Override

public void printSystem() {

System.out.println ("这是一个外星人系统！" );

}

}

interface ProductionFactory{

Computer createComputer();

OperateingSystem createSystem();

}

class AppleFactory implements ProductionFactory{

@Override

public Computer createComputer() {

return new MacbookProcComputer ();

}

@Override

public OperateingSystem createSystem() {

return new MacSystem ();

}

}

class AlineFactory implements ProductionFactory{

@Override

public Computer createComputer() {

return new SurfacebookComputer ();

}@Override

public OperateingSystem createSystem() {

return new SurfaceSystem ();

}

}

class Client{

public void buyComputer(Computer computer){

computer.printComputer ();

}

public void use(OperateingSystem s){

s.printSystem ();

}

}

public class Test {

public static void main(String[] args) {

ProductionFactory factory=new AppleFactory ();

Client client =new Client ();

client.buyComputer ( factory.createComputer () );

client.use ( factory.createSystem() );

}

}

1. **抽象类与接口的区别**



1. **三大特殊类**

**String类**

**（1）**class Test {

public int data1 = 10;//堆

public static int data2 = 20;//方法区

public static final int DATA = 30;//方法区

public final int DATA2 = 40;//堆

public void fun() {

int a = 10;//栈

int b = 30;

//Person p = new Person();

}

}

1. **字符串相等的比较：**

1.使用equals比较字符串的内容。格式为（str1.equals(str)）

2.== 本身是进⾏数值⽐较的，如果现在⽤于对象⽐较，那么所⽐较的就应该是个对象所保存的内存地址数值⽐较，⽽并没有⽐较对象的内容。那么要想⽐较内容⽐较，则必须采⽤String类提供的equals⽅法。

**（3）String类与equals的区别**

1. ”==”：进⾏的数值⽐较，⽐较的是两个字符串对象的内存地址数值。

2. “equals（）”：可以进⾏字符串内容的⽐较

**（4）String 、String Buffer、StringBuild的区别和联系**：

1.StringBuffer 和 StringBuilder 也是操作字符串的两个类；

2.、StringBuffer有synchronized（处理线程安全的），但是StringBuilder没有， StringBuffer多线程情况下使用，synchronized线程安全的关键字， StringBuilder没有 ，String 没有 ，单线程情况下 ，线程不安全。

3.拼接上：String每次都会产生新的空间，而StringBuffer StringBuilder ==》append() ==>不会产生新的空间。

4.String的拼接底层被优化为StringBuilder，append进行拼接 结果将会调用StringBuilder的toString()，String str = "hello";str = str+"bit";System.out.println(str);

5.字符串与字符数组的转换：

String str = "helloworld" ;

char[] data = str.toCharArray() ;

if（！Charactor.isDigit(str.charAtci)）//判断一个数是否为字符

**例子：**string类对字符串的一些操作

public static void main(String[] args) {

String str1 = "abcdefbc";//3

//把第一个参数替换为第二个参数

String str2 = str1.replaceAll("bc","gg");

System.out.println(str2);

String str3 = str1.replaceFirst("bc","gg");

System.out.println(str3);

}

//找到第一个不相同的字符，进行比较

//如果长度不一样，

public static void main4(String[] args) {

String str1 = "abcdefbc";//3

String str2 = "abc";//3

System.out.println(str1.equals(str2));

System.out.println(str1.equalsIgnoreCase(str2));

System.out.println(str1.compareTo(str2));

System.out.println(str1.contains("ac"));

System.out.println(str1.indexOf("bc",15));

System.out.println(str1.lastIndexOf("bc",4));

System.out.println(str1.startsWith("ab",2));

System.out.println(str1.startsWith("d"));

System.out.println(str1.endsWith("d"));

System.out.println(str1.endsWith("c"));

System.out.println(str1.endsWith("bc"));

}

**Object类**

1. Object是Java默认提供的⼀个类。Java⾥⾯除了Object类，所有的类都是存在继承关系的。默认会继承Object⽗类。即，所有类的对象都可以使⽤Object进⾏接收。

2.通过以上代码发现，默认Object类提供的toString()⽅法只能够得到⼀个对象的地址（⽽这是所有对象都共同具备的特征）。如若觉得默认给出的toString()⽅法功能不⾜，就在需要的⼦类上覆写toString()⽅法。

3在之前已经分析了Object可以接收任意的对象，因为Object是所有类的⽗类，但是Obejct并不局此，它可以接收所有数据类型，包括：类、数组、接⼝。

**包装类**

1.将基本数据类型包装为⼀个类对象的本质就是使⽤Object进⾏接收处理。

Java中有8种数据类型，如果每种数据类型都按照以上⽅式编写，存在以下问题：

2.开发中代码重复

3. 在进⾏数学计算的时候，必须利⽤明确的⽅法将包装的数据取出后才可以进⾏运算。

所以Java为了⽅便开发，专⻔提供了包装类的使⽤，⽽对于包装类的使⽤，提供了两种类型。

对象型(Object的直接⼦类):Boolean、Character(char)；

数值型(Number的直接⼦类): Byte、Double、Short、Long、Integer(int)、Float；

说明：关于Number类

4.Number类的定义:public abstract class Number implements java.io.Serializable.

5.在Number类⾥⾯实际定义有六种重要⽅法

6. String变为int 类型（Integer类）：public static int parseInt(String s) throws

NumberFormatException

7. String变为double类型（Double类）：public static double parseDouble(String s) throws

NumberFormatException

8. String变为Boolean类型（Boolean类）：public static boolean parseBoolean(String s)

1. **面向对象开发总结**
2. **导入文件包的格式:**package [www.bit.java.util](http://www.bit.java.util)

**（2）类使⽤class和public class的区别：**

1. public class: ⽂件名称必须与类名称保持⼀致，如果希望⼀个类被其他包访问，则必须定义为public class .

2. class: ⽂件名称可以与类名称不⼀致，在⼀个\*.java中可以定义多个class，但是这个类不允许被其他包所访问。

**（3）jar命令**

jar本质上也是⼀种压缩⽂件，⾥⾯保存的都是\*.class⽂件。也就是说现在要实现某⼀个功能模块，可能

有⼏百个类，最终交付给⽤户使⽤时，为了⽅便管理，就会将这些⽂件形成压缩包提供给⽤户。

在JDK中提供实现jar⽂件操作的命令，只需要输⼊⼀个jar即可。

**（4）单例设计模式和多例设计模式**

**一、单例设计模式**

1.懒汉式单例设计模式：

例如：class Singleton {

private static Singleton INSTANCE=new Singleton();

private Singleton(){

System.out.println ("singleton.init()" );

}

public static Singleton getInstance(){

return INSTANCE;

}

public void print(){

System.out.println ("Singleton.getInstance ()" );

System.out.println ("Hello Word" );

}

}

public class Mac{

public static void main(String[] args) {

Singleton singleton =null;

singleton=Singleton.getInstance ();

singleton.print ();

}

}

2.饿汉式单例设计模式

例如：class Singleton{

private static Singleton instance;

private Singleton(){

System.out.println ("singleton.init()" );

}

public static Singleton getInstance(){

if(instance==null){

instance=new Singleton ();

}

return instance;

}

public void print(){

System.out.println ("Singleton.getInstance ()" );

}

}

public class Mac{

public static void main(String[] args) {

Singleton singleton=null;

singleton=Singleton.getInstance ();

singleton.print ();

}

}

**二、多例设计模式**

例如;class Sex{

private String title;

public static final int MALE\_FLAG=1;

public static final int FEMALE\_FLAG=2;

private static final Sex MALE=new Sex("男");

private static final Sex FEMALE=new Sex("女");

private Sex(String title){

this.title=title;

}

public static Sex getInstance(int flag){

switch(flag){

case MALE\_FLAG:

return MALE;

case FEMALE\_FLAG:

return FEMALE;

default:

return null;

}

}

public String tostring(){

return this.title;

}

}

public class Mac{

public static void main(String[] args) {

Sex male=Sex.getInstance (Sex.MALE\_FLAG );

System.out.println ("male" );

}

}

**（5）异常处理的几种格式，以代码为例：**

1.public class Mac{

public static void main(String[] args) {

System.out.println ("[1].数学计算开始前" );

try{

System.out.println ("2.进行数学计算："+10/0 );

}

catch(ArithmeticException e){

2.public class Mac{

public static void main(String[] args) {

System.out.println ("[1].数学计算开始前" );

try{

System.out.println ("2.进行数学计算："+10/0 );

}

catch(ArithmeticException e){

e.printStackTrace ();

}

finally {

System.out.println ("[finally]不管是否异常，都执行此语句" );

}

System.out.println ("3.数学计算结束" );

}

}

3.public class Mac {

public static void main(String[] args) {

System.out.println ( "[1].数学计算开始前" );

try {

int x = Integer.parseInt ( args[0] );

int y = Integer.parseInt ( args[1] );

System.out.println ( "[2].进行数学计算：" + x / y );

} catch (ArithmeticException e) {

e.printStackTrace ( );

} finally {

System.out.println ( "[finally]不管是否异常，都执行此语句" );

}

System.out.println ( "3.数学计算结束" );

}

}

二、throws关键字(主方法抛出异常）

public class Mac{

public static int calculate(int x,int y)throws Exception{

return x/y;

}

public static void main(String[] args)throws Exception{

System.out.println (calculate ( 10,0 ) );

}

}

e.printStackTrace；

System.out.println ("3.数学计算结束" );

}

}

三、throw 关键字（是直接编写在语句中，表示认为异常地抛出

public class Mac{

public static void main(String[] args) {

try{

throw new Exception ( "抛出异常玩玩