

王海涛

# 开发环境

JDK

Idea

# JDK

Java开发库，Java Development Kit

JDK中包含：

* 一组开发工具 jdk\bin
* 基础类库 jdk\jre\lib\rt.jar
* 基础类库的源码 jdk\src.zip

## 下载安装 JDK

必应“JDK8 download”，第一条是官方下载页面

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

选择对应操作系统的版本下载

电脑屏幕截图

描述已自动生成

## 环境变量

JAVA\_HOME

PATH

CLASSPATH (可选)

### JAVA\_HOME

配置JDK的安装目录路径，其他开发工具，或其他Java程序，运行时，通过JAVA\_HOME变量，来找到JDK目录

JAVA\_HOME=d:\java\jdk1.8

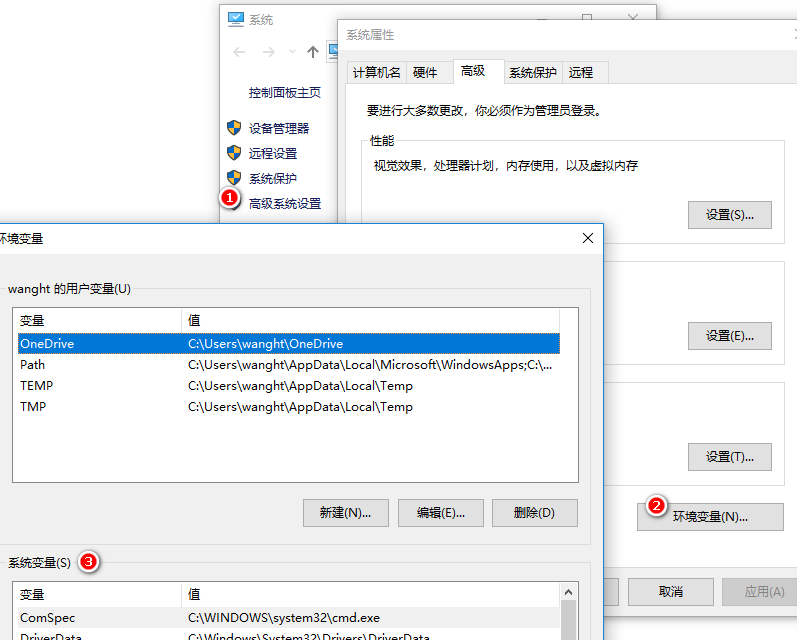
c:\program files\java\jdk1.8

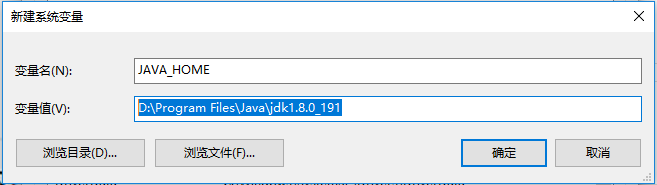
c:\java\jdk1.8

d:\software\jdk1.8\_45

d:\java\jdk1.8

电脑 -> 属性 -> 高级系统设置 -> 环境变量 -> 系统变量





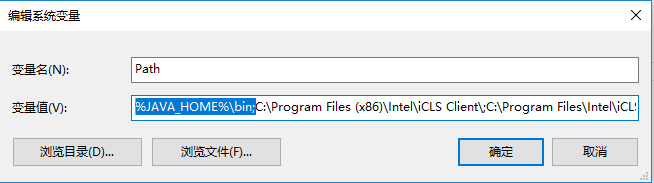
### PATH

path变量是系统中已经存在的变量，在系统中执行一个命令，系统在path配置的目录路径中，依次查找可执行文件

要在系统中，执行jdk的开发工具命令，要把jdk的bin目录路径，添加到path变量

d:\software\jdk1.8\_45\bin;原来的内容

%JAVA\_HOME%\bin;原来的内容



### 验证环境变量

WIN+R，输入 cmd，(打开命令行终端)

输入命令：

echo %JAVA\_HOME%

D:\Program Files\Java\jdk1.8.0\_191

path

PATH=D:\Program Files\Java\jdk1.8.0\_191\bin;C:\Program Files (x86)\Int......

java -version

java version "1.8.0\_191"

Java(TM) SE Runtime Environment (build 1.8.0\_191-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)

### CLASSPATH (可选)

只配置一个“点”就可以

CLASSPATH=.

# Idea

集成的开发环境（IDE）

把jdk的命令行开发工具，集成起来，提供一个窗口环境

* eclipse
* IntelliJ IDEA

## Idea下载

必应“idea download”第一条官方下载页面

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

# 第一个Java程序 Hello world!

1. 新建 Java 项目
2. 新建“类”： d0101.HelloWorld

包名：d0101 小写

类名：HelloWorld 驼峰

1. 添加代码：

public static void main(String[] args) {

System.out.println("Hello world!!!");

}

1. 保存 ctrl + s，保存时自动编译
2. 运行测试 shift+f10

**package** d0101;

/\*

\* 新版本 eclipse

\* ctrl+shift+数字键盘加减

\* ctrl+大键盘加减

\*

\* 旧版eclipse

\* window->preference->General->Appearance->Color and Fonts

\* Basic->Text Font

\*

\* eclipse黑暗主题

\* window->prefernece->Genderal->Appearance

\* Theme 选择 Dark

\*/

**public** **class** HelloWorld {

**public** **static** **void** main(String[] args) {

System.***out***.println("Hello world!!!");

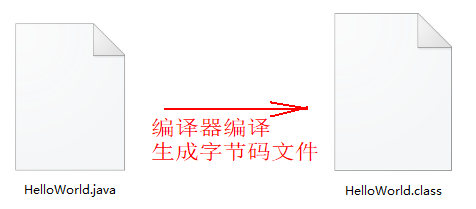
System.***out***.println("师姐你好！！！");

}

}

## 编译

src目录下的HelloWorld.java 编译成 HelloWorld.class 字节码文件，存放在 bin 目录下



## 执行

* 先启动Java虚拟机（JVM）
* JVM加载HelloWorld.class字节码文件
* 进入这个类的 main() 方法执行

1. 录入个人信息

类：d0101.Test2

**package** d0101;

//导包，指明路径，后面代码可以简写类名

**import** java.util.Scanner;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

System.***out***.println("录入个人信息：");

System.***out***.print("姓名：");

//获得输入的值

//再保存到变量 name

String name =

**new** Scanner(System.***in***).nextLine();

System.***out***.print("性别：");

String gender = **new** Scanner(System.***in***).nextLine();

System.***out***.print("年龄：");

**int** age = **new** Scanner(System.***in***).nextInt();

System.***out***.println("您录入的个人信息：");

System.***out***.println(name+", "+gender+", "+age);

System.***out***.println(

"我在达内一定要努力磨炼技术，"

+ "比旁边同学多写一遍代码，"

+ "薪水一定要比旁边同学高");

}

}

# 数据类型（两类）

基本类型（8种）

引用类型

# 八种基本类型

|  |  |  |  |
| --- | --- | --- | --- |
| byte | 1 | -128到127，-27到27-1 | |
| short | 2 | -215到215-1 | |
| int | 4 | -231到231-1 | |
| long | 8 | -263到263-1 | |
| float | 4 | 单精度 | IEEE-754规范,美国电子电气工程师协会的浮点数表示规范 |
| double | 8 | 双精度 |
| char | 2 | UNICODE编码，0-65535 | |
| boolean | 1 | true: 00000001  false:00000000 | |

Byte如何表示整数：

-128 - 10000000

…

-3 - 11111101

-2 - 11111110

-1 – 11111111 用0一位一位的借位减

0 – 00000000

1 – 00000001

2 – 00000010

3 – 00000011

4 – 00000100

…

127 - 01111111

1. 基本类型

项目：d0103\_基本类型

类：d0103.Test1

**package** d0103;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//定义八个变量abcdefgh，

//来保存四种整数的最小值和最大值

**byte** a = -128;

**byte** b = 127;

//调用jdk类库中的 Short 类

//其中的 MIN\_VALUE 常量，

//保存的就是short类型的最小值

**short** c = Short.***MIN\_VALUE***;

**short** d = Short.***MAX\_VALUE***;

**int** e = Integer.***MIN\_VALUE***;

**int** f = Integer.***MAX\_VALUE***;

**long** g = Long.***MIN\_VALUE***;

**long** h = Long.***MAX\_VALUE***;

System.***out***.println(a);

System.***out***.println(b);

System.***out***.println(c);

System.***out***.println(d);

System.***out***.println(e);

System.***out***.println(f);

System.***out***.println(g);

System.***out***.println(h);

}

}

Test2

**package** d0103;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

//四个变量 abcd 保存两个浮点数

//的最小值和最大值

**float** a = Float.***MIN\_VALUE***;

**float** b = Float.***MAX\_VALUE***;

**double** c = Double.***MIN\_VALUE***;

**double** d = Double.***MAX\_VALUE***;

System.***out***.println(a);

System.***out***.println(b);

System.***out***.println(c);

System.***out***.println(d);

}

}

输入两个整数，保存到两个变量

交换两个变量的值

项目：d0201\_变量交换

类：d0201.Test1

**package** d0201;

**import** java.util.Scanner;

/\*

\* window->preference->java->editor->

\* content assist->auto activation

\*

\* tiggers 添加字符小写a到z，大写a到z，和下划线

\* .abcdefghijklmnopqrstuvwxyz\_ABCDEFGHIJKLMNOPQRSTUVWXYZ

\*

\* 避免干扰输入，可以设置提示的延迟时间 500

\*/

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("输入整数a：");

//先获得输入的整数

//再保存到变量 a

**int** a = **new** Scanner(System.***in***).nextInt();

System.***out***.print("输入整数b：");

**int** b = **new** Scanner(System.***in***).nextInt();

System.***out***.println("a = "+a);

System.***out***.println("b = "+b);

System.***out***.println("--交换----------");

//三句交换自己再写一遍

//比别人多写一遍！！！！

//下次再写变量交换能直接写出来，不写错！！！

**int** t = a;

a = b;

b = t;

System.***out***.println("a = "+a);

System.***out***.println("b = "+b);

}

}

* 1. **char**

Unicode编码的字符，或字符的整数编码

|  |  |
| --- | --- |
| 0 | ? |
| 1 | ? |
| 2 | ? |
| 3 |  |
| ... |  |
| 97 | a |
| 98 | b |
| ... |  |
| 20013 | 中 |
| ... |  |
| 65535 |  |

char c1 = 'a';

char c2 = 97;

char c3 = '中';

char c4 = 20013;

1. **char**

项目：d0202\_基本类型

类：d0202.Test1

**package** d0202;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

**char** c1 = 'a';

**char** c2 = 97;

**char** c3 = 'b';

**char** c4 = 98;

**char** c5 = '中';

**char** c6 = 20013;

System.***out***.println(c1);

System.***out***.println(c2);

System.***out***.println(c3);

System.***out***.println(c4);

System.***out***.println(c5);

System.***out***.println(c6);

System.***out***.println('王'+0);

System.***out***.println('海'+0);

System.***out***.println('涛'+0);

}

}

1. **基本类型的字面值规则（5条）**
2. **整数字面值，是int类型**

int a = 74563;

//错，右侧字面值是int，但超出范围

long a = 9999999999;

1. **byte,short,char三种比int小的整数，可以用范围内的值直接赋值**

byte a = 127; //右侧是byte

byte a = 128; //错，右侧是 int

1. **浮点数字面值，是 double 类型**

double d = 3.14;

float f = 3.14; //错，右侧字面值是double

1. **字面值的后缀**

L - long

long a = 9999999999L;

F - float

float f = 3.14f;

D - double

double d = 3.0;

double d = 3d;

1. **字面值前缀**

0b - 二进制

0b101110011011011

0x - 16进制

0xfa8c10f

0 - 8进制

001

002

003

007

\u - char类型16进制

'\u0061'

'\u4e2d'

System.out.println('\u0061');

System.out.println('\u4e2d');

System.out.println('\u738B');

System.out.println('\u6D77');

System.out.println('\u6D9B');

1. **基本类型的运算规则（5条）**
2. **计算结果的数据类型，与算项的最大类型一致**

3/2 int/int 结果是int

3d/2 double/int 结果是 double

1. **byte,short,char三种比int小的整数，运算时，会自动转成 int**

byte a = 3;

byte b = 4;

byte c = a + b; //错，int + int，结果是int

1. **整数运算溢出**

整数运算，可以理解为像时钟转圈，超出范围不出错，而是转到一个错误的结果

Integer.MAX\_VALUE + 1

得负数最小值

光年计算：

300000000\*60\*60\*24\*365 溢出的错误结果

300000000L\*60\*60\*24\*365 long类型正确结果

形状

描述已自动生成

1. **浮点数运算不精确**

IEEE-754规范规定了计算机中浮点数的表示格式，根据这个格式规则，在做某些浮点数运算时，就会产生不精确结果

2-1.9 得 0.100000000009

4.35\*100 得 434.99999999999994

* Java提供了方法，可以做精确运算，使用BigDecimal，后面再讲

1. **浮点数的特殊值**

**Infinity**

无穷大

3.14 / 0

**NaN**

Not a Number

Math.sqrt(-6)

1. **运算规则**

Test2

**package** d0202;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

System.***out***.println(3/2);

System.***out***.println(3d/2);

// \n 是换行符的转义符

System.***out***.println("\n-----------------");

**char** c1 = 'S';

**char** c2 = 'B';

System.***out***.println(c1 + c2);//int + int

System.***out***.println("\n-----------------");

System.***out***.println(Integer.***MAX\_VALUE***+1);

System.***out***.println(300000000\*60\*60\*24\*365);

System.***out***.println(300000000L\*60\*60\*24\*365);

System.***out***.println("\n-----------------");

System.***out***.println(2-1.9);

System.***out***.println(2-1.8);

System.***out***.println(2-1.7);

System.***out***.println(2-1.6);

System.***out***.println(2-1.5);

System.***out***.println(4.35\*100);

System.***out***.println(4.36\*100);

System.***out***.println(4.37\*100);

System.***out***.println(4.38\*100);

System.***out***.println(4.39\*100);

System.***out***.println("\n-----------------");

System.***out***.println(3.14/0);

System.***out***.println(Math.*sqrt*(-6));

}

}

1. **基本类型的类型转换**

数字类型之间，可以互相转换，从小到大转换，可以自动完成转换；从大到小转换，需要强制转换

图示

描述已自动生成

* 从小到大

byte a = 127;

int b = a;

* 从大到小

int a = 356;

//切掉三个字节，保留1个字节

//[1][2][3][4] ---> [4]

byte b = (byte) a; // 100

1. **牛郎织女**

牛郎织女星相距 16.4 光年

光速 299792458 米/秒

喜鹊身长 0.46 米

牛郎织女相会，需要多少只喜鹊

项目：d0203\_牛郎织女

类：d0203.Test1

**package** d0203;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//1光年是多少米

**long** ly = 299792458L\*60\*60\*24\*365;

//16.4光年是多少米

**double** d = 16.4 \* ly; // double \* long 是 double

//喜鹊数量

**double** r = d/0.46;

//Math.ceil(r) 对 r 向上取整

//35345.345 --> 35346.0 double类型的结果

//double 转成 long 整数类型

**long** n = (**long**) Math.*ceil*(r);

System.***out***.println(n);

}

}

1. **运算符**

|  |  |
| --- | --- |
| +-\*/ |  |
| + | 字符串连接运算 |
| % | 取余、求模 |
| == | 相等比较 |
| != | 不等比较 |
| > >= < <= | 大小比较 |
| && | **逻辑与**，左边是真并且右边是真结果才是真  **短路与**，左边是假，右边忽略不执行 |
| || | **逻辑或**，左边是真或者右边是真结果都是真  **短路或**，左边是真，右边忽略不执行 |
| ! | 非，非真是假，非假是真 |
| ++  -- | 自增、自减运算  a = 6;  print(a++);  1)先取出a的原值 6 临时存到一个内存空间  2)a再自增变成 7  3)打印原值 6  a = 6;  print(++a);  1)先自增变成7  2)再取出a的新值 7 临时存到一个内存空间  3)打印 a 的新值7 |
| ? : | 三项运算符、三目运算  1 ? 2 : 3  1是真，取2的值  1是假，取3的值 |
| = | 赋值运算符 |
| +=, -=... | 复合的赋值运算符  a = a+2;  a += 2;  会自动转型  byte a = 3;  a = (byte) (a+2);  a += 2;  a += 3.9; |
| () | 转型运算 |

* 1. **运算符的优先级**

运算符优先级不需要看，多加小括号

* 想让谁先算谁先算
* 便于代码阅读理解，可读性更强

(1+(2-3))\*4/5

1. **平年闰年**

闰年的条件：

\*) 能被4整除，并且不能被100整除

\*) 能被400整除

项目：d0204\_平年闰年

类：d0204.Test1

**package** d0204;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

闰年的条件：

\*) 能被4整除，并且不能被100整除

\*) 能被400整除

2000 闰年

1000 平年

2016 闰年

2019 平年

\*/

System.***out***.print("输入年号：");

**int** y = **new** Scanner(System.***in***).nextInt();

//定义一个字符串变量 r="平年"

//后面如果判断 y 是闰年，把 r 修改成 "闰年"

String r = "平年";

//y能被4整除并且不能被100整除，

//或者，y能被400整除

**if**((y%4==0 && y%100!=0) || y%400==0) {

r = "闰年";

}

System.***out***.println(y+"年是"+r);

}

}

1. **自增自减**

项目：d0205\_自增自减

类：d0205.Test1

**package** d0205;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* 添加断点

\* f11 启动调试，在断点位置暂停

\* f6 单步执行

\*/

**int** a = 6;

System.***out***.println(a++);//先取值6，打印取出的6

System.***out***.println(a);

System.***out***.println("\n-----------------");

a = 6;

System.***out***.println(++a);//后取值7，打印取出的7

System.***out***.println(a);

System.***out***.println("\n------------------");

a = 6;

**int** b = a++;//先取值6，再自增成7，把取出的6赋给b

System.***out***.println(a);

System.***out***.println(b);

System.***out***.println("\n------------------");

a = 6;

a = a++;//先取值6，再自增成7，把取出的6赋给a

System.***out***.println(a);

System.***out***.println("\n------------------");

a = 6;

a++;

System.***out***.println(a);

System.***out***.println("\n--再写变量交换----------------");

a = 3;

b = 4;

//交换

**int** t = a;

a = b;

b = t;

//打印

System.***out***.println(a);

System.***out***.println(b);

}

}

1. **三个数的最大值**

输入三个整数，取最大值

项目：d0206\_三个数的最大值

类：d0206.Test1

**package** d0206;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("输入整数a：");

**int** a = **new** Scanner(System.***in***).nextInt();

System.***out***.print("输入整数b：");

**int** b = **new** Scanner(System.***in***).nextInt();

System.***out***.print("输入整数c：");

**int** c = **new** Scanner(System.***in***).nextInt();

//取 a,b 的最大值，存到变量 max

//取 max,c 的最大值，再存到变量 max

//int max = a>b ? a : b;

//max = max>c ? max : c;

//int max = a<b ? b : a;

//max = max<c ? c : max;

**int** max = a>b ? (a>c?a:c) : (b>c?b:c);

System.***out***.println("最大值："+max);

System.***out***.println("\n--再写变量交换------------");

// max 和 c 交换，交换后打印

**int** t = max;

max = c;

c = t;

System.***out***.println("max = "+max);

System.***out***.println("c = "+c);

}

}

1. **方法**

命名的代码块，方便重复调用

static void f() {

...

}

f()

f()

f()

1. **方法**

项目：d0207\_方法

类：d0207.Test1

**package** d0207;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* 添加断点

\* f11 启动调试

\* f6 单步执行

\* f5 进入方法

\*/

System.***out***.println(1);

*f*();

System.***out***.println(2);

*f*();

System.***out***.println(3);

*f*();

}//main方法结束

**static** **void** f() {

System.***out***.println(4);

}

}

1. **二进制**

计算机中用 0 和 1 表示数据，一个电位是 bit，8个bit是一个字节

计算机可以用10进制来和人交互

0 0

1 1

10 2

11 3

100 4

1. **二进制**

项目：d0208\_二进制

类：d0208.Test1

1. 练习

* 输入时间（秒），求自由落体距离

输入降落时间（秒）：2.5

2.5秒降落了 xx.xx 米

* 输入半径，求圆的面积和周长

1. **流程控制**

* if
* switch
* for
* while
* do-while
* break
* continue
* 循环命名

1. **if**

if(条件) {

代码

}

if(条件) {

代码1

} else {

代码2

}

if(条件1) {

代码1

} else if(条件2){

代码2

} else if(条件3){

代码3

} else {

代码4

}

1. **商品打折**

满500，9折

满1000，85折

满3000，8折

满5000，7折

项目：d0301\_商品打折

类：d0301.Test1

**package** d0301;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("输入原价：");

**double** p = **new** Scanner(System.***in***).nextDouble();

//调用 f() 方法，把p变量的值，传递到f()方法

*f*(p);//取出p的值，保存到方法参数变量x

}//双击大括号，找到main结束

**static** **void** f(**double** x) {//新变量x，和上面的变量p无关

/\*

满500，9折

满1000，85折

满3000，8折

满5000，7折

\*/

//定义一个折扣变量 r=1

//根据原价范围，要给r重新赋值

//如果下面条件都不成立，r不会被重新赋值

**double** r = 1;

**if**(x>=5000) {

r = 0.7;

} **else** **if**(x>=3000) {

r = 0.8;

} **else** **if**(x>=1000) {

r = 0.85;

} **else** **if**(x>=500) {

r = 0.9;

}

x \*= r; //x = x\*r;

//四舍五入到小数点后2位

/\*

\* x = 5.57496345

\* 乘 1000， 5574.96345

\* 转成 long，5574

\* 取个位数，9

\* 如果<5，减掉余数，5570

\* 否则，先加10，再减掉余数 5580

\* 除1000

\*/

x \*= 1000;

**long** n = (**long**) x;

**int** y = (**int**) (n % 10); //long % int

**if**(y<5) {

n -= y;

} **else** {

n = n+10 - y;

}

x = n/1000d; //long/double, double

System.***out***.println(

"折扣："+r+"， 实价："+x);

}

}

1. **switch**

switch只能判断：

* byte,short,char,int
* enum
* jdk1.7，String

从成立的case，向后无条件穿透执行所有 case ，包括default，直到结束，或者遇到 break 中断跳出switch

switch(变量或表达式) {

case 30:

case 10:

break;//中断、跳出

case 60:

case 20:

default:

}

1. **switch**

输入年、月，求这个月的天数

项目：d0302\_天数

类：d0302.Test1

**package** d0302;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("年：");

**int** y = **new** Scanner(System.***in***).nextInt();

System.***out***.print("月：");

**int** m = **new** Scanner(System.***in***).nextInt();

//调用 f() 方法，把 y 和 m 的值传递到f()方法

*f*(y, m);

System.***out***.println("---------------");

}//双击大括号，找到main结束

**static** **void** f(**int** y, **int** m) {

/\*

\* 1,3,5,7,8,10,12 -- 31

\* 4,6,9,11 -- 30

\* 2 -- 平年28，闰年29

\*/

//月是错误的值

**if**(m<1 || m>12) {

System.***out***.println("月份值输入错误");

//返回，控制程序执行的流程，

//返回到调用位置，继续执行

**return**;//方法执行到此结束，后面代码不再执行

}

//定义天数变量 day

//初始值0，后面根据判断，要修改day的值

**int** day = 0;

//判断月份值

**switch**(m) {

**case** 1:

**case** 3:

**case** 5:

**case** 7:

**case** 8:

**case** 10:

**case** 12:

day = 31;

**break**;

**case** 4:

**case** 6:

**case** 9:

**case** 11:

day = 30;

**break**;

**case** 2:

**if**((y%4==0&&y%100!=0)||y%400==0) {

day = 29;

} **else** {

day = 28;

}

}

System.***out***.println(y+"年"+m+"月的天数："+day);

}

}

1. **for循环**

for(变量赋值; 执行条件; 变量修改) {

代码

}

例如打印1到10：

for(int i=1; i<=10; i++) {

System.out.println(i);

}

1. **for循环**

项目：d0303\_for循环

类：d0303.Test1

**package** d0303;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("\n--1到10---------------");

*f1*();

System.***out***.println("\n--10到1---------------");

*f2*();

System.***out***.println("\n--2,4,6,8,10---------------");

*f3*();

System.***out***.println("\n--8,88,888....888888888---------------");

*f4*();

System.***out***.println("\n-----------------");

*f5*();

}//双击大括号

**static** **void** f1() {

**for** (**int** i = 1;i <= 10;i++) {

System.***out***.println(i);

}

}

**static** **void** f2() {

**for** (**int** i = 10;i >= 1;i--) {

System.***out***.println(i);

}

}

**static** **void** f3() {

**for** (**int** i = 2;i <= 10;i+=2) {

System.***out***.println(i);

}

}

**static** **void** f4() {

**for** (**long** i = 8;i <= 888888888;i=i\*10+8) {

System.***out***.println(i);

}

}

**static** **void** f5() {

//死循环，无限循环，打印1,2,3.....

**for**(**int** i=1; ;i++) {

System.***out***.println(i);

}

}

}

1. **约数**

找出一个整数的所有约数

24

1,2,3,4,6,8,12,24

17

1,17

项目：d0304\_约数

类：d0304.Test1

**package** d0304;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("输入整数：");

**int** n = **new** Scanner(System.***in***).nextInt();

//把n变量的值取出来

//把值传递到 f() 方法

*f*(n);//ctrl+1，回车

}

**private** **static** **void** f(**int** n) {

/\*

\* i循环从 1 到 n

\* 找 n 能被 i 整除的情况，打印 i 的值

\*/

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

**if** (n%i == 0) {

System.***out***.println(i);

}

}

}

}

1. **鸡兔同笼**

鸡和兔共35只

共有94只脚

鸡和兔各多少只

穷举法

鸡 兔

0 35

1 34

2 33

3 32

4 31

...

23 12

...

35 0

项目：d0305\_鸡兔同笼

类：d0305.Test1

**package** d0305;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

*f1*();

}

**private** **static** **void** f1() {

/\*

\* 鸡的数量j，从0到35

\* 兔的数量t，从35到0

\*

\* for第一项，可以给多个变量赋值

\* 第三项，可以修改多个变量

\*/

**for** (**int** j=0,t=35; j<=35; j++,t--) {

//如果脚的数量是94，打印这个组合

**if** (j\*2 + t\*4 == 94) {

System.***out***.println(

"鸡："+j+"，兔："+t);

}

}

}

}

1. **方法的返回值**

方法的计算结果数据，可以传回到调用位置

如果方法不返回值，用 void

void f() {

}

如果方法有返回值，要定义返回值的数据类型

int f() {

}

1. **阶乘**

5!

5\*4\*3\*2\*1

项目：d0306\_阶乘

类：d0306.Test1

**package** d0306;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("输入整数，求阶乘：");

**int** n = **new** Scanner(System.***in***).nextInt();

//把n变量的值取出来，传递到f()方法

**long** r = *f*(n);

System.***out***.println(r);

}

/\*

\* r = n

\*

\* i

\* n-1, r=r\*i

\* 3, r=r\*i

\* 2, r=r\*i

\* 1, r=r\*i

\*/

**private** **static** **long** f(**int** n) {

//如果参数n的值是0，直接返回1

**if**(n == 0) {

**return** 1;

}

**long** r = n;

//i循环从n-1到1

**for** (**int** i = n-1; i >= 1; i--) {

r=r\*i;

}

//向调用位置返回计算结果

**return** r;

}

}

1. **循环嵌套**

for(int i=1; i<=3; i++) {

for(char j='a'; j<='c'; j++) {

打印 i, j

}

}

1. **循环嵌套**

d0303\_for循环

Test2

**package** d0303;

**import** java.util.Random;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

System.***out***.println("\n---------------------");

*f1*();

System.***out***.println("\n--打印正方形-------------------");

*f2*();

System.***out***.println("\n--打印三角形-------------------");

*f3*();

System.***out***.println("\n--又打印三角形-------------------");

*f4*();

}

**private** **static** **void** f1() {

**for**(**int** i=1; i<=3; i++) {

**for**(**char** j='a'; j<='c'; j++) {

System.***out***.println(i+", "+j);

}

}

}

**private** **static** **void** f2() {

/\*

\* i

\* 1 \*\*\*\*\*

\* 2 \*\*\*\*\*

\* 3 \*\*\*\*\*

\* 4 \*\*\*\*\*

\* n \*\*\*\*\*

\* 1234n j

\*/

//产生一个随机整数，作为行数，

//范围 5+ [0, 50) --> [5, 55)

**int** n = 5 + **new** Random().nextInt(50);

System.***out***.println(n+"行正方形");

//i循环从第1行到第n行

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

//j循环从第1个星号到第n个星号

**for** (**int** j = 1; j<=n; j++) {

System.***out***.print("\*");

}

//行尾补一个换行

System.***out***.println();

}

}

**private** **static** **void** f3() {

/\*

\* i

\* 1 \*

\* 2 \*\*

\* 3 \*\*\*

\* 4 \*\*\*\*

\* n \*\*\*\*\*

\* 1234i j

\*/

**int** n = 5 + **new** Random().nextInt(50);

System.***out***.println(n+"行三角形");

/\*

\* i循环从第1行到第n行

\* j循环从第1个星号到第i个星号

\*/

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

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

System.***out***.print("\*");

}

System.***out***.println();

}

}

**private** **static** **void** f4() {

/\*

\* i

\* 1 \*

\* 2 \*\*\*

\* 3 \*\*\*\*\*

\* 4 \*\*\*\*\*\*\*

\* n \*\*\*\*\*\*\*\*\*

\*

\* 空格数量从第1个空格到第n-i个空格

\* 星号数量从第1个星号到第2i-1个星号

\*/

**int** n = 5 + **new** Random().nextInt(50);

System.***out***.println(n+"行三角形");

//i循环1到n行

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

//j循环1到n-i个空格

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

System.***out***.print(" ");

}

//j循环1到2i-1个星号

**for** (**int** j = 1; j <= 2\*i-1; j++) {

System.***out***.print("\*");

}

//补换行

System.***out***.println();

}

}

}

1. **作业**

* 重写，比旁边同学多写一遍
  + d0302\_天数
  + d0305\_鸡兔同笼
* 99乘法表

i

1 11

2 12 22

3 13 23 33

4 14 24 34 44

5 15 25 35 45 55

...

9 19 29 39 ........... 99

1 2 3 ...... i j

* Code First

**package** d0407;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

      //调用f()方法，从方法获取一个int[]数组

**int**[] a = *f*();

      //遍历打印数组数据

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

         System.***out***.println(a[i]);

      }

      System.***out***.println("\n\n----------------");

      //把a数组，传递到 shuffle() 方法打乱顺序

*shuffle*(a);

      System.***out***.println(Arrays.*toString*(a));

   }

**private** **static** **void** shuffle(**int**[] a) {

      /\*

       \*        j

       \* [4, 2, 3, 1, 5]

       \*     i

       \*

       \* \*) i循环遍历数组

       \* \*) 随机定位下标j与i交换

       \*/

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

         //随机下标j，范围：[0, a.length)

**int** j = **new** Random().nextInt(a.length);

**int** t = a[i];

         a[i] = a[j];

         a[j] = t;

      }

   }

**private** **static** **int**[] f() {

      //新建int[]数组，长度5

      //再把它的内存地址存到变量 a

**int**[] a = **new** **int**[5];

      //遍历访问5个位置，填入1,2,3,4,5

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

         a[i] = i+1;

      }

      //返回数组，把数组返回到调用位置

      //本质是把数组地址返回去

**return** a;

   }

**package** d0307;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//1到9行

**for** (**int** i = 1; i<=9; i++) {

//1到i

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

**if** (j==3 && (i==3 || i==4)) {

System.***out***.print(" ");

}

System.***out***.print(

j+"\*"+i+"="+(j\*i)+" ");

}

System.***out***.println();

}

}

}

1. **break和continue**
   1. **break**

中断、跳出循环和switch

for(...) {

代码1

if(条件) {

break;

}

代码2

}

....

* 1. **continue**

继续进入循环的下一轮执行

for(...;...; i++) {

代码1

if(条件) {

continue;

}

代码2

}

1. **break和continue**

项目：d0401\_break和continue

类：d0401.Test1

**package** d0401;

/\*

\* 隐藏暂时不使用的项目

\* 1. 关闭项目

\* 2. 过滤隐藏关闭项目

\*/

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("\n------------------");

*f1*();//ctrl+1, create method

System.***out***.println("\n------------------");

*f2*();//ctrl+1, create method

}

**private** **static** **void** f1() {

// 随机浮点数 [0, 1)

//double d = Math.random();//alt+shift+L

//System.out.println(d);

// >0.999

//死循环产生随机值

**for** (**int** i = 1; ;i++) {

**double** d = Math.*random*();

**if** (d > 0.999) {

System.***out***.println("第"+i+"次："+d);

**break**;//中断、跳出

}

}

}

**private** **static** **void** f2() {

//打印 1 到 100

**for** (**int** i = 1; i <= 100; i++) {

//尾数是3,5,7，不打印，直接跳到下一个值

**int** y = i%10;

**if** (y==3 || y==5 || y==7) {

**continue**;//跳到i++，进入循环下一轮

}

System.***out***.println(i);

}

}

}

1. **二进制和十六进制**

0123456789abcdef

1111 f

1101 d

11111101 fd

* 一个16进制字符，可以表示二进制4位
* 两个16进制字符，可以表示二进制的一个字节，8位

10111011 01100111 01101011 01101011

bb 67 6b 6b

输入二进制或十六进制，显示10进制值

"0b101011" --> "101011" --> 345

"0xfa8cef" --> "fa8cef" --> 6345232

项目：d0402\_二进制十六进制

类：d0402.Test1

**package** d0402;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//死循环

**while**(**true**) {

System.***out***.println(

"输入0b开头二进制，或0x开头十六进制，q退出");

String s = **new** Scanner(System.***in***).nextLine();

//字符串比较是否相等，不能用 ==

//要用 equals() 方法

**if**(s.equals("q")) {

**break**;

}

//把s传递到 f() 方法，转成10进制整数

**int** r = *f*(s);

System.***out***.println("10进制："+r);

System.***out***.println("\n---------------------");

}

}

**private** **static** **int** f(String s) {

/\*

\* 参数字符串 s

\* "0B1101011"

\* "0xf9aac5e" --> "f9aac5e"

\* 如果是大写，先变成小写

\* s.toLowerCase()

\*

\* 获取0b或0x后面的数字部分

\* s.substring(2)

\*

\* 判断前缀是0b还是0x

\* s.startsWith("0b")

\*

\* 转成10进制

\* Integer.parseInt("1101011", 2)

\* Integer.parseInt("f9aac5e", 16)

\*/

//转成小写 "0B1101" --> "0b1101"

s = s.toLowerCase();

//截取0b、0x后面的数字部分 "0b1101"--> "1101"

String n = s.substring(2);

//准备一个保存结果的变量 r

**int** r = 0;

//判断前缀是0b还是0x

**if** (s.startsWith("0b")) {

r = Integer.*parseInt*(n, 2);//"1101"-->13

} **else** **if** (s.startsWith("0x")) {

r = Integer.*parseInt*(n, 16);//"d" -->13

}

**return** r;

}

}

1. **循环命名**

在内层循环中，控制外层循环，需要对外层循环命名

outer:

for(...) {

for(...) {

break outer;

continue outer;

}

}

1. **循环命名**

项目：d0403\_循环命名

类：d0403.Test1

**package** d0403;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* 您现在的状态：

\* 1. 好嗨哟

\* 2. 感觉人生已经达到了高潮

\* 3. 感觉人生已经达到了巅峰

\* 4. 我想静静，bye~

\* 选择：> 1

\*/

outer:

**while** (**true**) {

//显示菜单并获得输入的选项值

**int** c = *caiDan*();

**switch**(c) {

**case** 1:

System.***out***.println("好嗨哟");

**break**;

**case** 2:

System.***out***.println("感觉人生已经达到了高潮");

**break**;

**case** 3:

System.***out***.println("感觉人生已经达到了巅峰");

**break**;

**case** 4:

System.***out***.println("我想静静，bye~");

**break** outer;

**default**:

System.***out***.println("输入有误，请重试");

}

}

}

**private** **static** **int** caiDan() {

System.***out***.println("\n------------------------------------");

System.***out***.println("您现在的状态：");

System.***out***.println("1. 好嗨哟");

System.***out***.println("2. 感觉人生已经达到了高潮");

System.***out***.println("3. 感觉人生已经达到了巅峰");

System.***out***.println("4. 我想静静，bye~");

System.***out***.print("选择：> ");

**int** c = **new** Scanner(System.***in***).nextInt();

**return** c;

}

}

1. **while**

先判断，再执行

while(条件) {

代码

}

1. **do-while**

先执行，再判断

代码至少执行一次

do {

代码

} while(条件);

1. **猜数字**

已经随机产生了一个[1, 1001) 范围的整数

请猜这个数是几

猜：1

小

猜：1000

大

猜：2

小

猜：999

对

项目：d0404\_猜数字

类：d0404.Test1

**package** d0404;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//随机整数 1+ [0, 1000)

// 1+ 0,1,2,3,4....999

**int** r = 1+ **new** Random().nextInt(1000);

System.***out***.println(r);

System.***out***.println("已经随机产生了一个[1, 1001) 范围的整数");

System.***out***.println("请猜这个数是几");

//定义变量 c, 保存每次猜的值

**int** c;

//先猜，再判断不相等，继续猜

**do** {

System.***out***.print("猜：");

c = **new** Scanner(System.***in***).nextInt();

**if**(c>r) {

System.***out***.println("大");

} **else** **if**(c<r) {

System.***out***.println("小");

}

} **while** (c != r);

System.***out***.println("对");

}

}

1. **练习**

* 最大公约数和最小公倍数

24 1,2,3,4,6,8,12,24

16 1,2,4,8,16

48

最大公约数，从小值向前找

最小公倍数，从大值向后一直累加大值

1. **面向对象1**

## 入门

### 对象

对象可以把数据，打包在一起

void f(int id, String name, String gender, int age, int height, int weight,

xx,xx,xx,xx,xx,xxx,xx,xx,xx,xx,xxx,xxx,xxx,xxx,xxx,xxx,xxx,xxxx) {

}

f(5,"sd","d",12,...)

定义一个类，是对象模板

学生类 Student（对象模板）

{

int id,

String name,

String gender,

int age,

......

}

学生实例a：

{

id: 6,

name: "张三",

gender: "男",

age: 23

}

f(a);

学生实例b:

{

id: 9,

name: "李四",

gender: "女",

age: 22

}

f(b);

void f(Student a) {

}

* 1. **对象**

项目：d0405\_学生对象

类：d0405.Test1

Student

**Student**

**package** d0405;

/\*

\* 类

\* 学生的对象模板

\* 定义学生对象要包含哪些数据

\*/

**public** **class** Student {

**int** id;

String name;

String gender;

**int** age;

**void** print() {

System.***out***.println(id);

System.***out***.println(name);

System.***out***.println(gender);

System.***out***.println(age);

}

}

**Test1**

**package** d0405;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("录入学生信息：");

/\*

\*

\* Student 类, 学生的对象模板

\* {

\* int id,

\* String name,

\* String gender,

\* int age,

\* print() //模板方法

\* }

\*

\* 新建 Student 实例，存到变量 s1

\* s1 = {

\* id: 6,

\* name: "张三",

\* gender: "男",

\* age: 21

\* }

\*

\*/

Student s1 = **new** Student();

System.***out***.print("学号：");

s1.id = **new** Scanner(System.***in***).nextInt();

System.***out***.print("姓名：");

s1.name = **new** Scanner(System.***in***).nextLine();

System.***out***.print("性别：");

s1.gender = **new** Scanner(System.***in***).nextLine();

System.***out***.print("年龄：");

s1.age = **new** Scanner(System.***in***).nextInt();

//把录入的学生数据，传递到 f() 方法处理

*f*(s1);

}

**private** **static** **void** f(Student s1) {

/\*

\* s1 = {

\* id: xxx,

\* name: "xxx",

\* gender: "xxx",

\* age: xx

\* }

\*/

s1.print(); //让学生实例s1，执行print()方法

}

}

* 1. **坐标点**

图片包含 图示

描述已自动生成

项目：d0406\_坐标点

类：d0406.Test1

Point

Point

**package** d0406;

/\*

\* Point类，坐标点的对象模板

\*/

**public** **class** Point {

**int** x;

**int** y;

**void** fuZhi(**int** x, **int** y) {

//把接收的参数x值，保存到“这个实例”的x

//“这个实例”，“当前实例” - 正在调用的实例

**this**.x = x;

**this**.y = y;

}

**void** print() {

String s = **this**.x + ", " + **this**.y;

System.***out***.println(s);

}

}

Test1

**package** d0406;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* Point类，坐标点的对象模板

\* {

\* int x,

\* int y,

\* fuZhi(x, y),

\* print()

\* }

\*

\* p1 = {

\* x: 3,

\* y: 4

\* }

\*

\* p2 = {

\* x: 1,

\* y: 5

\* }

\*

\* p3 = {

\* x: 8,

\* y: 3

\* }

\*/

Point p1 = **new** Point();

Point p2 = **new** Point();

Point p3 = **new** Point();

p1.fuZhi(3, 4);

p2.fuZhi(1, 5);

p3.fuZhi(8, 3);

p1.print();

p2.print();

p3.print();

}

}

### 构造方法

构造一个实例之后，立即执行的一个特殊方法

new Student()

new Student(参数)

new Point()

new Point(3, 4)

一个类，必须有构造方法

如果自己不定义构造方法，编译器会添加默认构造方法

class Point {

public Point() {

}

}

自己定义构造方法，一般是要给属性变量赋值

class Point {

int x;

int y;

public Point(int x, int y) {

this.x = x;

this.y = y;

}

}

Point p1 = new Point(3, 4);

Point p2 = new Point(1, 5);

Point p3 = new Point(8, 3);

* 1. **圆**

Circle 圆类，圆实例的模板

{

int r,

Point o,

juLi()

}

c1 = {

r: 9,

o: {x:3, y:4}

}

c2 = {

r: 6,

o: {x:8, y:6}

}

项目：d0407\_圆

类：d0407.Test1

Point

Circle

* 1. **类**

模板、图纸

类在第一次用到时，会被加载到内存的“方法区”

文本

低可信度描述已自动生成

* 1. **实例**

从模板创建的具体实例

新建实例时，在“堆内存”中新分配内存空给给这一个实例

图形用户界面, 应用程序

描述已自动生成

* 1. **引用变量**

理解成“遥控器”

引用类型的变量，保存一个实例的内存地址

引用变量的特殊值：null，空，不保存任何实例的地址

图片包含 日历

描述已自动生成

* 1. **构造方法**

新建实例时执行的一个特殊方法

* 1. **this**

this.xxx

this(...)

* 1. **方法重载 Overload**

同名不同参

1. **数组**

用来存放一组数据的数据结构

图片包含 图表

描述已自动生成

* 1. **数组的创建（3种语法）**

int[] a = new int[6];

* 新建 int[] 数组，长度6
* 6个位置都存默认值 0
* 数组的起始地址值，保存到变量a

图标

中度可信度描述已自动生成

int[] a = {6,2,6,8};

* 直接初始化数组数据

a = new int[]{7,3,8,1,7,9,3,1};

* 为存在的数组变量赋值，直接初始化数据，要添加类型

1. **数组**

项目：d0502\_数组

类：d0502.Test1

**package** d0502;

**import** java.util.Arrays;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("\n--------------------");

*f1*();

System.***out***.println("\n--------------------");

*f2*();

System.***out***.println("\n--------------------");

*f3*();

}

**private** **static** **void** f1() {

/\*

\* 先新建 String[]，长度5

\* 再把数组地址保存到变量a

\* a = [null, null, null, "abc", null]

\* 3

\* 在下标3位置放入 "abc"

\*/

String[] a = **new** String[5];

//Arrays.toString(a)

//把数组中的数据，连接成字符串

System.***out***.println(Arrays.*toString*(a));

a[3] = "abc";

System.***out***.println(Arrays.*toString*(a));

}

**private** **static** **void** f2() {

**int**[] a = {6,2,7,3,7,3,1,5};

System.***out***.println(Arrays.*toString*(a));

a = **new** **int**[]{7,2,9,3};

System.***out***.println(Arrays.*toString*(a));

}

**private** **static** **void** f3() {

**int**[] a = **new** **int**[5];

a[2] = 565;

System.***out***.println(a[2]);

System.***out***.println(a[2] \* 2);

a[2]++;

/\* 交换 2,3 位置的值

\* [0, 0, 0, 0, 0]

\* 2 3

\* t = 566 \*/

**int** t = a[2];

a[2] = a[3];

a[3] = t;

System.***out***.println(Arrays.*toString*(a));

}

}

* 1. **数组的长度属性**

a.length

* 数组一旦创建，长度不可变
* 最大下标 a.length - 1
* 允许0长度的数组
  1. **数组遍历**

从头到尾，依次访问每个位置

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

a[i]

}

1. **最小值的位置下标**

d0502\_数组

Test2

**package** d0502;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

//调用 suiJi() 产生一个乱序数组，存到变量a

**int**[] a = *suiJi*();

System.***out***.println(Arrays.*toString*(a));

//把数组，传递到 f() 方法求最小值的位置下标

**int** index = *f*(a);

System.***out***.println(

"最小值的位置下标："+index);

}

**private** **static** **int** f(**int**[] a) {

/\*

\* a = [92, 7, 45, 74, 22, 58, 2, 96, 43, 19]

\*

\* 假设第一个值就是最小的

\* 最小值位置下标 m = 0

\*

\* m = 6

\* a = [92, 7, 45, 74, 22, 58, 2, 96, 43, 19]

\* 0 1 2 3 4 5 6 7 8 9

\*

\* i循环从下标1向后找更小值

\* 找到更小值，把下标保存到m

\*/

**int** m = 0;

**for** (**int** i = 1; i < a.length; i++) {

//a[i] 比 a[m] 更小

**if** (a[i] < a[m]) {

m = i;//更小值的下标，存到m

}

}

**return** m;

}

**private** **static** **int**[] suiJi() {

/\*

\* 新建 int[] 数组，长度10

\* a = [0,0,0,0,0,0,0,0,0,0]

\*

\* 遍历填入 100 内随机整数

\* a = [67,21,7,56,99,78,12,88,54,28]

\*

\* 把数组的地址返回到调用位置

\*/

**int**[] a = **new** **int**[10];

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

a[i] = **new** Random().nextInt(100);//[0, 100)

}

**return** a;

}

}

1. **商品录入查询**

项目：d0503\_商品录入查询

类：d0503.Test1

Item

Item 类

{

id,

name,

price,

number

}

Item[]

items = [{...}, {...}, {...}, {...}, {...}]

**package** d0503;

**import** java.util.Scanner;

**public** **class** Test1 {

/\*

\* 商品数组

\* [{...}, {...}, {...}, {...}, {...}]

\*/

**static** Item[] *items* = {

**new** Item(1,"iPhoneXS",8199,20),

**new** Item(2,"华为P30 pro",5799,80),

**new** Item(3,"小米8",2699,50),

**new** Item(4,"vivo NEX",3599,90),

**new** Item(5,"OPPO FIND",3099,60)

};

**public** **static** **void** main(String[] args) {

/\*

\* 1. 商品列表

\* 2. 商品录入

\* 3. 商品查询

\* 4. 统计信息

\* 5. 退出

\* 选择：> 2

\*/

outer:

**while**(**true**) {

//调用 caiDan() 方法显示菜单并获得选择的值

**int** c = *caiDan*();

//判断选项值

**switch**(c) {

**case** 1: *f1*(); **break**;

**case** 2: *f2*(); **break**;

**case** 3: *f3*(); **break**;

**case** 4: *f4*(); **break**;

**case** 5: **break** outer;

}

}

}

**private** **static** **int** caiDan() {

System.***out***.println("-------------------");

System.***out***.println("1. 商品列表");

System.***out***.println("2. 商品查询");

System.***out***.println("3. 商品录入");

System.***out***.println("4. 统计信息");

System.***out***.println("5. 退出");

System.***out***.println("-------------------");

System.***out***.print("选择：> ");

**int** c = **new** Scanner(System.***in***).nextInt();

**return** c;

}

**private** **static** **void** f1() {

/\*

\* [{...}, {...}, {...}, {...}, {...}]

\* 0 1 2 3 4

\* 遍历显示商品信息

\* items[0].id

\* items[0].name

\* items[0].toString()

\*/

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

//println()方法的内部会自动调用对象的toString()方法

System.***out***.println(*items*[i].toString());

}

}

**private** **static** **void** f2() {

/\*

\* [{...}, {...}, {...}, {...}, {...}]

\* 0 1 2 3 4

\*

\* 查询的商品名：a 存到变量 n

\* n items[i].name

\* 判断商品名称中是否包含 n 子串

\*

\* s.indexOf(子串), 在s中寻找子串的起始位置

\* s = "adfdabcabc"

\* s.indexOf("bc") 1 下标值

\* s.indexOf("xx") -1 特殊值表示不存在

\*/

System.***out***.print("查询的商品名：");

String n = **new** Scanner(System.***in***).nextLine();

//遍历所有商品比对

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

**if**(*items*[i].name.indexOf(n) >= 0) {

System.***out***.println(*items*[i].toString());

}

}

}

**private** **static** **void** f3() {

/\*

\* [{xxx}, {xxx}, {xxx}, {xxx}, {...}]

\* 0 1 2 3 4

\*/

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

System.***out***.println(

"录入第 "+(i+1)+" 件商品：");

System.***out***.print("编号：");

*items*[i].id = **new** Scanner(System.***in***).nextInt();

System.***out***.print("名称：");

*items*[i].name = **new** Scanner(System.***in***).nextLine();

System.***out***.print("价格：");

*items*[i].price = **new** Scanner(System.***in***).nextDouble();

System.***out***.print("数量：");

*items*[i].number = **new** Scanner(System.***in***).nextInt();

}

//重新显示列表

*f1*();

}

**private** **static** **void** f4() {

/\*

\* 商品总价

\* 单价均价

\* 最高总价

\* 最高单价

\*/

**double** spzj = 0;//商品总价

**double** djzj = 0;//单价总价

**double** zgzj = 0;//最高总价

**double** zgdj = 0;//最高单价

//遍历数组

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

Item s = *items*[i];//访问数组中的对象，存到变量s

spzj += s.price \* s.number;//商品总价

djzj += s.price;//单价总价

**if**(s.price\*s.number > zgzj) { //最高总价

zgzj = s.price\*s.number;

}

**if**(s.price > zgdj) { //最高单价

zgdj = s.price;

}

}

System.***out***.println("商品总价："+spzj);

System.***out***.println("单价均价："+(djzj/*items*.length));

System.***out***.println("最高总价："+zgzj);

System.***out***.println("最高单价："+zgdj);

}

}

1. **练习**

* 创建数组，顺序填入 1,2,3,4,....

打乱顺序

* + i循环遍历数组
  + 随机定位一个 j 位置
  + i和j位置的值交换

j

[5,3,7,4,1,6,2]

i

**package** d0504;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

**int**[] a = **new** **int**[10];

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

a[i] = i+1;

}

System.***out***.println(Arrays.*toString*(a));

//把数组 a 传递到 shuffle() 方法打乱顺序

*shuffle*(a);

System.***out***.println(Arrays.*toString*(a));

}

**private** **static** **void** shuffle(**int**[] a) {

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

**int** j = **new** Random().nextInt(a.length);

**int** t = a[i];

a[i] = a[j];

a[j] = t;

}

}

}

# 面向对象2

1. **士兵**

Soldier士兵类

{

id,

blood = 100,

go(),

attack()

}

s1 = {

id: 9527,

blood: 100

}

s2 = {

id: 9528,

blood: 100

}

项目：d0601\_士兵

类：d0601.Test1

Soldier

Soldier

**package** d0601;

**import** java.util.Random;

/\*

\* 封装

\* 把士兵相关的数据、运算方法，封装成一个“类”

\*/

**public** **class** Soldier {

//成员变量、属性变量

**int** id; //默认值0

**int** blood = 100;

//成员方法

**public** **void** go() {

System.***out***.println(**this**.id+"号士兵前进");

}

**public** **void** attack() {

**if**(**this**.blood == 0) {

System.***out***.println("这是"+id+"号士兵的尸体");

**return**;

}

System.***out***.println(**this**.id+"号士兵进攻");

//模拟受到伤害，产生一个随机的减血量

**int** d = **new** Random().nextInt(10);

//如果减血量比当前血量大

**if**(d > **this**.blood) {

d = **this**.blood;//有多少减多少

}

**this**.blood -= d;

System.***out***.println("血量："+**this**.blood);

**if**(**this**.blood == 0) {

System.***out***.println(id+"号士兵阵亡");

}

}

}

Test1

**package** d0601;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* s1 = {

\* id: 9527,

\* blood: 100

\* }

\* s2 = {

\* id: 9528,

\* blood: 100

\* }

\*/

//新建士兵对象，把它的内存地址保存到s1

Soldier s1 = **new** Soldier();

Soldier s2 = **new** Soldier();

s1.id = 9527;

s2.id = 9528;

s1.go();

s2.go();

s2.attack();

s2.attack();

s2.attack();

s2.attack();

s2.attack();

}

}

图示

描述已自动生成

1. **士兵的武器**

d0601\_士兵

复制

d0602\_今晚吃鸡

新建类： Weapon 武器类

{

name,

bullets,

fire()

}

**package** d0602;

**import** java.util.Random;

**public** **class** Weapon {

String name;// 武器名称

**int** bullets;// 子弹数量

**public** Weapon(String name) {

**this**(name, 100);//构造方法之间调用

}

**public** Weapon(String name, **int** bullets) {

**this**.name = name;

**this**.bullets = bullets;

}

**public** **void** fire() {

System.***out***.println("开火");

//产生随机的发射子弹数量

**int** r = 3 + **new** Random().nextInt(10);

**if**(r > **this**.bullets) {

r = **this**.bullets;

}

**this**.bullets -= r;

**for** (**int** i = 0; i < r; i++) {

System.***out***.print("突");

}

System.***out***.println("~");

System.***out***.println("剩余子弹："+bullets);

}

**public** **void** reload() {

**this**.bullets = 100;

}

}

Test2

**package** d0602;

**import** java.util.Scanner;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

//新建两个武器实例，

//调用发射方法、装载方法测试

Weapon w1 = **new** Weapon("AKM");

Weapon w2 = **new** Weapon("M416", 80);

System.***out***.println("回车发射，输入r装载子弹");

**while**(**true**) {

String s = **new** Scanner(System.***in***).nextLine();

**if** (s.equals("r")) {

w1.reload();

w2.reload();

System.***out***.println("弹夹已装满");

**continue**;

}

w1.fire();

w2.fire();

}

}

}

Soldier

**package** d0602;

**import** java.util.Random;

/\*

\* 封装

\* 把士兵相关的数据、运算方法，封装成一个“类”

\*/

**public** **class** Soldier {

//成员变量、属性变量

**int** id; //默认值0

**int** blood = 100;

Weapon weapon; // 默认值 null

//成员方法

**public** **void** go() {

System.***out***.println(**this**.id+"号士兵前进");

}

**public** **void** attack() {

**if**(**this**.blood == 0) {

System.***out***.println("这是"+id+"号士兵的尸体");

**return**;

}

System.***out***.println(**this**.id+"号士兵进攻");

//如果有枪

**if** (**this**.weapon != **null**) {

**if** (**this**.weapon.bullets == 0) {

**this**.weapon.reload();//装载子弹

}

**this**.weapon.fire(); //开火

}

//模拟受到伤害，产生一个随机的减血量

**int** d = **new** Random().nextInt(10);

//如果减血量比当前血量大

**if**(d > **this**.blood) {

d = **this**.blood;//有多少减多少

}

**this**.blood -= d;

System.***out***.println("血量："+**this**.blood);

**if**(**this**.blood == 0) {

System.***out***.println(id+"号士兵阵亡");

}

}

}

Test3

**package** d0602;

**import** java.util.Scanner;

/\*

\* 100个士兵进入刺激战场

\*

\* Soldier[] 数组

\* [

\* {

\* id: 1,

\* blood: 100,

\* weapon: {...}

\* },

\* {

\* id: 2,

\* blood: 100,

\* weapon: {...}

\* }

\* ....

\* ]

\*/

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

System.***out***.println("欢迎进入刺激战场");

//100个士兵，每个士兵拿一把枪

Soldier[] a = **new** Soldier[100];

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

//新建士兵实例，为id和weapon赋值

Soldier s = **new** Soldier();

s.id = i+1;

s.weapon = **new** Weapon("AK"+(i+1));

//把士兵放入数组

a[i] = s;

}

System.***out***.println("按回车执行");

**while**(**true**) {

//暂停等待，按回车后才会继续向后执行

**new** Scanner(System.***in***).nextLine();

//每个士兵进攻一轮

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

**if** (a[i].blood != 0) {//如果有血

a[i].attack(); //执行进攻

System.***out***.println("----");

}

}

System.***out***.println("\n\n---------------------");

}

}

}

1. **二维数组**

存放数组的数组

形状, 矩形

描述已自动生成

* 1. **创建二维数组**

int[][] a = new int[3][2];

* 外围数组长度 3
* 内部的三个数组长度 2
* 共创建了4个数组
* 内部数组中是默认值0
* 外围数组中保存内部数组的地址

int[][] a = new int[3][];

* 只创建一个外围数组，长度3
* 三个位置都是 null 值
* 可以之后再新建数组，放入

a[0] = new int[4];

a[1] = new int[]{5,1,5};

a[2] = new int[]{7,3,7,5,1};

int[][] a = {

{6,2,5,7},

{1,2,5},

{7,3,4,8,3}

};

* 1. **遍历二维数组**

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

for(int j=0;j<a[i].length;j++) {

a[i][j]

}

}

1. **二维数组**

项目：d0603\_二维数组

类：d0603.Test1

**package** d0603;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

**char**[][] a = { //j

{'道','路','千','万','条'}, //0

{'安','全','第','一','条'}, //1

{'行','车','不','规','范'}, //2

{'亲','人','两','行','泪'} //3

}; //i 0 1 2 3 4

/\*

\* 亲行安道

\* 人车全路

\* 两不第千

\* 行规一万

\* 泪范条条

\*/

**for** (**int** i = 0; i < a[0].length; i++) {

**for** (**int** j = a.length-1; j >=0; j--) {

System.***out***.print(a[j][i]);

}

System.***out***.println();

}

}

}

1. **java.util.Arrays**

数组的工具类，提供几个数组操作的工具方法

* **Arrays.toString(数组)**

把数组数据，连接成字符串

* **Arrays.sort(数组)**

数组排序

基本类型数组，优化的快速排序

引用类型数组，优化的合并排序

* **Arrays.binarySearch(数组, 目标值)**

二分法查找、折半查找

在有序数组中，查找目标值的位置下标

找不到，返回 -(插入点+1)

* **Arrays.copyOf(数组, 长度)**

把数组复制成一个指定长度的新数组

1. **数组的复制**

* **Arrays.copyOf(数组, 长度)**

会创建新数组

* **System.arraycopy(**

**原数组,**

**原数组起始位置,**

**目标数组,**

**目标数组起始位置,**

**复制的数据数量)**

不会新建数组，目标数组要事先存在

1. **数组复制**

d0602\_今晚吃鸡

Test3

复制

Test4

阵亡士兵从数组删除，复制一个少一个位置的数组

a = [{},{},{x},{},{},{},{}]

i

a = [{},{},{},{},{},{}]

i

**package** d0602;

**import** java.util.Scanner;

/\*

\* 100个士兵进入刺激战场

\*

\* Soldier[] 数组

\* a = [

\* {

\* id: 1,

\* blood: 100,

\* weapon: {...}

\* },

\* {

\* id: 2,

\* blood: 100,

\* weapon: {...}

\* }

\* ....

\* ]

\*/

**public** **class** Test4 {

**public** **static** **void** main(String[] args) {

System.***out***.println("欢迎进入刺激战场");

//100个士兵，每个士兵拿一把枪

Soldier[] a = **new** Soldier[100];

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

//新建士兵实例，为id和weapon赋值

Soldier s = **new** Soldier();

s.id = i+1;

s.weapon = **new** Weapon("AK"+(i+1));

//把士兵放入数组

a[i] = s;

}

System.***out***.println("按回车执行");

**while**(**true**) {

//暂停等待，按回车后才会继续向后执行

**new** Scanner(System.***in***).nextLine();

//每个士兵进攻一轮

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

//if (a[i].blood != 0) {//如果有血

a[i].attack(); //执行进攻

System.***out***.println("----");

/\*

a = [{},{},{x},{},{},{},{}]

i

a = [{},{},{},{},{},{}]

i

\* \*/

**if** (a[i].blood == 0) {

Soldier[] b = **new** Soldier[a.length-1];

System.*arraycopy*(a,0,b,0,i);

System.*arraycopy*(a,i+1,b,i,a.length-i-1);

a = b;

i--;

}

//}

}

System.***out***.println("\n\n---------------------");

}

}

}

1. **变量**
   1. **局部变量**

定义在方法中，或局部代码块中

* 局部变量必须手动初始化（第一次赋值时，分配内存空间）

int a;

print(a); //错，没有初始化

a = 6; //初始化，分配内存空间

print(a);

* 局部变量的作用域、作用范围，在它定义的大括号内有效
* 在作用范围内，不能重复定义

void f() {

int a = 6;

if(...) {

int a = 8;//错

print(a);

int b = 7;

}

print(b);//错

int b = 9;

}

* 1. **成员变量**

定义在类中

* 自动初始化成默认值
* 访问范围，受访问控制符的控制
* 局部变量，可以和成员变量同名

1. **方法递归**

在方法中，调用自身

void f() {

f();

}

例如用 f() 方法求阶乘

f(5)

f(4)

f(3)

f(2)

f(5)

5\*f(4)

4\*f(3)

3\*f(2)

2\*f(1)

1\*f(0)

递归算法，一步一步地简化问题，简化成最简问题，再反向倒推求出结果

一般不同时做多次递归调用，否则运算量会急剧增加，这时考虑是否能用循环来代替递归

1. **递归求阶乘**

项目：d0604\_递归求阶乘

类：d0604.Test1

**package** d0604;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("输入整数求阶乘：");

**int** n = **new** Scanner(System.***in***).nextInt();

**long** r = *f*(n);

System.***out***.println(r);

}

**private** **static** **long** f(**int** n) {

//最简问题

**if** (n == 0) {

**return** 1;//从最简问题向前返回

}

**return** n \* *f*(n-1); //递归调用

}

}

图示

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

1. **递归求斐波那契数**

1 1 2 3 5 8 13 21 34 55 89 144 233 ...

项目：d0605\_递归求斐波那契数

类：d0605.Test1

**package** d0605;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("求第几个斐波那契数：");

**int** n = **new** Scanner(System.***in***).nextInt();

**long** r = *g*(n);

System.***out***.println(r);

}

**private** **static** **long** g(**int** n) {

/\*

\* 1 1 2 3 5 8

\* a b

\*

\* b=a+b

\* a=b-a

\*/

**long** a = 1;

**long** b = 1;

//从第3个求到第n个

**for** (**int** i = 3; i <= n; i++) {

b=a+b;

a=b-a;

}

**return** b;

}

//反例，多次递归，运算量大

**private** **static** **long** f(**int** n) {

//最简问题

**if** (n==1 || n==2) {

**return** 1;

}

**return** *f*(n-1) + *f*(n-2);

}

}

1. **面向对象3**

## 继承

作用：代码重用、复用

Java的继承是单继承，只能继承一个父类

不能继承：

* 构造方法
* 私有成员

子类对象

* 先新建父类对象
* 再新建子类对象
* 两个对象绑定在一起，整体作为一个对象
* 调用成员时，先找子类，再找父类

图示

中度可信度描述已自动生成

* 1. **方法重写 Override**

继承的方法，在子类中重新定义重新编写

可以理解成“覆盖”、“改写”

* 1. **父类的构造方法**

新建子类对象时，会先新建父类对象，也会先执行父类的构造方法

* 默认执行父类的无参构造，默认隐含调用 super()

new Student()， 默认先执行父类的无参

new Student("","",0,"")， 默认先执行父类的无参

* 手动调用父类有参构造，super(参数)
  + 当父类没有无参构造，必须手动调用有参构造
  1. **super**

**super.xxxxx()**

* 方法重写时，调用父类中同一个方法的代码

**super(参数)**

* 调用父类构造方法
* 默认调用父类无参构造 super()
* 手动调用父类有参构造 super(参数)
* 必须是首行代码

1. **继承练习**

人类 Person

{

name,gender,age

toString()

}

学生 Student 继承 Person

{

school

}

员工 Employee 继承 Person

{

salary

}

项目：d0701\_继承

类：d0701.Test1

Person

Student

Employee

Person

**package** d0701;

**public** **class** Person /\*extends Object\*/ {

String name;

String gender;

**int** age;

**public** Person() {

**this**("无名", "不明", 16);

}

**public** Person(String name,String gender,**int** age) {

**this**.name = name;

**this**.gender = gender;

**this**.age = age;

}

**public** String toString() {

**return** name+", "+gender+", "+age;

}

}

Student

**package** d0701;

**public** **class** Student **extends** Person{

String school;

**public** Student() {

//super();//默认的隐含调用

}

**public** Student(String name,

String gender,

**int** age,

String school) {

//super();//默认的隐含调用

**super**(name,gender,age);

this.school = school;

}

@Override

**public** String toString() {

/\*

\* super.toString()

\* 目的是减少代码重复，

\* 一般在方法重写时，

\* 调用父类同一个方法的代码

\*/

**return** **super**.toString()+", "+school;

}

}

Employee

**package** d0701;

**public** **class** Employee **extends** Person{

**double** salary;

@Override

**public** String toString() {

**return** **super**.toString()+", "+salary;

}

}

Test1

**package** d0701;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* p = {

\* name:"张三",

\* gender:"男",

\* age:22

\* }

\*

\* s = {

\* name:"李四",

\* gender:"女",

\* age:21,

\*

\* school:"牛蹄筋大学"

\* }

\* e = {

\* name:"王五",

\* gender:"男",

\* age:20,

\*

\* salary: 9000

\* }

\*

\*/

Person p = **new** Person("张三","男",22);

Student s = **new** Student();

Employee e = **new** Employee();

s.name = "李四";

s.gender = "女";

s.age = 21;

s.school = "牛蹄筋大学";

e.name = "王五";

e.gender = "男";

e.age = 20;

e.salary = 9000;

//println()打印方法中，

//默认调用了对象的 toString() 方法

System.***out***.println(p.toString());

System.***out***.println(s.toString());

System.***out***.println(e.toString());

}

}

Test2

**package** d0701;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

//默认执行 Person()

Student s1 = **new** Student();

//默认执行 Person()

Student s2 = **new** Student("a","b",0,"c");

System.***out***.println(s1.toString());

System.***out***.println(s2.toString());

}

}

1. **二维点、三维点**

Point 类

{

x,

y,

toString(), //x,y

distance() //x,y

}

Point3D 类

{

z,

toString(),

distance()

}

项目：d0702\_二维点三维点

类：d0702.Test1

Point

Point3D

Point

**package** d0702;

**public** **class** Point {

**int** x;

**int** y;

**public** Point(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

**public** String toString() {

// "(3, 4)"

**return** "("+x+", "+y+")";

}

**public** **double** distance() {

**return** Math.*sqrt*(x\*x + y\*y);

}

}

Point3D

**package** d0702;

**public** **class** Point3D **extends** Point{

**int** z;

**public** Point3D(**int** x, **int** y, **int** z) {

//父类没有无参构造，必须手动调用有参构造

**super**(x, y);

**this**.z = z;

}

@Override

**public** String toString() {

**return** "("+x+", "+y+", "+z+")";

}

@Override

**public** **double** distance() {

**return** Math.*sqrt*(x\*x + y\*y + z\*z);

}

}

Test1

**package** d0702;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* p1 = {

\* x: 3,

\* y: 4

\* }

\*

\* p2 = {

\* x: 3,

\* y: 4,

\* z: 5

\* }

\*/

Point p1 = **new** Point(3, 4);

Point3D p2 = **new** Point3D(3, 4, 5);

System.***out***.println(p1.toString());

System.***out***.println(p1.distance());

System.***out***.println("--------------");

System.***out***.println(p2.toString());

System.***out***.println(p2.distance());

}

}

1. **电子宠物**

Pet 类

{

name,

full,

happy,

feed(),

play(),

punish()

}

Dog 类 继承 Pet

Cat 类 继承 Pet

项目：d0703\_电子宠物

类：d0703.Test1

Pet

Dog

Cat

Pet

**package** d0703;

**public** **class** Pet {

String name;

**int** full;

**int** happy;

**public** Pet(String name) {

**this**(name, 50 ,50);

}

**public** Pet(String name, **int** full, **int** happy) {

**this**.name = name;

**this**.full = full;

**this**.happy = happy;

}

**public** **void** feed() {

**if** (full == 100) {

System.***out***.println(name+"已经吃饱了");

**return**;

}

System.***out***.println("给"+name+"喂食");

full += 10;

System.***out***.println("饱食度："+full);

}

**public** **void** play() {

**if** (full == 0) {

System.***out***.println(name+"饿得玩不动了");

**return**;

}

System.***out***.println("陪"+name+"玩耍");

happy += 10;

full -= 10;

System.***out***.println("快乐度："+happy);

System.***out***.println("饱食度："+full);

}

**public** **void** punish() {

//不同的代码，就调用方法

System.***out***.println(

"打"+name+"的pp，"+name+"哭叫："+cry());

happy -= 10;

System.***out***.println("快乐度："+happy);

}

**public** String cry() {

//无意义代码，必须在子类中重写这个方法

**return** "此处有哭叫声~";

}

}

Dog

**package** d0703;

**public** **class** Dog **extends** Pet{

**public** Dog(String name) {

**super**(name);

}

**public** Dog(String name, **int** full, **int** happy) {

**super**(name,full,happy);

}

@Override

**public** String cry() {

**return** "汪~";

}

}

Cat

**package** d0703;

**public** **class** Cat **extends** Pet{

**public** Cat(String name) {

**super**(name);

}

**public** Cat(String name, **int** full, **int** happy) {

**super**(name,full,happy);

}

@Override

**public** String cry() {

**return** "喵~";

}

}

Test1

**package** d0703;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("1. 猫");

System.***out***.println("2. 狗");

System.***out***.print("选择：>");

**int** c = **new** Scanner(System.***in***).nextInt();

System.***out***.print("给宠物起个名：");

String n = **new** Scanner(System.***in***).nextLine();

Cat cat = **null**;

Dog dog = **null**;

**if** (c == 1) {

cat = **new** Cat(n);

*f*(cat);

} **else** {

dog = **new** Dog(n);

*f*(dog);

}

}

**private** **static** **void** f(Pet pet) {

System.***out***.println("按回车执行");

**while**(**true**) {

**new** Scanner(System.***in***).nextLine();

**int** r = **new** Random().nextInt(9);

**switch**(r) {

**case** 0: pet.feed(); **break**;

**case** 1: pet.play(); **break**;

**default**: pet.punish();

}

}

}

}

1. **冒泡排序**

Arrays.sort(数组)

基本类型数组，优化的快速排序

引用类型数组，优化的合并排序

项目：d0704\_冒泡排序

类：d0704.Test1

**package** d0704;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//获得一个随机乱序数组

**int**[] a = *suiJi*();

System.***out***.println(Arrays.*toString*(a));

System.***out***.println("----------------");

*sort*(a);//对a数组排序

System.***out***.println("----------------");

System.***out***.println(Arrays.*toString*(a));

}

**private** **static** **int**[] suiJi() {

//产生一个 5+ [0,6) 范围的整数存到变量 n

//新建 int[] 数组a，长度 n

//遍历数组，填入 100 内的随机整数

//返回数组a

**int** n = 5+ **new** Random().nextInt(6);

**int**[] a = **new** **int**[n];

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

a[i] = **new** Random().nextInt(100);

}

**return** a;

}

**private** **static** **void** sort(**int**[] a) {

/\*

\* j

\* [2, 11, 24, 51, 46, 95, 33, 45, 64, 72]

\* i

\*

\* j

\* [1, 5, 5, 33, 40, 48, 71, 50, 77]

\* i

\*

\*

\*

\* [1, 5, 5, 33, 40, 48, 50, 71, 77]

\* i

\*

\* j循环开始之前 flag = false，标记没有交换

\*

\* if(! flag) {

\*

\* }

\*/

//i循环从头到尾遍历

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

**boolean** flag = **false**;//没有交换

//j循环，把较小值向前交换，最小值交换到i位置

//j循环从尾部向前递减到 >i

**for** (**int** j = a.length-1; j > i; j--) {

//j和j-1位置的值比大小

**if** (a[j]<a[j-1]) {

**int** t = a[j];

a[j] = a[j-1];

a[j-1] = t;

flag = **true**;//有交换

}

}

//flag还是false，没有修改成true

//没有执行交换，数据位置都是正确的

**if** (! flag) {

**break**;

}

System.***out***.println(Arrays.*toString*(a));

}

}

}

1. **Object**

所有类的顶层父类，一个类如果不继承其他类，默认从 Object 继承

class A /\*extends Object\*/ {

}

Object 的方法

* **toString()**

获得一个对象的字符串表示

Object中的默认实现是：

"d0801.Point@a19c832c"

"类名@地址"

如果需要，可以在子类中重写 toString()

* **equals(Object obj)**

当前对象，与参数对象 obj 比较是否相等

a.equals(b)

Object中的默认实现是比较内存地址

this == obj

如果需要，在子类中，可以重写这个方法

1. **Object**

项目：d0801\_Object

类：d0801.Test1

Point

Point

**package** d0801;

**public** **class** Point /\*extends Object\*/ {

**int** x;

**int** y;

//无参构造、全参构造

**public** Point() {

}

**public** Point(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

@Override

**public** String toString() {

**return** "Point [x=" + x + ", y=" + y + "]";

}

/\*

\* a = {x:3, y:4}

\* b = {x:3, y:4}

\*

\* a.equals(b)

\*/

@Override

**public** **boolean** equals(Object obj) {

Point p = (Point) obj;

//当前对象的x,y，和参数对象p的x,y比较

**return** **this**.x == p.x && **this**.y == p.y;

}

}

Test1

**package** d0801;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

Point p1 = **new** Point(3, 4);

Point p2 = **new** Point(3, 4);

System.***out***.println(p1.toString());

System.***out***.println(p2.toString());

System.***out***.println(p1 == p2);

/\*

\* equals() 方法比较什么，

\* 要进入方法，看方法代码比较的是什么

\* Object中比较内存地址

\*/

System.***out***.println(p1.equals(p2));

}

}

1. **String**

String是封装 char[] 数组的对象

s = "abcd"

s = {

-value: ['a', 'b', 'c', 'd']

}

* 1. **字符串创建**

char[] a = {'a', 'b', 'c', 'd'};

String s = new String(a);

上面创建方式的简化语法

String s = "abcd";

* 1. **字符串的常量池**

第一次使用一个字符串字面值时，会在“字符串常量池”中新分配内存，再次使用相同字面值时，直接访问常量池中存在的对象，而不重复创建

String s1 = "abcd";

String s2 = "abcd";

1. **字符串**

项目：d0802\_字符串

类：d0802.Test1

**package** d0802;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

**char**[] a = {'a', 'b', 'c', 'd'};

String s1 = **new** String(a);//新分配内存

String s2 = "abcd";//在常量池新分配内存

String s3 = "abcd";//访问常量池存在的对象

System.***out***.println(s1);

System.***out***.println(s2);

System.***out***.println(s3);

System.***out***.println(s1 == s2);//false，地址不同

System.***out***.println(s2 == s3);//true，地址相同

/\*

\* equals()比较什么，

\* 进入equals()方法看里面的代码在比较什么

\*/

System.***out***.println(s1.equals(s2));

}

}

* 1. **字符串不可变**

String s1 = "aaa"

String s2 = "bbb"

String s3 = "ccc"

String s4 = s1 + s2 + s3 + s1 + s2 + s3 + s1 + s2 + s3 + s1 + s2 + s3 +

字符串连接，效率低，每次连接都新建字符串对象

1. **字符串连接效率**

Test2

**package** d0802;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

String s0 = "abcdefghijklmnopqrstuvwxyz";

String s = "";

//连接前，取系统当前时间的时间点毫秒值

//毫秒值：1970-1-1 0点开始的毫秒值

**long** t = System.*currentTimeMillis*();

**for** (**int** i = 0; i < 100000; i++) {

s += s0;

}

//求时间差

t = System.*currentTimeMillis*()-t;

System.***out***.println(t);

}

}

* 1. **字符串的常用方法**

**charAt(i)**

获取指定位置的字符

**length()**

字符串长度，字符的数量

**indexOf(子串)**

找第一个子串出现的起始位置，找不到返回 -1

**indexOf(子串, start)**

从执行位置向后查找

**lastIndexOf(子串)**

从后向前找

**substring(start)**

截取 start 到末尾

**substring(start, end)**

截取 [start, end) 范围

**trim()**

去除两端的空白字符

1. **二进制十六进制格式判断**

**package** d0402;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//死循环

outer:

**while**(**true**) {

System.***out***.println(

"输入0b开头二进制，或0x开头十六进制，q退出");

String s = **new** Scanner(System.***in***).nextLine();

//字符串比较是否相等，不能用 ==

//要用 equals() 方法

**if**(s.equals("q")) {

**break**;

}

//前缀不是0b，也不是0x

**if** (! (s.startsWith("0b") || s.startsWith("0x"))) {

System.***out***.println("前缀错误");

**continue**;

}

//字符串长度至少3

**if**(s.length() < 3) {

System.***out***.println("格式错误");

**continue**;

}

**if**(s.startsWith("0b")) {

// 0b10x0111101

**for** (**int** i = 2; i < s.length(); i++) {

**char** c = s.charAt(i);

**if** (! (c=='1' || c=='0')) {

System.***out***.println("二进制格式错误");

**continue** outer;

}

}

}

**if**(s.startsWith("0x")) {

// 0x9af9ce

**for** (**int** i = 2; i < s.length(); i++) {

**char** c = s.charAt(i);

**if** (! ((c>='0' && c<='9') || (c>='a'&&c<='f'))) {

System.***out***.println("十六进制格式错误");

**continue** outer;

}

}

}

//把s传递到 f() 方法，转成10进制整数

**int** r = *f*(s);

System.***out***.println("10进制："+r);

System.***out***.println("\n---------------------");

}

}

**private** **static** **int** f(String s) {

/\*

\* 参数字符串 s

\* "0B1101011"

\* "0xf9aac5e" --> "f9aac5e"

\* 如果是大写，先变成小写

\* s.toLowerCase()

\*

\* 获取0b或0x后面的数字部分

\* s.substring(2)

\*

\* 判断前缀是0b还是0x

\* s.startsWith("0b")

\*

\* 转成10进制

\* Integer.parseInt("1101011", 2)

\* Integer.parseInt("f9aac5e", 16)

\*/

//转成小写 "0B1101" --> "0b1101"

s = s.toLowerCase();

//截取0b、0x后面的数字部分 "0b1101"--> "1101"

String n = s.substring(2);

//准备一个保存结果的变量 r

**int** r = 0;

//判断前缀是0b还是0x

**if** (s.startsWith("0b")) {

r = Integer.*parseInt*(n, 2);//"1101"-->13

} **else** **if** (s.startsWith("0x")) {

r = Integer.*parseInt*(n, 16);//"d" -->13

}

**return** r;

}

}

1. **StringBuilder**

可变的字符序列

封装 char[] 数组，提供了一组方法，可以对内部封装的字符进行修改

常用来代替字符串，做高效率的字符串连接

append() 方法，追加字符内容

* 内部数组默认初始容量 16
* 放满后，容量翻倍+2

1. **StringBuilder**

项目：d0803\_StringBuilder

类：d0803.Test1

**package** d0803;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

String s0 = "abcdefghijklmnopqrstuvwxyz";

StringBuilder sb = **new** StringBuilder();

**long** t = System.*currentTimeMillis*();

**for** (**int** i = 0; i < 100000; i++) {

sb.append(s0);

}

t = System.*currentTimeMillis*() - t;

System.***out***.println(t);

}

}

* 1. **StringBuilder 的方法**

**append()**

追加

**delete(start, end)**

删除区间 [start, end)

**deleteCharAt(i)**

删除指定位置的单个字符

**insert(i, 内容)**

在指定位置插入

**setCharAt(i, 字符)**

替换指定位置的单个字符

**replace(start, end, 内容)**

替换指定范围的内容

...

* 1. **StringBuilder 和 StringBuffer**

StringBuffer 是一个旧版本的类，JDK1.0中就存在StringBuffer

StringBuilder 是一个新版本的类，JDK1.5中新添加的类

StringBuffer 线程安全

StringBuilder 线程不安全，效率高

1. **正则表达式 Regex**

Regex - Regular Expression

规则表达式

正确的字符串格式规则

一般用来判断用户输入的内容是否符合格式要求

例如，email，用户名，密码，生日，网址，手机号

百度“正则表达式大全”

|  |  |
| --- | --- |
| 正则表达式 | 匹配的字符串 |
| k | k |
| abc | abc |
| [abc] | a, b, c |
| [abc][123] | a1, a2, a3, b1, b2, b3, c1, c2, c3 |
| [a-z] | a, z, b, k |
| [a-zA-Z\_0-9] | a,A,9,\_ |
| [\u4e00-\u9fa5] | 中文范围 |
| [^a-zA-Z] | 排除英文  9,\*,中 |
| \d | 数字[0-9] |
| \D | 排除数字[^0-9] |
| \w | 单词字符[a-zA-Z\_0-9] |
| \W | 排除单词字符 |
| \s | 空白字符 |
| \S | 排除空白字符 |
| . | 任意字符 |
| [abc]? | ? 0或1个  a,b,,c |
| [abc]?[123] | a1,b3,a2,1,2,3 |
| [abc]\* | \* 0到多个  a,ab,,abccbbccaacbc |
| [abc]+ | + 1到多个  a,ab,abcbcbcaa |
| [abc]{3} | 3个  aaa,bbb,abc,ccb,caa |
| [abc]{3,5} | 3到5个  abc,abca,abcab |
| [abc]{3,} | 3到多个，至少3个  abc,abca,abcccbbacb |
| | |  |
| ^ | 匹配起始位置 |
| $ | 匹配结束位置 |

字符串的 matches() 方法，用来做匹配判断

if(s.matches(regex)) {

}

**split(正则)**

用匹配的子串，来拆分字符串

String s = "aaa,bbb,ccc";

String[] a = s.split(",");

**replaceAll(正则, 子串)**

替换所有匹配的子串

String s = "sdf754sdfsfd44767sdf7575";

s = s.replaceAll("\\d+", "\*\*\*");

1. **正则表达式**

项目：d0804\_正则表达式

类：d0804.Test1

**package** d0804;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("\n-----------------");

*f1*();

System.***out***.println("\n-----------------");

*f2*();

}

**private** **static** **void** f1() {

System.***out***.println("身份证号：");

String s = **new** Scanner(System.***in***).nextLine();

/\*

\* 123456789012345

\* 123456789012345678

\* 12345678901234567x

\* 12345678901234567X

\*

\* |

\* \d{15}|

\* \d{15}|\d{17}

\* \d{15}|\d{17}[\dxX] 正则表达式语法

\*

\* "\\d{15}|\\d{17}[\\dxX]"

\*/

String regex = "\\d{15}|\\d{17}[\\dxX]";

//判断字符串s能否匹配指定的正则表达式

**if** (s.matches(regex)) {

System.***out***.println("格式正确");

}**else** {

System.***out***.println("格式错误");

}

}

**private** **static** **void** f2() {

System.***out***.println("固定电话：");

String s = **new** Scanner(System.***in***).nextLine();

/\*

\* 1234567

\* 12345678

\* 010-1234567

\* 0101-12345678

\* (010)1234567

\* (0101)12345678

\*

\* (区号表达式)?\d{7,8}

\* (|)?\d{7,8}

\* (\d{3,4}-|)?\d{7,8}

\* (\d{3,4}-|())?\d{7,8}

\* (\d{3,4}-|\(\d{3,4}\))?\d{7,8}

\*/

String regex = "(\\d{3,4}-|\\(\\d{3,4}\\))?\\d{7,8}";

//判断字符串s能否匹配指定的正则表达式

**if** (s.matches(regex)) {

System.***out***.println("格式正确");

}**else** {

System.***out***.println("格式错误");

}

}

}

1. **基本类型的包装类**

有时需要把基本类型，当做一个对象来使用

void f(Object obj) {

}

f(new Integer(6))

|  |  |
| --- | --- |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

* 1. **数字父类 Number**

子类：

Byte,Short,Integer,Long

Float,Double

BigDecimal,BigInteger

取出基本类型值的方法：

byteValue()

shortValue()

intValue()

longValue()

floatValue()

doubleValue()

* 1. **Integer 类**

创建Integer对象

a = {

value: 6

}

Integer a = new Integer(6);

Integer a = Integer.valueOf(6);

Integer类中，存在 256 个Integer 的缓存对象，封装 -128 到 127

[{-128}, {-127}, {} ... {6} ... {127}]

如果指定范围内的值，访问缓存对象

如果指定范围外的值，直接新建对象

1. **基本类型的包装类**

项目：d0805\_基本类型的包装类

类：d0805.Test1

**package** d0805;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

Integer a = **new** Integer(6);

Integer b = Integer.*valueOf*(6);

Integer c = Integer.*valueOf*(6);

System.***out***.println(a == b);//false，地址不同

System.***out***.println(b == c);

System.***out***.println(a.equals(b));//

System.***out***.println(a.byteValue());

System.***out***.println(a.shortValue());

System.***out***.println(a.intValue());

System.***out***.println(a.longValue());

System.***out***.println(a.floatValue());

System.***out***.println(a.doubleValue());

}

}

* 1. **Integer 的方法**

字符串解析成 int

Integer.parseInt("255") 255

Integer.parseInt("11111111", 2) 255

Integer.parseInt("377", 8) 255

Integer.parseInt("ff", 16) 255

Byte.parseByte()

Short.parseShort()

Integer.parseInt()

Long.parseLong()

Float.parseFloat()

Double.parseDouble()

Boolean.parseBoolean()

进制转换

Integer.toBinaryString(255) "11111111"

Integer.toOctalString(255) "377"

Integer.toHexString(255) "ff"

* 1. **自动装箱、自动拆箱**
     1. **自动装箱**

基本类型值，自动包装成包装对象

Integer a = 6;

编译器编译成：

Integer a = Integer.valueOf(6);

* + 1. **自动拆箱**

int i = a;

编译器编译成：

int i = a.intValue();

a = a + 1;

编译器编译成：

a = Integer.valueOf(a.intValue() + 1);

* 自动拆箱，要当心 null 值

1. **双色球**

红球 33 选 6

蓝球 16 选 1

项目：d0806\_双色球

类：d0806.Test1

**package** d0806;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* r = [1,2,3,4,5,6,7,8,9,10....33]

\* b = [1,2,3,4,5...16]

\*/

//准备两个号码数组

**int**[] r = *zbsz*(33);

**int**[] b = *zbsz*(16);

System.***out***.println(Arrays.*toString*(r));

System.***out***.println(Arrays.*toString*(b));

**int**[] red = *xuanRed*(r);//选6个红球

**int** blue = *xuanBlue*(b);//选1个蓝球

System.***out***.println("红球：");

System.***out***.println(Arrays.*toString*(red));

System.***out***.println("蓝球：");

System.***out***.println(blue);

}

**private** **static** **int**[] xuanRed(**int**[] r) {

/\*

\* j

\* [12,15,3,1,17,10,7,8,9,6,11,4,13,14,2,16,5,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33]

\* i

\*

\* [12,15,3,1,17,10]

\*/

//i循环从0到<6，随机交换6个值到前面

**for** (**int** i = 0; i < 6; i++) {

//j随机定位 i+ [0, 33-i) --> [i, 33)

**int** j = i+ **new** Random().nextInt(33-i);

//i,j交换

**int** t = r[i];

r[i] = r[j];

r[j] = t;

}

//复制前6个位置，生成一个新的数组

**int**[] arr = Arrays.*copyOf*(r, 6);

**return** arr;

}

**private** **static** **int** xuanBlue(**int**[] b) {

**return** b[**new** Random().nextInt(16)];

}

**private** **static** **int**[] zbsz(**int** n) {

//新建int[]数组a，长度n

**int**[] a = **new** **int**[n];

//遍历，填入1到n

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

a[i] = i+1;

}

//返回数组a

**return** a;

}

}

1. **练习**

* 字符串
  + 输入 email 地址，提取名字部分

abc@def.com ---> abc

* + 输入字符串，判断是否对称

abcdedcba

# 日期

* java.util.Date

用毫秒值表示一个精确的时间点

* java.text.SimpleDateFormat

处理时间格式

Date对象格式化成字符串

日期格式的字符串解析成Date对象

* java.util.Calander

做日期运算

## Date

// 封装系统当前时间的毫秒值

Date d1 = new Date();

// 封装指定的毫秒值， 1970-1-1 0:0:1

Date d2 = new Date(1000L);

// 存取内部的毫秒值

long t = d1.getTime();

d1.setTime(0);

// Date的toString()方法，返回长格式日期字符串

print(d1.toString())

## SimpleDateFormat

SimpleDateFormat f = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String s1 = f.format(d1);

print(s1);

// 用户输入的日期字符串，解析成Date类型

Print("输入生日：");

s = new Scanner(System.in).nextLine();

d1 = new SimpleDateFormat("yyyy-MM-dd").parse(s);

t = System.currentTimeMillis() – d1.getTime();

age = t/1000/60/60/24/365

1. **面向对象4**

## 多态

一个子类型实例，可以被当做父类型来处理

子类实例

Point3D a =

{

x,y,z

}

Point a =

{

x,y,z

}

多态作用：一致的类型

void f(父类型 o1) {

}

所有子类型对象，都可被传递到该方法，被当做一致的父类型来处理

### 类型转换

#### 向上转型

子类型实例，转型成父类型

用父类型的引用变量，来引用子类实例

向上转型后，只能调用父类定义的通用成员，子类特有成员被隐藏

#### 向下转型

已经转成父类型的子类实例，转回成子类型

### instanceof

运行期类型识别

当多种子类型，都当做父类型来处理，要对某一种子类型进行特殊处理，可以先判断真实类型，再向下转型

对真实类型，及其父类型判断，都返回true

Line s = new Line();

s instanceof Line true

s instanceof Shape true

s instanceof Object true

* 1. **多态测试**

d0702\_二维点三维点

d0703\_电子宠物

项目：d0901\_图形形状

类：d0901.Test1

Shape

Line

Square

Circle

Shape 图形形状类

{

draw(),

clear()

}

Line 线段类

{

draw()

length()

}

Square 方形类

{

draw()

}

Circle圆形类

{

draw()

}

Shape

**package** d0901;

**public** **abstract** **class** Shape {

/\*

\* \*) 作为通用方法

\* \*) 子类必须实现

\*/

**public** **abstract** **void** draw();

**public** **void** clear() {

System.***out***.println("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

System.***out***.println("图形已被擦除");

System.***out***.println("---------------------------");

}

}

Line

**package** d0901;

**public** **class** Line **extends** Shape{

@Override

**public** **void** draw() {

System.***out***.println("一");

}

//线段自己的方法，与其他类无关

**public** **void** length() {

System.***out***.println("十几厘米长...");

}

}

Square

**package** d0901;

**public** **class** Square **extends** Shape{

@Override

**public** **void** draw() {

System.***out***.println("口");

}

}

Circle

**package** d0901;

**public** **class** Line **extends** Shape{

@Override

**public** **void** draw() {

System.***out***.println("一");

}

//线段自己的方法，与其他类无关

**public** **void** length() {

System.***out***.println("十几厘米长...");

}

}

Test1

**package** d0901;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* Shape s = null;

\*

\* s = Shape实例

\* s = Line实例

\* s = Square实例

\* s = Circle实例

\*/

System.***out***.println("按回车执行");

Shape s = **null**;

**while**(**true**) {

**new** Scanner(System.***in***).nextLine();

**int** r = **new** Random().nextInt(4);

**switch**(r) {

**case** 0:

System.***out***.println("Shape抽象类，不能创建实例");

**continue**;

**case** 1: s = **new** Line(); **break**;

**case** 2: s = **new** Square(); **break**;

**case** 3: s = **new** Circle(); **break**;

}

*f*(s);

}

}

/\*

\* Shape

\* |- Line， length()

\* |- Square

\* |- Circle

\*/

**private** **static** **void** f(Shape s) {

System.***out***.println("---当做Shape父类型处理----------------");

//获得真实类型的类名,

//模仿Object中toString()方法的代码

String n = s.getClass().getName();

System.***out***.println("真实类型："+n);

s.draw();

//当做父类型，只能调用父类定义的通用成员，

//子类特有成员被隐藏

//s.length();

//s引用的实例，真实类型是Line

**if** (s **instanceof** Line) {

System.***out***.println(

"向下转回成 Line 类型,再调用长度方法");

Line line = (Line) s;

line.length();

}

**new** Scanner(System.***in***).nextLine();

s.clear();

}

}

## 什么是面向对象

封装、继承、多态

* 封装
  + 类
  + 对象、实例
  + 引用
  + 构造方法
  + this
    - this.xxx
    - this(...)
  + 方法重载 Overload
* 继承
  + 作用：代码重用、复用
  + 单继承
  + 子类对象
    - 父类对象和子类对象绑定，整体作为一个对象
    - 调用成员，先找子类，再找父类
  + 方法重写 Override
  + super
    - super.xxx()
    - super()
* 多态
  + 作用：一致的类型
  + 向上转型
  + 向下转型
  + instanceof

## 抽象类

半成品类，没有写完的类

抽象方法，没有代码，只有方法定义

public abstract void f();

包含抽象方法的类，必须是抽象类

public abstract class A {

public abstract void f();

}

没有抽象方法的类，也可以被定义成抽象类

public abstract class A {

public void f() {

。。。。。

}

}

抽象类不能创建实例

抽象方法的作用：

* 作为通用方法，在父类中定义
* 要求子类，必须实现这个方法

图示, 示意图

描述已自动生成

* 1. **抽象类测试**

d0703\_电子宠物

Pet

d0901\_图形形状

Shape

1. **生成一个有序的数字数组**

按回车，随机生成数字，添加到数组，直接生成有序数组

[3,34,56,77,87]

项目：d0902\_随机生成有序数组

类：d0902.Test1

**package** d0902;

**import** java.math.BigDecimal;

**import** java.util.Arrays;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Test1 {

/\*

\* 二分法查找、折半查找

\* 在有序数组中查找一个值所在的位置下标，

\* 如果找不到，返回 -(插入点+1)

\*

\* 29

\* mid

\* [2,3,3,5,6,8,12,13,13,13,17,28,30,32] 14

\* lo

\* hi

\* \*) lo和hi定位低位和高位下标

\* \*) 中间位置是 mid = (lo+hi)/2;

\* \*) mid位置大，hi定位到mid-1

\* \*) mid位置小，lo定位到mid+1

\* \*) mid位置找到相等的，mid下标就是结果

\* \*) 找不到，-(lo+1)，为什么+1，防止lo是0

\*/

**public** **static** **void** main(String[] args) {

/\*

\* Number是抽象父类

\*

\* Number[]

\* []

\* [45]

\* [23 45 760.7619368204128]

\* Double Integer

\*/

System.***out***.println("按回车执行");

Number[] a = **new** Number[0];

**while**(**true**) {

**new** Scanner(System.***in***).nextLine();

Number n = **null**;

//随机产生整数，或者是浮点数

//一半机会产生整数 Integer

**if**(Math.*random*() < 0.5) {

// 682 自动装箱 Integer

// 再向上转型成父类型 Number

n = **new** Random().nextInt(1000);

} **else** {//一半机会产生浮点数 Double

// 917.72345546 自动装箱 Double

// 再向上转型成父类型 Number

**double** d = Math.*random*() \* 1000;

//舍入运算，保留1位小数

d = BigDecimal

.*valueOf*(d)

.setScale(1, BigDecimal.***ROUND\_HALF\_UP***)

.doubleValue();

n = d;

}

//System.out.println(n);

//二分法查找，在a数组中，查找n的下标位置

//index是正数，就是插入点

//index是负数（-(插入点+1)），用-index-1计算插入点

**int** index = *binarySearch*(a, n);

**if**(index<0) {

index = -index-1;

}

/\* 5

\* [2,3,3,5,6,8, ]

\* i

\* 5

\* [2,3,3,5,5,6,8]

\* i \*/

a = Arrays.*copyOf*(a, a.length+1);

//index是插入点，把插入点后面的数据，向后挪一位

System.*arraycopy*(a,index,a,index+1,a.length-index-1);

a[index] = n; //把数据放入插入点位置

System.***out***.println(Arrays.*toString*(a));

}

}

**private** **static** **int** binarySearch(

Number[] a, Number n) {

**int** lo = 0;

**int** hi = a.length-1;

**int** mid;

//lo和hi不能交叉，交叉就是找不到

**while**(lo<=hi) {

//中间位置

mid = (lo+hi)/2;

//mid位置的值和 n 比较

**if** (a[mid].doubleValue() > n.doubleValue()) {

hi = mid-1;

} **else** **if**(a[mid].doubleValue() < n.doubleValue()) {

lo = mid+1;

} **else** {

**return** mid;

}

}

**return** -(lo+1);

}

}

1. **BigDecimal，BigInteger**

BigDecimal精确的浮点数运算

BigInteger超大整数运算

把运算数，先都封装成 BigDecimal 对象，再使用 BigDecimal 提供的计算方法来计算

创建 BigDecimal 对象

BigDecimal bd = BigDecimal.valueOf(2);

方法

* add(BigDecimal bd)
* subtract(BigDecimal bd)
* multiply(BigDecimal bd)
* divide(BigDecimal bd)
* divide(BigDecimal bd,保留位数,舍入方式)
* setScale(保留位数,舍入方式)

1. **BigDecimal**

项目：d0903\_BigDecimal

类：d0903.Test1

**package** d0903;

**import** java.math.BigDecimal;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("输入两个浮点数");

**double** a = **new** Scanner(System.***in***).nextDouble();

**double** b = **new** Scanner(System.***in***).nextDouble();

System.***out***.println(a+b);

System.***out***.println(a-b);

System.***out***.println(a\*b);

System.***out***.println(a/b);

/\*

\* 2 4.35

\* 1.9 100

\*/

System.***out***.println("------------------------");

BigDecimal bd1 = BigDecimal.*valueOf*(a);

BigDecimal bd2 = BigDecimal.*valueOf*(b);

BigDecimal bd3;//用来保存计算结果

bd3 = bd1.add(bd2);

System.***out***.println(bd3.doubleValue());

bd3 = bd1.subtract(bd2);

System.***out***.println(bd3.doubleValue());

bd3 = bd1.multiply(bd2);

System.***out***.println(bd3.doubleValue());

bd3 = bd1.divide(

bd2, 50, BigDecimal.***ROUND\_HALF\_UP***);

System.***out***.println(bd3.doubleValue());

System.***out***.println(bd3.toString());

}

}

阶乘

**package** d0306;

**import** java.math.BigInteger;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("输入整数，求阶乘：");

**int** n = **new** Scanner(System.***in***).nextInt();

//把n变量的值取出来，传递到f()方法

String r = *f*(n);

System.***out***.println(r);

}

/\*

\* r = n

\*

\* i

\* n-1, r=r\*i

\* 3, r=r\*i

\* 2, r=r\*i

\* 1, r=r\*i

\*/

**private** **static** String f(**int** n) {

/\*//如果参数n的值是0，直接返回1

if(n == 0) {

return 1;

}

long r = n;

//i循环从n-1到1

for (int i = n-1; i >= 1; i--) {

r=r\*i;

}

//向调用位置返回计算结果

return r;\*/

**if**(n == 0) {

**return** "1";

}

BigInteger r = BigInteger.*valueOf*(n);

**for** (**int** i = n-1; i >= 1; i--) {

r = r.multiply(BigInteger.*valueOf*(i));

}

**return** r.toString();

}

}

1. **位运算符**

+-\*/% == != > >= < <= && || ! ++ -- ? : = += () instanceof

|  |  |
| --- | --- |
| & | 位与，两位都是1，结果才是1 |
| | | 位或，任意一位是1，结果就是1 |
| ^ | 异或，不同是1  对同一个值异或两次得到原值 |
| ~ | 求反，1变0,0变1 |
| >> | 带符号右移位  符号位是1，左边补1 |
| >>> | 不带符号右移位  左边都补0 |
| << | 左移位 |

1. **位运算**

项目：d0904\_位运算

类：d0904.Test1

**package** d0904;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* 746345

\* 00101010 11111001 10111010 10111011

\* <<--------24-----------

\* ---------24---------->>>

\* 00000000 00000000 00000000 10111011

\*/

System.***out***.print("输入整数：");

**int** n = **new** Scanner(System.***in***).nextInt();

n = n<<24>>>24;

System.***out***.println(n);

/\*

\* 00101010 11111001 10111010 10111011

\* 00000000 00000000 00000000 11111111 &

\* --------------------------------------

\* 00000000 00000000 00000000 10111011

\*/

System.***out***.print("输入整数：");

n = **new** Scanner(System.***in***).nextInt();

n = n & 0x000000ff;

System.***out***.println(n);

}

}

int的四个字节值，拆分成4个byte值

00101010 11111001 10111010 10111011

00101010

11111001

10111010

10111011

Test2

**package** d0904;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

System.***out***.print("整数：");

**int** n = **new** Scanner(System.***in***).nextInt();

**byte**[] a = *f1*(n);

System.***out***.println(Arrays.*toString*(a));

**int** r = *f2*(a);

System.***out***.println(r);

}

**private** **static** **int** f2(**byte**[] a) {

/\*

\* 00101010

\* 11111001

\* 10111010

\* 10111011

\*

\* r = 00101010 11111001 10111010 00000000

\*

\* 每一个字节值先和0x000000ff求位与，

\* 再左移24,16,8,0位

\* 再和r求位或，结果保存到r

\*/

**int** r = 0;

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

r = r | ((a[i]&0x000000ff)<<(24-8\*i));

}

**return** r;

}

**private** **static** **byte**[] f1(**int** n) {

/\*

\* 00101010 11111001 10111010 10111011

\*

\* n右移24,16,8,0，再强转byte

\* i 0 1 2 3

\*/

**byte**[] a = **new** **byte**[4];

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

a[i] = (**byte**) (n>>(24-8\*i));

}

**return** a;

}

}

作业

**package** d0905;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("email:");

String e = **new** Scanner(System.***in***).nextLine();

String name = *getName*(e);

System.***out***.println(name);

System.***out***.println("对称：");

String s = **new** Scanner(System.***in***).nextLine();

**if** (*duiChen*(s)) {

System.***out***.println("对称");

} **else** {

System.***out***.println("不对称");

}

}

**private** **static** String getName(String e) {

/\*

\* "abc@def.com"

\* |

\*/

**int** index = e.indexOf("@");

**if** (index == -1) {

**return** "格式错误";

}

**return** e.substring(0, index);

}

**private** **static** **boolean** duiChen(String s) {

/\*

\* j

\* abcdedcba

\* i

\*/

**for** (**int** i = 0,j=s.length()-1; i<j; i++,j--) {

//i和j位置字符不相等

**if**(s.charAt(i) != s.charAt(j)) {

**return** **false**;

}

}

**return** **true**;

}

}

1. **练习**

* 有序数组的并归
  + 两个有序数组，合并成一个有序数组

[1,2,2,4,5]

j

[2,2,4,5,7,7,8,9]

k

[1 2 2 2 2 4 4 5 5 ]

i

* + j和k的较小值放入i位置，对应下标递增
  + j越界，复制k后面的值
  + k越界，复制j后面的值

1. **面向对象5**

## final

可以修饰常量、方法、类

### final常量

值不可变

final int a = 6;

a = 7;//错

final Point a = new Point(3,4);

a.x = 30;//对

a.y = 40;//对

a = new Point(1,6);//错

a = null;//错

### final方法

不能在子类中重写

### final类

不能被继承，没有子类

java.lang.System

java.lang.String

java.lang.Integer

## static

静态

静态成员属于类,而不属于实例

* 静态成员
  + 用“类”来调用静态成员

Soldier.count

Math.random()

Integer.parseInt()

* 实例成员
  + 用“实例”来调用实例成员

s1.id

s2.blood

s1.attack()

s2.go()

### 静态的使用场景

static 的使用原则：能不用就不用

静态是“非面向对象”的语法

* 共享的数据

Soldier.count

* 工具方法

Math.random()

Arrays.toString()

String.valueOf()

### 静态方法中，不能直接调用实例的成员

class A {

public static void main(String[] args) {

f();//静态调静态

}

static void f() {

g();//错，静态不能直接调用非静态

A a = new A();

a.g();//只能用实例调用

}

void g(){

}

}

### 静态初始化块

class A {

static {

静态初始化块

类被加载时，只执行一次

}

}

一般执行初始化工作，例如，加载一些资源文件

文本, 应用程序

中度可信度描述已自动生成

1. 士兵的数量

d0601\_士兵

复制

d1002\_士兵

Soldier

**package** d1002;

**import** java.util.Random;

/\*

\* 封装

\* 把士兵相关的数据、运算方法，封装成一个“类”

\*/

**public** **class** Soldier {

//成员变量、属性变量

**int** id; //默认值0

**int** blood = 100;

//静态变量，士兵的数量

//在方法区，类的内存空间中，只保存一份

//可以在所有士兵实例之间共享的数据

**static** **int** *count*;

**public** Soldier() {

//静态属于类，一般用类名调用

Soldier.*count*++;

}

//成员方法

**public** **void** go() {

System.***out***.println(**this**.id+"号士兵前进");

}

**public** **void** attack() {

**if**(**this**.blood == 0) {

System.***out***.println("这是"+id+"号士兵的尸体");

**return**;

}

System.***out***.println(**this**.id+"号士兵进攻");

//模拟受到伤害，产生一个随机的减血量

**int** d = **new** Random().nextInt(10);

//如果减血量比当前血量大

**if**(d > **this**.blood) {

d = **this**.blood;//有多少减多少

}

**this**.blood -= d;

System.***out***.println("血量："+**this**.blood);

**if**(**this**.blood == 0) {

System.***out***.println(id+"号士兵阵亡");

Soldier.*count*--;

}

}

}

Test2

**package** d1002;

**import** java.util.Scanner;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

/\*

\* 新建数组，并在数组中新建 Soldier 实例

\* [{}, {}, {}]

\*

\* 当有存活士兵，士兵数量!=0

\* 让士兵进攻

\*/

System.***out***.print("士兵数量：");

**int** n = **new** Scanner(System.***in***).nextInt();

Soldier[] a = **new** Soldier[n];

//新建n个士兵实例存入数组

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

a[i] = **new** Soldier();

a[i].id = i+1;

}

System.***out***.println("按回车执行");

//当有存活的士兵

**while**(Soldier.*count* != 0) {

**new** Scanner(System.***in***).nextLine();

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

**if** (a[i].blood != 0) {//活着才执行进攻

a[i].attack();

}

}

System.***out***.println("存活士兵数量："+Soldier.*count*);

System.***out***.println("\n-----------------------");

}

}

}

## 访问控制符

用来控制一个类、成员的访问范围

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 类 | 包 | 子类 | 任意 |
| public |  |  |  |  |
| protected |  |  |  |  |
| [default] |  |  |  |  |
| private |  |  |  |  |

使用原则：尽量使用小范围

public 是与其他开发人员的一个契约，约定公开的东西会尽量保持稳定不变

private 私有隐藏，便于修改

1. **私有**

项目：d1003\_学生

类：d1003.Test1

Student

Student

**package** d1003;

**public** **class** Student {

//静态初始化块

**static** {

System.***out***.println(

"静态块 - Student类被加载时执行");

}

//成员变量一般都用private隐藏

//便于维护修改

**private** **int** id;

**private** String name;

**private** String gender;

**private** **int** age;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getGender() {

**return** gender;

}

**public** **void** setGender(String gender) {

**this**.gender = gender;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + ", gender=" + gender + ", age=" + age + "]";

}

}

Test1

**package** d1003;

**public** **class** Test1 {

**static** {

System.***out***.println("Test1 - 静态块");

}

**public** **static** **void** main(String[] args) {

System.***out***.println("main");

Student s = **new** Student();

Student s2 = **new** Student();

Student s3 = **new** Student();

Student s4 = **new** Student();

//s.id = 9527;

s.setId(9527);

s.setName("唐伯虎");

s.setGender("男");

s.setAge(19);

System.***out***.println(s.getId());

System.***out***.println(s.getName());

System.***out***.println(s.getGender());

System.***out***.println(s.getAge());

}

}

## 常量

static final int MAX\_VALUE=100;

static final 两个关键字定义常量

final - 不可变

static - 节省内存

命名习惯，全大写，单词之间用下划线连接

## 对象的创建过程

class A {

int v1 = 1;

static int v2 = 2;

static {...}

A() {...}

}

class B extends A {

int v3 = 3;

static int v4 = 4;

static {...}

B() {...}

}

new B();

**类加载**

1. 加载父类，为父类静态变量分配内存
2. 加载子类，为子类静态变量分配内存
3. 执行父类静态变量的赋值运算，和静态初始化块
4. 执行子类静态变量的赋值运算，和静态初始化块

**新建实例**

1. 新建父类实例，为父类的实例变量分配内存
2. 新建子类实例，为子类的实例变量分配内存
3. 执行父类的实例变量赋值运算
4. 执行父类的构造方法
5. 执行子类的实例变量赋值运算
6. 执行子类的构造方法
7. 对象创建过程

项目：d1004\_对象创建过程

类：d1004.Test1

**package** d1004;

// 公共类类名，必须是文件名

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

**new** B();

System.***out***.println("-----------");

**new** B();

}

}

**class** A {

**int** v1 = 1;

**static** **int** *v2* = 2;

**static** {

System.***out***.println("A静态块");

}

A() {

System.***out***.println("A构造");

}

}

**class** B **extends** A {

**int** v3 = 3;

**static** **int** *v4* = 4;

**static** {

System.***out***.println("B静态块");

}

B() {

System.***out***.println("B构造");

}

}

1. **集合**

用来存放一组数据的数据结构

**数组的缺点：**

* 长度固定
* 访问方式单一，只能下标访问
* 前面增删数据，操作繁琐，效率低

1. **ArrayList**

数组列表，封装了一个数组，封装了数组操作代码，提供了几个更简便的方法

* 内部数组默认的初始容量 10
* 放满后，1.5倍增长

## 方法

**add(数据)**

添加数据

**get(int i)**

访问指定下标数据

**remove(int i)**

移除指定位置数据，返回被移除的数据

**remove(数据)**

找到第一个相等数据，移除，

找到并移除，返回 true

找不到，返回 false

**size()**

元素的数量

**iterator()**

辅助新建一个迭代器对象

1. **ArrayList 测试**

项目：d1005\_ArrayList

类：d1005.Test1

**package** d1005;

**import** java.util.ArrayList;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* <> 泛型，对集合中存放的数据类型进行限制

\* 泛型和集合工具，不支持基本类型

\*

\* list = {

\* elementData=[null,null,null,null,null,null,null,null,null,null]

\* }

\*/

ArrayList<String> list = **new** ArrayList<>();

System.***out***.println(list);

list.add("aaa");

list.add("nnn");

list.add("ttt");

list.add("jjj");

list.add("kkk");

list.add("qqq");

list.add("ccc");

list.add("aaa");

list.add("aaa");

list.add("aaa");

list.add("hhh");

list.add("yyy");

System.***out***.println(list.size());

System.***out***.println(list);

System.***out***.println(list.get(0));

System.***out***.println(list.get(list.size()-1));

System.***out***.println(list.remove(5));

System.***out***.println(list);

System.***out***.println(list.remove("aaa"));

System.***out***.println(list);

**for** (**int** i = 0; i < list.size(); i++) {

System.***out***.println(list.get(i));

}

}

}

**package** d1006;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

**int**[] a = *suiJi*();

**int**[] b = *suiJi*();

Arrays.*sort*(a);

Arrays.*sort*(b);

System.***out***.println(Arrays.*toString*(a));

System.***out***.println(Arrays.*toString*(b));

**int**[] c = *heBing*(a, b);

System.***out***.println(Arrays.*toString*(c));

}

**private** **static** **int**[] heBing(**int**[] a, **int**[] b) {

/\*

a=[11, 12, 13, 14, 15, 16]

j

b=[12, 15, 16, 59, 61, 85, 91, 94]

k

c=[11, 12, 12, 13, 14, 15, 15, 16, 16 ]

i

\*/

//新建数组，长度是两个数组长度和

**int**[] c = **new** **int**[a.length+b.length];

**for** (**int** i = 0,j=0,k=0; i < c.length; i++) {

**if**(j>=a.length) {

System.*arraycopy*(b,k,c,i,b.length-k);

**break**;

}

**if**(k>=b.length) {

System.*arraycopy*(a,j,c,i,a.length-j);

**break**;

}

**if** (a[j]<b[k]) {

c[i] = a[j++];

} **else** {

c[i] = b[k++];

}

}

**return** c;

}

**private** **static** **int**[] suiJi() {

**int** n = 5+ **new** Random().nextInt(6);

**int**[] a = **new** **int**[n];

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

a[i] = **new** Random().nextInt(100);

}

**return** a;

}

}

* 1. **ArrayList效率**

访问任意位置，效率高

增删数据，效率可能降低

1. **ArrayList测试**

项目：d1101\_ArrayList

类：d1101.Test1

**package** d1101;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

//插入随机整数，生成有序数字列表

/\*

\* ArrayList 代替数组

\* \*）长度1.5倍自动增长

\* \*）增删数据，数据复制移位，自动进行

\*/

ArrayList<Integer> list = **new** ArrayList<>();

System.***out***.println("回车执行，按q退出");

**while**(**true**) {

String s = **new** Scanner(System.***in***).nextLine();

**if**(s.equals("q")) {

**break**;

}

**int** r = **new** Random().nextInt(1000);

//Collections是集合的工具类

//二分法查找，正数是插入点，负数是-(插入点+1)

**int** index = Collections.*binarySearch*(list, r);

**if**(index<0) {// -(插入点+1)

index = -index-1;//得到插入点

}

list.add(index, r);//向index位置插入数据

System.***out***.println(list);

}

//迭代器遍历集合

//Iterator - 迭代器

//新建迭代器，必须用集合的方法辅助创建

Iterator<Integer> it = list.iterator();

//当还有数据

**while**(it.hasNext()) {

//取下一个

Integer n = it.next();

System.***out***.println(n);

}

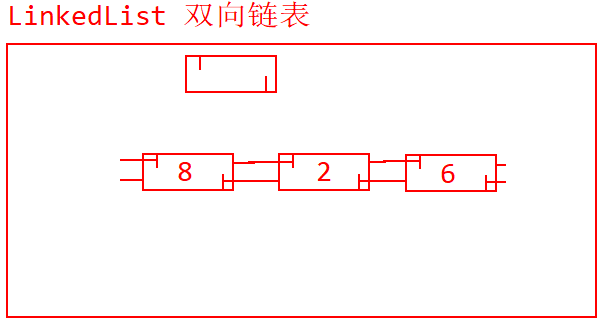
//背着写迭代遍历

}

}

1. **LinkedList**

双向链表



a={

value=8,

prev=c,

next=b

}

b={

value=2,

prev=a,

next=c

}

c={

value=2,

prev=b,

next=a

}

* 1. **LinkedList 效率**

两端效率高

* 1. **方法**

和ArrayList具有相同的方法

LinkedList 两端操作数据的方法

addFirst(数据)

addLast(数据)

getFirst()

getLast()

removeFirst()

removeLast()

* 1. **队列操作 Queue，FIFO**

offer() addLast()

peek() getLast()

poll() removeFirst()

* 1. **栈操作 Stack，LIFO**

push() addFirst()

pop() removeFirst()

1. **LinkedList测试**

项目：d1102\_LinkedList

类：d1102.Test1

**package** d1102;

**import** java.util.Iterator;

**import** java.util.LinkedList;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

/\*

\* add(), get(), remove(), size()

\*

\* 添加 1000 万数据量

\* [{1}, {1}, {1}, {1} ...]

\*/

LinkedList<Integer> list = **new** LinkedList<>();

**long** t;

t = System.*currentTimeMillis*();

**for** (**int** i = 0; i < 100000; i++) {

list.add(1);//添加1000万数据

}

System.***out***.println(System.*currentTimeMillis*()-t);

System.***out***.println("\n-----------------------");

t = System.*currentTimeMillis*();

//访问头部和尾部

System.***out***.println(list.getFirst());

System.***out***.println(list.getLast());

System.***out***.println(System.*currentTimeMillis*()-t);

System.***out***.println("\n-----------------------");

t = System.*currentTimeMillis*();

System.***out***.println(list.get(50000));//中间

System.***out***.println(System.*currentTimeMillis*()-t);

System.***out***.println("\n--下标遍历---------------");

t = System.*currentTimeMillis*();

**for** (**int** i = 0; i < list.size(); i++) {

list.get(i);

}

System.***out***.println(System.*currentTimeMillis*()-t);

System.***out***.println("\n--迭代器遍历---------------");

t = System.*currentTimeMillis*();

Iterator<Integer> it = list.iterator();

**while**(it.hasNext()) {

it.next();

}

System.***out***.println(System.*currentTimeMillis*()-t);

}

}

1. **丑数**

被 2,3,5 整除的数

1也是丑数

2326520/2/2/2/3/3/5/5/5

求第 n 个丑数

项目：d1103\_丑数

类：d1103.Test1

**package** d1103;

**import** java.util.ArrayList;

**import** java.util.LinkedList;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.print("求第几个丑数：");

**int** n = **new** Scanner(System.***in***).nextInt();

**long** r = *f*(n);

System.***out***.println(r);

}

//低效算法

**private** **static** **long** g(**int** n) {

**if**(n == 1) {

**return** 1;

}

**int** count = 1;

**for**(**long** i=2; ;i++){

//i是否是丑数

**long** j = i;

//当j能被2整除，除2

**while**(j%2==0) j /= 2;

**while**(j%3==0) j /= 3;

**while**(j%5==0) j /= 5;

//结果j是不是1

**if**(j==1) {

count++;

**if**(count == n) {

**return** i;

}

}

}

}

**private** **static** **long** f(**int** n) {

/\*

\* 10 12 16

\* ------------------2

\* 9 12 15 18 24

\* ------------------3

\* 10 15 20 25 30 40

\* ------------------5

\*

\* r = 1 2 3 4 5 6 8

\*

\* \*)先创建三个集合

\* \*)初始分别放入2,3,5

\* \*)从头部移除最小值

\* \*)最小值分别乘2,3,5放入三个集合

\*/

**if**(n == 1) {

**return** 1;

}

LinkedList<Long> list2 = **new** LinkedList<>();

LinkedList<Long> list3 = **new** LinkedList<>();

LinkedList<Long> list5 = **new** LinkedList<>();

list2.add(2L);

list3.add(3L);

list5.add(5L);

**long** r = 0;

//从第2个开始，求到第n个

**for** (**int** i = 2; i <= n; i++) {

//从头部移除最小值

**long** a = list2.getFirst();

**long** b = list3.getFirst();

**long** c = list5.getFirst();

r = Math.*min*(a, Math.*min*(b, c));

**if**(r == a) list2.removeFirst();

**if**(r == b) list3.removeFirst();

**if**(r == c) list5.removeFirst();

//\*2,3,5放入集合

list2.add(r \* 2);

list3.add(r \* 3);

list5.add(r \* 5);

}

**return** r;

}

}

1. **HashMap**

哈希表、散列表

存放键值对数据

|  |  |
| --- | --- |
| key | value |
| 9527 | 唐伯虎 |
| 9528 | 华夫人 |
| 9529 | 祝枝山 |
| 9530 | 旺财 |
| 9531 | 小强 |
| 9532 | 如花 |

用键，可以快速定位数据，来提取键对应的值

键：

* 不重复
* 无序

方法：

**put(key, value)**

放入键值对数据，重复的键，新值覆盖旧值

**get(key)**

获得键对应的值，键不存在，得到null

**remove(key)**

移除键值对数据，返回被移除的值

**size()**

键值对的数量

**keySet()**

的到包含所有键的Set类型集合

**values()**

得到包含所有值的Collection类型集合

**entrySet()**

得到包含所有Entry实例的Set集合

1. **HashMap测试**

项目：d1104\_HashMap

类：d1104.Test1

**package** d1104;

**import** java.util.HashMap;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

HashMap<Integer, String> map = **new** HashMap<>();

map.put(99527, "唐伯虎");

map.put(99528, "华夫人");

map.put(99529, "祝枝山");

map.put(99530, "旺财");

map.put(99531, "小强");

map.put(99532, "石榴姐");

map.put(99533, "秋香");

map.put(99533, "如花");//新值替换旧值

map.put(99534, **null**);

map.put(**null**, "---");

System.***out***.println(map.size());

System.***out***.println(map);

System.***out***.println(map.get(9527));

System.***out***.println(map.get(9999));

System.***out***.println(map.remove(9531));

System.***out***.println(map);

}

}

1. **字符串的字符统计**

"abacdedac"

i

key value

a 2

b 1

c 1

Test2

**package** d1104;

**import** java.util.HashMap;

**import** java.util.Scanner;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

System.***out***.println("输入字符串：");

String s = **new** Scanner(System.***in***).nextLine();

//统计方法

HashMap<Character, Integer> map = *tongJi*(s);

System.***out***.println(map);

}

**private** **static** HashMap<Character, Integer> tongJi(String s) {

/\*

\* \*)先新建 HashMap 对象 map

\* \*)i循环遍历字符串s

\* \*)从s取出i位置的一个字符 c

\* \*)从 map 取字符c对应的计数值 Integer count

\* \*)count是null

\* \*)放入c -- 1

\* \*)否则

\* \*)放入c -- count+1

\*

\* \*)返回 map

\*/

HashMap<Character, Integer> map = **new** HashMap<>();

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

**char** c = s.charAt(i);

Integer count = map.get(c);

**if**(count == **null**) {

map.put(c, 1);

} **else** {

map.put(c, count+1);

}

}

**return** map;

}

}

# 泛型

## 什么是泛型

是一种类型参数

## 使用示例

定义一个Holder类，用来包含、封装、持有一个数据

## <?>

使用带类型参数的类时，使用<?>语法，可以限制向对象中放入数据

<? extends Number>

# 枚举

类型安全的一组固定选项

**不使用枚举：**

interface Weapon {

int TYPE\_COLD = 0;

int TYPE\_HOT = 1;

int TYPE\_NUCLEAR = 2;

}

void f(int type) {

}

f(Weapon.TYPE\_COLD)

f(1)

f(2)

f(22)

**使用枚举，代替整数代码选项：**

定义枚举：

enum WeaponType {

// public static final WeaponType COLD = new WeaponType();

COLD, HOT, NUCLEAR

}

void f(WeaponType type) {

}

f(WeaponType.COLD);

f(WeaponType.HOT);

f(WeaponType.NUCLEAR);

# 注解

为编译器、其他开发工具或其他java程序来提供代码或对象的一些额外信息

## @Override注解

通知编译器在编译代码时，帮助检查方法重新的语法是否正确。

编译完成后，该注解的信息会被丢弃，在class文件中，不存在该注解的信息

## 自定义注解

自定义注解，要自己写处理代码来处理注解，才能让注解发挥一定的功能

1. 自定义一个 @Test 注解
2. 写一个工具，可以自动运行添加了 @Test 注解的方法
3. **异常**

封装错误信息的对象

* 1. **错误信息：**
* 类型
* 提示消息
* 行号
  1. **异常的继承结构**

Throwable

|- Error 系统级错误

|- Exception 可修复的错误

|- 其他Exception

|- RuntimeException

|- NullPointerException

|- ArrayIndexOutOfBoundsException

|- NumberFormatException

|- ArithmeticException

|- ClassNotFoundException

|- ...

* 1. **异常捕获 – try-catch-finally**

try {

} catch(AException e) {

} catch(BException e) {

} catch(父类型Exception e) {

} finally {

不管出不出错，都会执行

}

try {

} catch(AException e) {

}

try {

} finally {

}

1. **异常**

项目：d1105\_异常

类：d1105.Test1

**package** d1105;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

**while**(**true**) {

**try** {

*f*();

**break**;

} **catch** (ArrayIndexOutOfBoundsException e) {

System.***out***.println("输入两个！");

e.printStackTrace();

} **catch** (ArithmeticException e) {

System.***out***.println("不能除0！");

} **catch** (Exception e) {

System.***out***.println("出错，请重试");

} **finally** {

System.***out***.println("\n------------------------");

}

}

}

**private** **static** **void** f() {

/\*

\* "5235,45" --> ["5235", "45"]

\* 0 1

\*

\* 5235 / 45

\*/

System.***out***.println("输入逗号隔开的两个整数");

String s = **new** Scanner(System.***in***).nextLine();

String[] a = s.split(",");

**int** n1 = Integer.*parseInt*(a[0]);

**int** n2 = Integer.*parseInt*(a[1]);

**int** r = n1 / n2;

System.***out***.println(r);

/\*

\* 3453,33 345 abc,def 345,0

\*/

}

}

* 1. **throws**

在方法上，设置异常的抛出管道

形状

描述已自动生成

void f() throws X,Y,Z {

}

* 1. **处理异常的两种方式**
* throws
* try-catch

有异常时，两种处理方式，必须二选一，

如果不添加 throws，也不捕获异常，编译失败

* 1. **RuntimeException**

存在默认的抛出管道，RuntimeException和它的子类型异常，都可以从默认管道抛出

1. **异常**

项目：d1201\_异常

类：d1201.Test1

**package** d1201;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Scanner;

**public** **class** Test1 {

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

/\*

\* 输入生日(yyyy-MM-dd)：1999-9-12

\*

\* "1999-9-12" ---解析---> Date实例

\*

\* java.util.Date 封装一个毫秒值

\* java.text.SimpleDateFormat

\* 日期格式处理工具，

\* 把Date格式化成字符串

\* 把日期字符串，解析成Date对象

\*

\* 您已经生存了 xxx 天

\*/

System.***out***.print("生日(yyyy-MM-dd)：");

String s = **new** Scanner(System.***in***).nextLine();

//日期格式处理工具对象

SimpleDateFormat sdf =

**new** SimpleDateFormat("yyyy-MM-d");

Date d = sdf.parse(s);//把日期字符串解析成Date实例

// d.getTime()，访问它内部包含的毫秒值

**long** t = System.*currentTimeMillis*()-d.getTime();

t = t/1000/60/60/24;

System.***out***.println("您已经生存了"+t+"天");

}

}

* 1. **throw**

throw 手动抛出异常，执行异常的抛出动作

类比成 return

当程序出现逻辑错误，不自动创建并抛出异常，可以手动判断逻辑错误，手动创建异常对象并抛出

if(...) {

AException e = new AException();

throw e;

}

1. **throw**

Test2

**package** d1201;

**import** java.util.Scanner;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

*f*();

}

**private** **static** **void** f() {

/\*

\* 输入两个浮点数：

\* 234.23

\* 0

\*

\* 234.23 / 0 Infinity

\*/

System.***out***.println("输入两个浮点数：");

**double** a = **new** Scanner(System.***in***).nextDouble();

**double** b = **new** Scanner(System.***in***).nextDouble();

**try** {

**double** c = *divide*(a, b);

System.***out***.println(c);

} **catch** (ArithmeticException e) {

System.***out***.println("不能除零，是我们的错，请鞭笞我们吧！");

e.printStackTrace();

}

}

**private** **static** **double** divide(**double** a, **double** b) {

**if** (b == 0) {

ArithmeticException e =

**new** ArithmeticException("/ by zero");

**throw** e;

}

**return** a/b;

}

}

* 1. **异常包装**

捕获的异常对象，包装成另一种类型，再抛出

try {

} catch(AException e) {

BException e2 = new BException(e);

throw e2;

}

使用场景：

* 不能抛出的异常，包装成能抛出的异常再抛
* 多种类型异常，简化成一种类型

1. **异常包装**

Date

**package** d1201;

**import** java.text.SimpleDateFormat;

**public** **class** Date **extends** java.util.Date {

@Override

**public** String toString() {

// "yyyy-MM-dd HH:mm:ss"

SimpleDateFormat f =

**new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String s = f.format(**this**);

**return** s;

}

}

Test3

**package** d1201;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

*f*();

}

**private** **static** **void** f() {

Date d = **new** Date();

String s = d.toString();

System.***out***.println(s);

/\*

\* 在 d:\ 根目录，创建文件

\* “2019-04-17 11:09:53.txt”

\*/

s = s.replaceAll(":", ".");

String path = "d:\\"+s+".txt";

File f = **new** File(path);

//不想加throws

**try** {

f.createNewFile();

} **catch** (IOException e) {

//也不想在此处修复处理异常，

//希望把异常再抛到调用位置

//包装成能抛出的异常再抛

**throw** **new** RuntimeException(e);

}

}

}

* 1. **自定义异常**

现有的异常类型，不能表示所有业务中的错误情况

A到B转账失败

ZhuanZhangShiBaiException

用户名错误

UsernameNotFoundException

密码错误

WrongPasswordException

自定义异常

* 起一个合适的类名
* 选择一个合适的父类
* 添加合适的构造方法

1. **自定义异常**

UsernameNotFoundException

**package** d1201;

**public** **class** UsernameNotFoundException **extends** Exception{

**public** UsernameNotFoundException() {

**super**();

}

**public** UsernameNotFoundException(String message, Throwable cause, **boolean** enableSuppression,

**boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

}

**public** UsernameNotFoundException(String message, Throwable cause) {

**super**(message, cause);

}

**public** UsernameNotFoundException(String message) {

**super**(message);

}

**public** UsernameNotFoundException(Throwable cause) {

**super**(cause);

}

}

WrongPasswordException

**package** d1201;

**public** **class** WrongPasswordException **extends** Exception{

**public** WrongPasswordException() {

**super**();

}

**public** WrongPasswordException(String message, Throwable cause, **boolean** enableSuppression,

**boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

}

**public** WrongPasswordException(String message, Throwable cause) {

**super**(message, cause);

}

**public** WrongPasswordException(String message) {

**super**(message);

}

**public** WrongPasswordException(Throwable cause) {

**super**(cause);

}

}

Test4

**package** d1201;

**import** java.util.Scanner;

**public** **class** Test4 {

**public** **static** **void** main(String[] args) {

System.***out***.print("用户名：");

String n = **new** Scanner(System.***in***).nextLine();

System.***out***.print("密码：");

String p = **new** Scanner(System.***in***).nextLine();

**try** {

*login*(n, p);

System.***out***.println("欢迎登录");

} **catch** (UsernameNotFoundException e) {

System.***out***.println("用户名错误");

} **catch** (WrongPasswordException e) {

System.***out***.println("密码错误");

}

}

**private** **static** **void** login(String n, String p) **throws** UsernameNotFoundException, WrongPasswordException {

// "abc" "123"

**if**(! n.equals("abc")) {

**throw** **new** UsernameNotFoundException();

}

**if**(! p.equals("123")) {

**throw** **new** WrongPasswordException();

}

}

}

* 1. **RuntimeException 和 其他Exception**
     1. **RuntimeException**

非检查异常

编译器不检查RuntimeException 是否有处理代码

有默认的抛出管道

* + 1. **其他异常**

检查异常

编译器检查是否有处理代码，不处理，不能编译

强制开发人员事先考虑错误处理

* 1. **throws 和 try-catch**

不能随便 catch

底层的异常，一般要向上层抛，在上层处理

当经验不足时，不知道该catch还是throws，就选throws

1. **面向对象6**

## 接口

极端的抽象类

作用：结构设计工具，用来解耦合、隔离实现

用 interface 代替了 class 关键字

用 implements 代替了 extends 关键字

接口中只能定义：

* 公开的抽象方法
* 公开的常量
* 公开的内部类、内部接口

类可以同时继承多个接口

class A implements X,Y,Z {

}

class A extends B implements X,Y,Z {

}

接口和接口之间继承

interface A extends X,Y,Z {

}

1. **变形金刚**

项目：d1202\_变形金刚

类：d1202.Test1

Weapon

Sword

AK47

Lyb

Transformer

Weapon

**package** d1202;

**public** **interface** Weapon {

/\* public static final \*/

**int** ***TYPE\_COLD*** =0;

**int** ***TYPE\_HOT*** =1;

**int** ***TYPE\_NUCLEAR*** =2;

/\* public abstract \*/

**void** kill();

String getName();

**int** getType();

}

Sword

**package** d1202;

**public** **class** Sword **implements** Weapon {

@Override

**public** **void** kill() {

System.***out***.println("耍剑");

}

@Override

**public** String getName() {

**return** "倚天剑";

}

@Override

**public** **int** getType() {

**return** Weapon.***TYPE\_COLD***;

}

}

AK47

**package** d1202;

**public** **class** AK47 **implements** Weapon {

@Override

**public** **void** kill() {

System.***out***.println("耍枪");

}

@Override

**public** String getName() {

**return** "AK47";

}

@Override

**public** **int** getType() {

**return** Weapon.***TYPE\_HOT***;

}

}

Lyb

**package** d1202;

**public** **class** Lyb **implements** Weapon {

@Override

**public** **void** kill() {

System.***out***.println("耍棒");

}

@Override

**public** String getName() {

**return** "狼牙棒";

}

@Override

**public** **int** getType() {

**return** Weapon.***TYPE\_NUCLEAR***;

}

}

Transformer

**package** d1202;

**public** **class** Transformer {

//包含一个武器接口

**private** Weapon w; //默认 null

//给变量w赋值的方法

**public** **void** setWeapon(Weapon w) {

**this**.w = w;

}

**public** **void** attack() {

System.***out***.println("变形金刚进攻");

**if** (w == **null**) {

System.***out***.println("用牙咬");

**return**;

}

String t = "";

**switch**(w.getType()) {

**case** Weapon.***TYPE\_COLD***: t="冷兵器";**break**;

**case** Weapon.***TYPE\_HOT***: t="热兵器";**break**;

**case** Weapon.***TYPE\_NUCLEAR***: t="核武器";**break**;

}

//使用冷兵器倚天剑进攻

System.***out***.println("使用"+t+w.getName()+"进攻");

w.kill();

}

}

Test1

**package** d1202;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

Transformer t = **new** Transformer();

Sword s = **new** Sword();

AK47 a = **new** AK47();

Lyb l = **new** Lyb();

System.***out***.println("按回车执行");

**while**(**true**) {

**new** Scanner(System.***in***).nextLine();

**int** r = **new** Random().nextInt(4);

**switch** (r) {

//setWeapon()方法中，为变形金刚的成员w赋值

**case** 0: t.setWeapon(s); **break**;

**case** 1: t.setWeapon(a); **break**;

**case** 2: t.setWeapon(l); **break**;

**case** 3: t.setWeapon(**null**); **break**;

}

t.attack();

}

}

}

1. **集合（续）**

ArrayList

LinkedList

HashMap

* 1. **集合的继承结构**

Collection 接口

|- List 接口

|- ArrayList

|- LinkedList

|- Set 接口

|- HashSet

|- TreeSet

Map 接口

|- HashMap

|- TreeMap

Iterator 接口

Collections 工具类

1. **HashMap（续）**
   1. **哈希运算过程**

* HashMap内部，使用 Entry[] 数组存放数据
* 数组默认初始容量 16
* 放满后容量翻倍+2
* key.hashCode() 获得键的哈希值
* 用哈希值和数组长度，运算产生一个下标值 i
* 新建Entry 对象来封装键值对数据
* Entry对象，放在 i 位置
  + 如果是空位置，直接放入
  + 有数据，依次equals()比较是否相等
    - 找到相等的，覆盖旧值
    - 没有相等的，链表连接在一起
  + 负载率、加载因子 0.75
    - 新建翻倍容量的新数组
    - 所有数据，重新执行哈希运算，放入新数组
  + jdk1.8
    - 链表长度到8，转成红黑树
    - 树上的数据减少到6，转回成链表
  1. **hashCode()**

hashCode() 是 Object 的方法，默认实现是用内存地址作为哈希值

如果希望相同属性，具有相同哈希值，可以重写这个方法

1. **哈希运算测试**

项目：d1203\_哈希运算

类：d1203.Test1

Point

Point

**package** d1203;

**public** **class** Point {

**private** **int** x;

**private** **int** y;

**public** Point() {

**super**();

}

**public** Point(**int** x, **int** y) {

**super**();

**this**.x = x;

**this**.y = y;

}

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

@Override

**public** String toString() {

**return** "Point [x=" + x + ", y=" + y + "]";

}

@Override

**public** **int** hashCode() {

/\*

\* \*) 相同属性，产生相同的哈希值

\* \*) 不同属性，要尽量产生分散的哈希值

\*

\* 数学家发明了一种算法，

\* 是一种惯用的，有效算法

\*/

**int** p = 31; //神奇值31

**int** r = 1; //r起始值任意

r = r\*p + x;

r = r\*p + y;

**return** r;

}

@Override

**public** **boolean** equals(Object obj) {

//参数obj是Point类型

**if**(obj **instanceof** Point) {

Point p = (Point) obj;

**return** x == p.x && y == p.y;

}

**return** **false**;

}

}

Test1

**package** d1203;

**import** java.util.HashMap;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

Point a = **new** Point(1, 3);// -- "2.9亿"

Point b = **new** Point(1, 3);// -- "3.1亿"

//有相同的哈希值，才能保证计算出相同下标位置

System.***out***.println(a.hashCode());

System.***out***.println(b.hashCode());

//即使计算出相同位置，

//equals()也要像等，才能覆盖，否则链表连在一起

System.***out***.println(a.equals(b));

HashMap<Point, String> map = **new** HashMap<>();

map.put(a, "2.9亿");

map.put(b, "3.1亿");

System.***out***.println(map);

}

}

1. **学生对应成绩**

学生对象 -- 98

项目：dy1301\_学生对应成绩

类：d1301.Test1

Student

Student

**package** d1301;

**public** **class** Student {

**private** **int** id;

**private** String name;

**private** String gender;

**private** **int** age;

**public** Student() {

**super**();

}

**public** Student(**int** id, String name, String gender, **int** age) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.gender = gender;

**this**.age = age;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getGender() {

**return** gender;

}

**public** **void** setGender(String gender) {

**this**.gender = gender;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + ", gender=" + gender + ", age=" + age + "]";

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + age;

result = prime \* result + ((gender == **null**) ? 0 : gender.hashCode());

result = prime \* result + id;

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Student other = (Student) obj;

**if** (age != other.age)

**return** **false**;

**if** (gender == **null**) {

**if** (other.gender != **null**)

**return** **false**;

} **else** **if** (!gender.equals(other.gender))

**return** **false**;

**if** (id != other.id)

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

}

Test1

**package** d1301;

**import** java.util.HashMap;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

Student s1 = **new** Student(9527,"唐伯虎","男",19);

Student s2 = **new** Student(9527,"唐伯虎","男",19);

//哈希值相同，才能保证计算出相同下标位置

System.***out***.println(s1.hashCode());

System.***out***.println(s2.hashCode());

//即使计算出相同下标，equals()也必须相等，才能覆盖

System.***out***.println(s1.equals(s2));

HashMap<Student, Integer> map = **new** HashMap<>();

**for** (**int** i = 0; i < 20; i++) {

map.put(**new** Student(), 99);

}

System.***out***.println(map);

}

}

1. **io**

Input / Ouput

输入 / 输出

图示

描述已自动生成

java.io 包

File

InputStream/OutputStream

FileInputStream/FileOutputStream

ObjectinputStream/ObjectOutputStream

Reader/Writer

InputStreamReader/OutputStreamWriter

BufferedReader

PrintWriter、PrintStream

1. **File**

封装一个磁盘路径字符串，提供了一组对文件、文件夹操作的方法

可以封装文件路径、文件夹路径、不存在的路径

{path="d:/abc"}

{path="d:/a.txt"}

{path="f:/xx/xxx/xx"}

* 1. **文件、文件夹属性方法**

getName()

文件名

getParent()

父目录

getAbsolutePath()

完整路径，绝对路径

length()

文件的字节量，对文件夹无效

isFile()

是否是文件

isDirectory()

是否是文件夹

* 1. **创建、删除**

createNewFile()

新建文件

文件已经存在，不会再新建，返回 false

所在的文件夹不存在，会出现异常

mkdirs()

逐层创建多层文件夹

delete()

删除文件，或空目录

* 1. **目录列表**

list()

得到 String[]，包含所有的文件名

["a.txt", "b.mp3", "c.jpg", "aaa"]

listFiles()

得到 File[]，包含所有文件的封装的File对象

[{...}, {...}, {...}]

1. **File**

项目：d1302\_File

类：d1302.Test1

**package** d1302;

**import** java.io.File;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

String path;

//path = "D:\\home\\java\\eclipse\\eclipse.exe";//存在的文件

//path = "D:/home/java/eclipse";//存在的文件夹

path = "f:/xxx/xx/xx";//不存在的路径

File f = **new** File(path);//封装路径

System.***out***.println(f.getName());//取文件名

System.***out***.println(f.getParent());//取父目录

System.***out***.println(f.getAbsolutePath());//取完整路径

System.***out***.println(f.length());//文件字节量，目录无效

System.***out***.println(f.isFile());//是否是文件

System.***out***.println(f.isDirectory());//是否是文件夹

}

}

Test2

**package** d1302;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.Scanner;

**public** **class** Test2 {

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

/\*

\* d:/abc/aa/

\* d:/abc/aa/f1

\*

\* 删除f1

\* 删除aa文件夹

\*/

System.***out***.println("按回车执行");

**new** Scanner(System.***in***).nextLine();

File dir = **new** File("d:/abc/aa");

dir.mkdirs();

**new** Scanner(System.***in***).nextLine();

File file = **new** File(dir, "f1");

file.createNewFile();

**new** Scanner(System.***in***).nextLine();

file.delete();

**new** Scanner(System.***in***).nextLine();

dir.delete();

}

}

Test3

**package** d1302;

**import** java.io.File;

**import** java.util.Scanner;

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

System.***out***.println("文件夹：");

String s = **new** Scanner(System.***in***).nextLine();

File dir = **new** File(s);

**if** (! dir.isDirectory()) {

System.***out***.println("请输入正确的文件夹");

**return**;

}

/\*

\* 如果对文件进行列表、

\* 对不存在的路径列表、

\* 对无权访问的文件夹列表

\* 都返回 null 值

\*/

String[] names = dir.list();

File[] files = dir.listFiles();

**if**(names ==**null**) {

System.***out***.println("无法获得目录列表");

**return**;

}

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

System.***out***.println(names[i]);

}

System.***out***.println("\n\n-------------------------");

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

File f = files[i];

System.***out***.println(

f.getName()+" - "+f.length());

}

}

}

1. **求文件夹的总大小**

[a]

|- b

|- c

|- [d]

|- e

|- [f]

|- h

|- i

|- [g]

|- j

|- k

Test4

**package** d1302;

**import** java.io.File;

**import** java.util.Scanner;

**public** **class** Test4 {

**public** **static** **void** main(String[] args) {

System.***out***.println("文件夹：");

String s = **new** Scanner(System.***in***).nextLine();

File dir = **new** File(s);

**if** (! dir.isDirectory()) {

System.***out***.println("请输入正确的文件夹");

**return**;

}

**long** size = *dirLength*(dir);

System.***out***.println(size);

}

**private** **static** **long** dirLength(File dir) {

//对dir列表

File[] list = dir.listFiles();

**if** (list == **null**) {//不能获得列表

**return** 0;

}

**long** sum = 0;

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

File f = list[i];

**if** (f.isFile()) {//f是文件

sum += f.length();

} **else** {//f是文件夹

sum += *dirLength*(f);//递归求f文件夹大小

}

}

**return** sum;

}

}

1. **流 Stream**

数据的读写操作（io操作），抽象成数据在管道中流动

单方向流动

* 输入流，只能用来读取数据（读入内存）
* 输出流，只能用来输出数据（内存数据向外输出）

只能从头到尾，顺序流动一次，不能反复流动，如果要重复流动，可以重新创建新的流

1. **InputStream,OutputStream**

字节流的抽象父类

方法：

**write(int b)**

只输出int四个字节中，末尾的一个字节值

[1][2][3][4] ---> [4]

**write(byte[], start, length)**

输出byte[] 数组中，从start开始的length个字节值

**read()**

读取一个字节值，补三个0字节，变成int

[4] ---> [1][2][3][4]

读取结束后，再读取，返回 -1

**read(byte[] buff)**

按数组的长度，读取一批字节值，存放到指定的数组中，并返回这一批的字节数量

读取结束后，再读取，返回 -1

1. **FileInputStream,FileOutputStream**
2. **文件流**

项目：d1303\_文件流

类：d1303.Test1

**package** d1303;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**public** **class** Test1 {

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

/\*

\* FOS--d:/abc/f2

\*

\* new FileOutputStream(文件)

\* 不管文件是否存在，都新建空文件

\*/

FileOutputStream out =

**new** FileOutputStream("d:/abc/f2");

out.write(97);//00 00 00 61 --> 61

out.write(98);//00 00 00 62 --> 62

out.write(99);//00 00 00 63 --> 63

out.write(356);//00 00 01 64 --> 64

**byte**[] a = {

101,102,103,104,105,

106,107,108,109,110

};

out.write(a, 0, 10);

out.write(a, 3, 4);

out.close();//释放系统资源

}

}

Test2

**package** d1303;

**import** java.io.FileInputStream;

**public** **class** Test2 {

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

/\*

\* FIS--f2

\*/

FileInputStream in =

**new** FileInputStream("d:/abc/f2");

//标准格式

**int** b;

**while**((b = in.read()) != -1) {

System.***out***.println(b);

}

in.close();

/\*int b;

while(true) {

b = in.read();

if(b == -1) {

break;

}

System.out.println(b);

}\*/

}

}

Test3

**package** d1303;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.util.Arrays;

**public** **class** Test3 {

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

/\*

\* FIS--f2

\*/

FileInputStream in =

**new** FileInputStream("d:/abc/f2");

//批量读取标准格式，写了删，删了写

**byte**[] buff = **new** **byte**[5];

**int** n;//用来保存每次读取的数量

**while**((n = in.read(buff)) != -1) {

System.***out***.println(

n+"个："+Arrays.*toString*(buff));

}

in.close();

/\*while(true) {

n = in.read(buff);

if (n == -1) {

break;

}

System.out.println(

n+"个："+Arrays.toString(buff));

}\*/

}

}

1. **文件复制**

Test4

**package** d1303;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.util.Scanner;

**public** **class** Test4 {

/\*

\* 原文件：d:/abc/f2

\* 目标文件： d:/def/f3

\*/

**public** **static** **void** main(String[] args) {

System.***out***.println("原文件：");

String s1 = **new** Scanner(System.***in***).nextLine();

File from = **new** File(s1);

**if**(! from.isFile()) {

System.***out***.println("请输入正确的文件");

**return**;

}

System.***out***.println("目标文件：");

String s2 = **new** Scanner(System.***in***).nextLine();

File to = **new** File(s2);

**if** (to.isDirectory()) {

System.***out***.println(

"请具体到文件，不能是文件夹");

**return**;

}

**try** {

*copy*(from, to);

System.***out***.println("完成");

} **catch** (Exception e) {

System.***out***.println("失败");

e.printStackTrace();

}

}

**private** **static** **void** copy(

File from, File to) **throws** Exception {

/\*

\* 文件输入流插在 from

\* 文件输出流插在 to

\* 从头到尾读取 from 原文件数据，

\* 读取的数据，直接向目标文件 to 输出

\*/

FileInputStream in =

**new** FileInputStream(from);

FileOutputStream out =

**new** FileOutputStream(to);

/\*int b;

while((b = in.read()) != -1) {

out.write(b);

}\*/

// 8k - 8192

**byte**[] buff = **new** **byte**[8192];

**int** n;//保存每一批的数量

**while**((n = in.read(buff)) != -1) {

System.***out***.println(n);

out.write(buff, 0, n);

}

in.close();

out.close();

}

}

# BufferedInputStream, BufferedOutputStream

缓冲流

内部使用一个 8k 数组，来加速单字节读写速度

替代自己创建数组做批量读写

1. **ObjectInputStream，ObjectOutputStream**

对象序列化、反序列化

序列化 把一个对象的信息，按固定的字节格式，变成一串字节序列输出

图标

低可信度描述已自动生成

方法：

**writeObject(Object obj)**

把对象变成一串字节序列输出

**readObject()**

读取序列化数据，反序列化恢复对象

* 1. **Serializable 接口**

被序列化的对象，必须实现 Serializable 接口

* 1. **不序列化的变量**

static - 属于类，不随对象被序列化输出

transient - 临时，只在程序运行期间，在内存中存在，不会被序列化持久保存

* 1. **序列化版本**

控制旧版本的数据，不允许恢复成新版本的类型

如果自己不定义版本，编译器编译时，会根据类的定义信息，计算生成一个版本值

static final long serialVersionUID

1. **序列化**

项目：d1304\_序列化

类：d1304.Test1

Student

Student

**package** d1304;

**import** java.io.Serializable;

/\*

\* 空接口，

\* 标识接口，用来标识学生类型实例，允许被序列化

\*/

**public** **class** Student **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 2019L;

**private** **int** id;

**private** String name;

**private** **static** String *gender*;

**private** **transient** **int** age;

**public** Student() {

}

**public** Student(**int** id, String name, String gender, **int** age) {

**this**.id = id;

**this**.name = name;

**this**.*gender* = gender;

**this**.age = age;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getGender() {

**return** *gender*;

}

**public** **void** setGender(String gender) {

**this**.*gender* = gender;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + ", gender=" + *gender* + ", age=" + age + "]";

}

}

Test1

**package** d1304;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectOutputStream;

**public** **class** Test1 {

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

Student s =

**new** Student(9527,"唐伯虎","男",19);

/\*

\* OOS--FOS--f3

\*/

ObjectOutputStream out =

**new** ObjectOutputStream(

**new** FileOutputStream("d:/abc/f3"));

out.writeObject(s);

out.close();

}

}

Test2

**package** d1304;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**public** **class** Test2 {

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

/\*

\* OIS--FIS--f3

\*/

ObjectInputStream in =

**new** ObjectInputStream(

**new** FileInputStream("d:/abc/f3"));

Student s = (Student) in.readObject();

System.***out***.println(s);

in.close();

}

}

1. **练习**

* 在文件夹中查找所有的图片文件，包括深层目录文件

jpg,png,gif,bmp

1. **文件加密复制**

f1

---------------

10010101 11101101 01101110

---------------

f2

---------------

11011010 11001001 00100101

---------------

10010101

00011101 ^

10001000

项目：d1401\_文件加密复制

类：d1401.Test1

**package** d1401;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("原文件：");

String s1 = **new** Scanner(System.***in***).nextLine();

File from = **new** File(s1);

**if**(! from.isFile()) {

System.***out***.println("请输入正确的文件");

**return**;

}

System.***out***.println("目标文件：");

String s2 = **new** Scanner(System.***in***).nextLine();

File to = **new** File(s2);

**if** (to.isDirectory()) {

System.***out***.println(

"请具体到文件，不能是文件夹");

**return**;

}

System.***out***.print("KEY："); //0到255，超出255只有后面8位起作用

**int** key = **new** Scanner(System.***in***).nextInt();

**try** {

*copy*(from, to, key);

System.***out***.println("完成");

} **catch** (Exception e) {

System.***out***.println("失败");

e.printStackTrace();

}

}

**private** **static** **void** copy(

File from, File to, **int** key) **throws** Exception {

FileInputStream in = **new** FileInputStream(from);

FileOutputStream out = **new** FileOutputStream(to);

**byte**[] buff = **new** **byte**[8192];

**int** n;

**while**((n = in.read(buff)) != -1) {

/\* 先把buff数组中的前n个字节值加密

\* 加密之后再向目标文件输出 \*/

**for** (**int** i = 0; i < n; i++) {

buff[i] ^= key;

}

out.write(buff, 0, n);

}

in.close();

out.close();

}

}

1. **面向对象7**

## 内部类

定义在类内部、方法内部或局部代码块内部的类

### 非静态内部类、属于实例的内部类

class A {

class Inner {

}

}

非静态内部类实例，必须依赖于一个外部类的实例才能存在

A a1 = new A();

Inner i = a1.new Inner();

用来辅助外部实例运算，封装局部数据，或局部的运算逻辑

### 静态内部类

class A {

static class Inner {

}

}

Inner i = new Inner();

静态内部类，与普通的类没有区别

静态内部类，只是一个设计的选项，可以选择嵌套定义，也可以选择独立定义

### 局部内部类

class A {

Weapon f() {

class Inner implements Weapon{

}

Inner i = new Inner();

return i;

}

}

A a = new A();

Weapon w = a.f();

局部定义的类型，类似于局部变量，有作用范围，只能在局部代码块内使用这种类型

### 匿名内部类

Weapon w = new Weapon() {...};

* {} - 匿名类
* new - 新建匿名类的实例
* Weapon - 父类型
* () - super()，可传参数super(1,2,3)

1. **内部类**

项目：d1402\_内部类

类：d1402.Test1

**package** d1402;

**import** d1402.A.Inner1;

**import** d1402.A.Inner2;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

Inner1 i1 = **new** Inner1();

System.***out***.println(i1);

A a = **new** A();

Inner2 i2 = a.**new** Inner2();

System.***out***.println(i2);

}

}

**class** A {

**static** **class** Inner1 {

}

**class** Inner2 {

}

}

Test2

**package** d1402;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

// 得到的是 Weapon 的子类实例

Weapon w1 = *f1*();

w1.kill();

System.***out***.println(w1);

System.***out***.println("--------------------");

Weapon w2 = *f2*("方天画戟");

w2.kill();

System.***out***.println(w2);

}

**private** **static** Weapon f2(String name) {

/\*

\* 局部内部类中，使用外面的局部变量，

\* 必须加 final，jdk1.8，缺省

\*/

Weapon w = **new** Weapon() {

@Override

**public** **void** kill() {

System.***out***.println(

"使用"+name+"进攻");

}

};

**return** w;

}

**private** **static** Weapon f1() {

**class** AK47 **implements** Weapon {

@Override

**public** **void** kill() {

System.***out***.println(

"使用AK47进攻");

}

}

AK47 a = **new** AK47();

**return** a;

}

}

1. **for-each循环**

数组遍历、集合迭代遍历的语法简化

数组

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

String s = a[i];

//处理s

}

for-each 简化语法：

for(String s : a) {

//处理s

}

集合迭代遍历

for(Iterator<String> it=list.iterator(); it.hasNext(); ) {

String s = it.next();

//处理s

}

for-each 语法简化

for(String s : list) {

//处理

}

1. **手写双向链表**

模仿 LinkedList，实现双向链表数据结构

项目：d1403\_手写双向链表

类：d1403.Test1

MyList

**package** d1403;

/\*

\* {

\* {...}-{...}-{...}

\* }

\*

\* {

\* 123{ ... }

\*

\* {

\* prev: 123,

\* value: 8,

\* next: 456

\* }

\*

\* 456{ ... }

\* }

\*/

**public** **class** MyList {

//首尾节点

**private** Node first; //null

**private** Node last; //null

//数据的数量

**private** **int** size; //数据的数量，默认0

**public** **void** add(Object value) {

//如果添加的是第一个节点

Node n = **new** Node();

n.value = value;

//

**if** (size == 0) {

//添加第一个

n.prev = n;

n.next = n;

//

first = n;

last = n;

} **else** {

//在尾部新加数据

//n和last连接

n.prev = last;

last.next = n;

//n和first连接

n.next = first;

first.prev = n;

//新的节点n，变成尾部节点

last = n;

}

size++;

}

**public** Object get(**int** i) {

//越界

**if**(i<0 || i>=size) {

**throw**

**new** IndexOutOfBoundsException(""+i);

}

//访问 i 位置的节点对象

Node n = getNode(i);

//返回节点中封装的数据

**return** n.value;

}

**private** Node getNode(**int** i) {

**if** (i==0) {

**return** first;

} **else** **if**(i==size-1){

**return** last;

}

**if** (i<size/2) {//i<中间位置

Node n = first;

//从下标1到下标i

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

n = n.next;

}

**return** n;

} **else** {

Node n = last;

//从size-2到i

**for** (**int** j = size-2; j >= i; j--) {

n = n.prev;

}

**return** n;

}

}

**public** **int** size() {

**return** size;

}

//内部类，辅助双向链表，

//封装局部的数据：保存的值和前后引用

**private** **class** Node {

Node prev;

Node next;

Object value;

}

}

Test2

**package** d1403;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

MyList list = **new** MyList();

list.add("aaa");

list.add("rrr");

list.add("hhh");

list.add("qqq");

list.add("jjj");

list.add("iii");

list.add("ggg");

System.***out***.println(list.size());

System.***out***.println(list.get(0));

System.***out***.println(list.get(list.size()-1));

System.***out***.println(list.get(3));

**for** (**int** i = 0; i < list.size(); i++) {

System.***out***.println(list.get(i));

}

}

}

1. **字符编码**
   1. **ASC-II**

0到 127，英文、指令字符

* 1. **iso-8859-1，Latin-1**

西欧编码

ASC-II扩展到255

* 1. **CJK编码**

亚洲编码，中日韩

* 1. **GBK**

国标码

英文单字节

中文双字节

* 1. **Unicode**

统一码，万国码

100万+

常用表，双字节

生僻字符表，三字节、四字节...

* 1. **UTF-8**

Unicode 的传输格式

Unicode Transformation format

英文，单字节

某些字符，双字节

中文，三字节

特殊符号，四字节

* 1. **Java的char类型是 Unicode**

Java的转码运算

Unicode ---> 其他编码

String s = "abc中文";

//转成系统默认编码

byte[] a = s.getBytes();

//转成指定编码

byte[] a = s.getBytes("UTF-8");

其他编码 --> Unicode

//从系统默认编码转成 Unicode

String s = new String(byte[]);

//从指定编码转成 Unicode

String s = new String(byte[], "UTF-8");

1. **编码转换**

项目：d1404\_编码转换

类：d1404.Test1

**package** d1404;

**import** java.io.UnsupportedEncodingException;

**import** java.util.Arrays;

**public** **class** Test1 {

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

String s = "abc中文";

System.***out***.println(s);

*f*(s, "UTF-16Be");

*f*(s, **null**);

*f*(s, "GBK");

*f*(s, "UTF-8");

}

/\*

\* encoding 编码

\* charset 字符集

\*

\* a 中

\* Unicode 00 61 4e 2d

\* UTF-8 61 e4 b8 ad

\* GBK 61 d6 d0

\*/

**private** **static** **void** f(

String s, String charset) **throws** Exception {

**byte**[] a;

**if** (charset == **null**) {

//Unicode ---> 系统默认编码

a = s.getBytes();

} **else** {

//Unicode ---> 指定的编码

a = s.getBytes(charset);

}

System.***out***.println(

charset+"\t"+Arrays.*toString*(a));

// 转回Unicode字符

**if** (charset == **null**) {

//从系统默认编码转回Unicode

s = **new** String(a);

} **else** {

//从指定编码转回Unicode

s = **new** String(a, charset);

}

System.***out***.println(s);

System.***out***.println("-------------------");

}

}

1. **Reader，Writer**

字符流的抽象父类，以字符为单位读写数据

1. **InputStreamReader,OutputStreamWriter**

字符编码转换流

OutputStreamWriter

把Java的Unicode编码字符，转成其他编码输出

InputStreamReader

读取其他编码字符，转成Unicode字符

1. **编码转换流**

项目：d1405\_编码转换流

类：d1405.Test1

**package** d1405;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.InputStreamReader;

**import** java.io.UnsupportedEncodingException;

**public** **class** Test1 {

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

/\*

\* f4 61 d6 d0

\* f5 61 e4 b8 ad

\*/

*f*("d:/abc/f4", "GBK");

*f*("d:/abc/f5", "UTF-8");

}

**private** **static** **void** f(

String path, String charset) **throws** Exception {

/\*

\* ISR--FIS--path, charset

\*

\* InputStreamReader

\* 读取其他编码，转成 Unicode 编码

\*/

InputStreamReader in =

**new** InputStreamReader(

**new** FileInputStream(path), charset);

System.***out***.println((**char**)in.read());

System.***out***.println((**char**)in.read());

in.close();

}

}

Test2

**package** d1405;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.OutputStreamWriter;

**import** java.io.UnsupportedEncodingException;

**public** **class** Test2 {

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

/\*

\* f6 GBK

\* f7 UTF-8

\*

\* 把Unicode编码所有的中文字符，输出到两个文件

\* \u4e00到\u9fa5，20902个中文

\*/

*f*("d:/abc/f6", "GBK");

*f*("d:/abc/f7", "UTF-8");

}

**private** **static** **void** f(

String path, String charset) **throws** Exception {

/\*

\* OSW--FOS--path

\*/

OutputStreamWriter out =

**new** OutputStreamWriter(

**new** FileOutputStream(path), charset);

**int** count=0;

**for** (**char** i = '\u4e00'; i <= '\u9fa5'; i++) {

out.write(i);

count++;

**if**(count == 30) {

out.write('\n');

count=0;

}

}

out.close();

}

}

1. **查找图片文件**

项目：d1501\_查找图片文件

类：d1501.Test1

**package** d1501;

**import** java.io.File;

**import** java.io.FileFilter;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("输入文件夹：");

String s = **new** Scanner(System.***in***).nextLine();

File dir = **new** File(s);

**if**(! dir.isDirectory()) {

System.***out***.println("请输入正确的文件夹");

**return**;

}

List<File> list = *f*(dir);

**for** (File f : list) {

System.***out***.println(f.getAbsolutePath());

}

}

**private** **static** List<File> f(File dir) {

ArrayList<File> pic = **new** ArrayList<>();

LinkedList<File> stack = **new** LinkedList<>();

stack.push(dir);//初始状态，dir先压入

**while** (! stack.isEmpty()) {//栈中还有数据

File f = stack.pop();//弹出

**if** (f.isFile()) { //f是文件

//if(isPic(f)) {//f是图片文件

pic.add(f);//图片添加到pic集合

//}

} **else** { //f是文件夹

FileFilter filter = **new** FileFilter() {

@Override

**public** **boolean** accept(File f) {

**if** (f.isDirectory()) {

**return** **true**;//接受文件夹

}

//也接受图片文件

String n = f.getName().toLowerCase();

**return** n.matches(".+\\.(jpg|png|gif|bmp)");

}

};

File[] list = f.listFiles(filter);

**if**(list == **null**) {

**continue**;//列不出来不处理，继续弹出下一项

}

**for** (File f2 : list) {//遍历，全部压入栈中

stack.push(f2);

}

}

}

**return** pic;

}

**private** **static** **boolean** isPic(File f) {

// sdf.sdfs.sdf.jpg, a.JPG, .Jpg

String n = f.getName().toLowerCase();

**return** n.matches(".+\\.(jpg|png|gif|bmp)");

}

}

1. **集合排序**

ArrayList，存放日期格式的字符串，排序

"2019-4-1"

"2019-4-10"

"2019-4-11"

"2019-4-12"

"2019-4-2"

"2019-4-20"

"2019-4-21"

"2019-4-22"

项目：d1502\_集合排序

类：d1502.Test1

**package** d1502;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.Date;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

ArrayList<String> list = **new** ArrayList<>();

Collections.*addAll*(

list,

"2019-4-2","2019-4-10",

//"dfhgfghsdfgsd",

"2019-4-21","2019-4-30","2019-4-1",

"2019-4-20","2019-4-12","2019-4-3",

"2019-4-11","2019-4-22");

System.***out***.println(list);

//排序

Collections.*sort*(list);

System.***out***.println(list);

//排序

/\*

\* sort()方法，可以使用一个接口，接入一个比较器对象

\* sort(list, 比较器)

\* sort()方法，排序运算过程中，

\* 对数据比大小时，调用比较器来比较

\*/

Collections.*sort*(list, **new** Comparator<String>() {

/\*

\* o1和o2比较大小

\* o1大，正数

\* o1小，负数

\* 相同，0

\*

\* 重写父类的方法，异常管道，不能比父类多

\*/

@Override

**public** **int** compare(String o1, String o2) {

//日期字符串，解析成Date

SimpleDateFormat f =

**new** SimpleDateFormat("yyyy-MM-dd");

**try** {

Date a = f.parse(o1);

Date b = f.parse(o2);

**return** a.compareTo(b);

} **catch** (Exception e) {

//包装成能抛的，再抛出

**throw** **new** IllegalArgumentException(e);

}

}

});

System.***out***.println(list);

}

}

1. **文本文件中的日期字符串，排序后保存成新文件**

d:/abc/f8

----------------

2019年4月2

2019年4月10

2019年4月21

2019年4月30

2019年4月1

2019年4月20

2019年4月12

2019年4月3

2019年4月11

2019年4月22

项目：d1503\_文本文件按行读写

类：d1503.Test1

**BufferedReader**，readLine()

读取一行字符串，读完再读，返回null

**PrintWriter**, println()

打印输出一行字符串，末尾补换行

**package** d1503;

**import** java.io.BufferedReader;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.io.UnsupportedEncodingException;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.util.List;

**public** **class** Test1 {

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

//读取 f8 文件，所有日期字符串存入集合

List<String> list = *f1*();

//对list中的日期字符串排序

*f2*(list);

//保存到 f9 文件

*f3*(list);

}

**private** **static** List<String> f1() **throws** Exception {

/\*

\* BR--ISR--FIS--f8

\*

\* FIS - 插在文件上

\* ISR - 处理编码转换

\* BR - 按行读取

\*/

BufferedReader in =

**new** BufferedReader(//处理换行，按行读取

**new** InputStreamReader(//编码转换流

**new** FileInputStream("d:/abc/f8"),"GBK"));

ArrayList<String> list = **new** ArrayList<>();

String line = **null**;

**while**((line = in.readLine()) != **null**) {

// trim() 去除两端的空白字符

String s = line.trim();

**if** (s.length() == 0) {

**continue**;

}

list.add(s);

}

in.close();

System.***out***.println(list);

**return** list;

}

**private** **static** **void** f2(List<String> list) {

Collections.*sort*(list, **new** Comparator<String>() {

@Override

**public** **int** compare(String o1, String o2) {

SimpleDateFormat f =

**new** SimpleDateFormat("yyyy年MM月dd日");

**try** {

Date a = f.parse(o1);

Date b = f.parse(o2);

**return** a.compareTo(b);

} **catch** (Exception e) {

**throw** **new** IllegalArgumentException(e);

}

}

});

System.***out***.println(list);

}

**private** **static** **void** f3(List<String> list) **throws** Exception {

/\*

\* PW--OSW--FOS--f9

\*

\* FOS - 插在文件上

\* OSW - 编码转换

\* PW - println()自动加换行

\*/

PrintWriter out =

**new** PrintWriter(

**new** OutputStreamWriter(

**new** FileOutputStream("d:/abc/f9"),"GBK"));

**for** (String s : list) {

out.println(s);

}

out.close();

}

}

# FileReader / FileWriter

只能使用系统的默认编码，不能自己指定

## FileReader

FileInputStream + InputStreamReader 的封装对象

## FileWriter

FileOutputStream + OutputStreamWriter 的封装对象

1. **线程**
   1. **进程**

在操作系统中，并行执行的任务

图表, 直方图

描述已自动生成

* 1. **线程**

在进程内部，并行执行的任务

文本

描述已自动生成

* 1. **创建线程（两种方式）**
* 继承 Thread
* 实现 Runnable
  1. **继承 Thread**

编写 Thread 的子类，并重写 run() 方法。启动之后，自动运行 run() 方法中的代码

信件

中度可信度描述已自动生成

1. **线程**

项目：d1504\_线程

类：d1504.Test1

**package** d1504;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

T1 t1 = **new** T1();

T1 t2 = **new** T1();

//启动线程

//线程启动后，自动执行 run() 方法

t1.start();

t2.start();

System.***out***.println("main");

}

**static** **class** T1 **extends** Thread {

@Override

**public** **void** run() {

//获取线程名称

String n = getName();

//打印1到1000

**for** (**int** i = 1; i <= 1000; i++) {

System.***out***.println(n+" - "+i);

}

}

}

}

图片包含 图表

描述已自动生成

* 1. **实现 Runnable**

实现 Runnable 接口，实现它的 run() 方法，Runnable 封装在线程中执行的代码，新建线程对象时，把Runnable对象放在线程内，启动

图示

低可信度描述已自动生成

1. **Runnable**

Test2

**package** d1504;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

R1 r1 = **new** R1();

Thread t1 = **new** Thread(r1);

Thread t2 = **new** Thread(r1);

//线程启动后，自动执行 r1.run()

t1.start();

t2.start();

}

//包含要在线程中运行代码的对象

**static** **class** R1 **implements** Runnable {

@Override

**public** **void** run() {

//获得正在执行的线程实例

Thread t = Thread.*currentThread*();

//获得线程名

String n = t.getName();

//打印1到1000

**for** (**int** i = 1; i <= 1000; i++) {

System.***out***.println(n+" - "+i);

}

}

}

}

* 1. **线程的状态**

图示

描述已自动生成

* 1. **线程的方法**

**Thread.currentThread()**

获得正在执行的线程实例

**Thread.sleep(毫秒值)**

让正在执行的线程，暂停指定的毫秒值时长

**getName()**

**setName()**

线程名

**start()**

启动线程

**interrupt()**

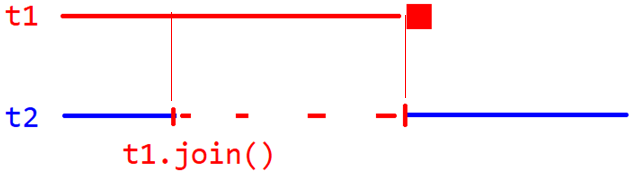
打断线程的暂停状态

形状, 多边形

描述已自动生成

**join()**

当前线程暂停，等待被调用的线程结束



**setDaemon(true)**

后台线程、守护线程

JVM虚拟机退出条件，是所有前台线程结束，当所有前台线程结束，虚拟机会自动退出

不会等待后台线程结束

例如：垃圾回收器是一个后台线程

1. **循环打印时间**

Test3

**package** d1504;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Scanner;

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

T1 t1 = **new** T1();

t1.start();

Thread t2 = **new** Thread() {

@Override

**public** **void** run() {

System.***out***.println("按回车捅醒 t1");

**new** Scanner(System.***in***).nextLine();

t1.interrupt();

}

};

t2.start();

}

**static** **class** T1 **extends** Thread {

@Override

**public** **void** run() {

SimpleDateFormat f =

**new** SimpleDateFormat("HH:mm:ss");

**while**(**true**) {

String s = f.format(**new** Date());

System.***out***.println(s);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

System.***out***.println("谁 TMD 捅醒了老子");

**break**;

}

}

}

}

}

1. **join**

Test4

**package** d1504;

**public** **class** Test4 {

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

//1000万内，有多少个质数

//2,3,5,7,11,13,17,19,23....

System.***out***.println("\n--单线程-----------------");

*f1*();

System.***out***.println("\n--5个线程-----------------");

*f2*();

}

**private** **static** **void** f1() **throws** InterruptedException {

**long** t = System.*currentTimeMillis*();

T1 t1 = **new** T1(0, 10000000);

t1.start();

//main线程暂停等待 t1 的执行结果

t1.join();

**int** c = t1.count;

t = System.*currentTimeMillis*()-t;

System.***out***.println("数量："+c);

System.***out***.println("时间："+t);

}

**private** **static** **void** f2() **throws** InterruptedException {

**long** t = System.*currentTimeMillis*();

**int** n = 5;

**int** m = 10000000/n;

T1[] a = **new** T1[n];

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

a[i] = **new** T1(m\*i, m\*(i+1));

a[i].start();

}

**int** sum = 0;

**for** (T1 t1 : a) {

t1.join();

sum += t1.count;

}

t = System.*currentTimeMillis*()-t;

System.***out***.println("数量："+sum);

System.***out***.println("时间："+t);

}

**static** **class** T1 **extends** Thread {

**int** from;

**int** to;

**int** count;//计数变量

**public** T1(**int** from, **int** to) {

**if**(from<3) {

from = 3;

count = 1;

}

**this**.from = from;

**this**.to = to;

}

@Override

**public** **void** run() {

**for** (**int** i = from; i < to; i++) {

**if** (isPrime(i)) {//判断i是否是质数

count++;

}

}

}

**private** **boolean** isPrime(**int** i) {

/\*

\* 从2到 (i开方+1)

\* 找有能把i整除的，i不是质数

\* 都不能吧i整除，i是质数

\*/

**double** d = Math.*sqrt*(i) + 1;

**for** (**int** j = 2; j < d; j++) {

**if** (i%j == 0) {

**return** **false**;

}

}

**return** **true**;

}

}

}

图表

描述已自动生成

## 练习

* 模仿d1501\_查找图片文件
  + 用栈运算方式，来计算文件夹的总大小

1. **加密复制文件夹**

项目：d1601\_加密复制文件夹

类：d1601.Test1

**package** d1601;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("原文件夹：");

String s1 = **new** Scanner(System.***in***).nextLine();

File from = **new** File(s1);

**if** (! from.isDirectory()) {

System.***out***.println("请输入正确的文件夹");

**return**;

}

System.***out***.println("目标文件夹：");

String s2 = **new** Scanner(System.***in***).nextLine();

File to = **new** File(s2);

**if** (to.exists()) {//如果已经存在

System.***out***.println("目标已经存在，不能复制");

**return**;

}

System.***out***.println("key: ");

**int** key = **new** Scanner(System.***in***).nextInt();

**try** {

*encryptCopyDir*(from, to, key);

System.***out***.println("完成");

} **catch** (Exception e) {

System.***out***.println("失败");

e.printStackTrace();

}

}

**private** **static** **void** encryptCopyDir(

File from, File to, **int** key) **throws** Exception {

**if**(! to.mkdirs()) {//新建文件夹，如果失败抛异常

**throw** **new** FileNotFoundException(

"路径不存在: "+to.getAbsolutePath());

}

File[] list = from.listFiles();

**if**(list == **null**) {

**return**;

}

**for** (File f : list) {

//目标文件夹 to 中的目标文件，保持原文件名

File toFile = **new** File(to, f.getName());

**if**(f.isFile()) {//f是文件

//加密复制单个文件

*encrypCopyFile*(f, toFile, key);

} **else** {//f是文件夹

//递归复制子目录

*encryptCopyDir*(f, toFile, key);

}

}

}

**private** **static** **void** encrypCopyFile(

File from, File to, **int** key) **throws** Exception {

FileInputStream in = **new** FileInputStream(from);

FileOutputStream out = **new** FileOutputStream(to);

**byte**[] buff = **new** **byte**[8192];

**int** n;

**while**((n = in.read(buff)) != -1) {

/\* 先把buff数组中的前n个字节值加密

\* 加密之后再向目标文件输出 \*/

**for** (**int** i = 0; i < n; i++) {

buff[i] ^= key;

}

out.write(buff, 0, n);

}

in.close();

out.close();

}

}

1. **随机抽奖**

d:/stu.txt

------------------

同学甲;6

同学乙;2

同学丙;4

同学丁;1

秀;3

项目：d1602\_随机抽奖

类：d1602.Test1

**package** d1602;

**import** java.io.BufferedReader;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.io.UnsupportedEncodingException;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.List;

**import** java.util.Random;

**public** **class** Test1 {

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

//读取加载文件数据

List<Student> list = *load*();

//随机访问list中的一个学生实例

Student s = *chouJiang*(list);

s.setCount(s.getCount()+1);

System.***out***.println(

"恭喜"+s.getName()+"中大奖");

//把列表数据，重新存到文件

*save*(list);

}

**private** **static** List<Student> load() **throws** Exception {

ArrayList<Student> list = **new** ArrayList<>();

BufferedReader in =

**new** BufferedReader(

**new** InputStreamReader(

**new** FileInputStream("d:/stu.txt"), "GBK"));

String line;

**while**((line = in.readLine()) != **null**) {

line = line.replaceAll("\\s+", "");

**if** (line.length() == 0) {

**continue**;

}

//同学甲;0 ---> ["同学甲", "0"]

// 0 1

String[] a = line.split(";");

Student s = **new** Student();

s.setName(a[0]);

s.setCount(Integer.*parseInt*(a[1]));

list.add(s);

}

in.close();

System.***out***.println(list);

**return** list;

}

**private** **static** Student chouJiang(

List<Student> list) {

//对list排序

Collections.*sort*(list, **new** Comparator<Student>() {

@Override

**public** **int** compare(Student o1, Student o2) {

**return** o1.getCount()-o2.getCount();

}

});

**int** mid = list.size()/2;

**int** j;

**if** (Math.*random*() < 0.7) {//70%概率在前面一半抽取

// [0, mid)

j = **new** Random().nextInt(mid);

} **else** {//30%概率在后面一半抽取

// mid + [0, list.size()-mid)

j =

mid+ **new** Random().nextInt(list.size()-mid);

}

**return** list.get(j);

}

**private** **static** **void** save(List<Student> list) **throws** Exception {

PrintWriter out =

**new** PrintWriter(

**new** OutputStreamWriter(

**new** FileOutputStream("d:/stu.txt"),"GBK"));

**for** (Student s : list) {

out.println(s);//默认调用 s.toString()

}

out.close();

}

}

1. **后台线程**

**package** d1504;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Scanner;

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

T1 t1 = **new** T1();

t1.start();

Thread t2 = **new** Thread() {

@Override

**public** **void** run() {

System.***out***.println("按回车捅醒 t1");

**new** Scanner(System.***in***).nextLine();

t1.interrupt();

}

};

//虚拟机不会等待后代线程结束

//所有前台线程结束时，虚拟机会自动退出

t2.setDaemon(**true**);

t2.start();

}

**static** **class** T1 **extends** Thread {

@Override

**public** **void** run() {

SimpleDateFormat f =

**new** SimpleDateFormat("HH:mm:ss");

**for**(**int** i=0; i<10; i++) {

String s = f.format(**new** Date());

System.***out***.println(s);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

System.***out***.println("谁捅醒了我");

**break**;

}

}

}

}

}

* 1. **多线程共享数据访问冲突**

一个线程修改数据，另一个线程同时访问数据，可能引起数据访问混乱

图片包含 图形用户界面

描述已自动生成

1. **数据访问冲突**

项目：d1603\_线程

类：d1603.Test1

**package** d1603;

**import** java.util.Arrays;

**public** **class** Test1 {

**static** **char**[] *a* = {'-','-','-','-','-'};

**static** **char** *c* = '\*';

**public** **static** **void** main(String[] args) {

Thread t1 = **new** Thread() {

@Override

**public** **void** run() {

**while**(**true**) {

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

*a*[i] = *c*;

}

*c* = (*c*=='\*'?'-':'\*');

}

}

};

Thread t2 = **new** Thread() {

@Override

**public** **void** run() {

**while**(**true**) {

System.***out***.println(

Arrays.*toString*(*a*));

}

}

};

t1.start();

t2.start();

}

}

* 1. **线程同步 synchronized**

让多个线程共享访问数据时，步调一致的执行。

一个线程修改时，其他线程等待修改完成后才能执行；

一个线程访问时，其他线程等待访问结束

任何实例，都有一个“同步锁”，synchronized 关键字，要求一个线程必须抢到同步锁才能执行

synchronized(对象) {

共享的数据访问代码

}

必须抢到指定对象的锁，才能执行

synchronized void f() {

}

抢当前实例（this）的锁

static synchronized void f() {

}

抢类的锁

1. **同步**

**package** d1603;

**import** java.util.Arrays;

**public** **class** Test1 {

**static** **char**[] *a* = {'-','-','-','-','-'};

**static** **char** *c* = '\*';

**public** **static** **void** main(String[] args) {

Thread t1 = **new** Thread() {

@Override

**public** **void** run() {

**while**(**true**) {

**synchronized** (*a*) {

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

*a*[i] = *c*;

}

}

*c* = (*c*=='\*'?'-':'\*');

}

}

};

Thread t2 = **new** Thread() {

@Override

**public** **void** run() {

**while**(**true**) {

**synchronized** (*a*) {

System.***out***.println(Arrays.*toString*(*a*));

}

}

}

};

t1.start();

t2.start();

}

}

1. **同步**

static int i=0

i++ i++ i++

i++ i++ i++

if(i%2==1) if(i%2==1) if(i%2==1)

Test2

应用程序

中度可信度描述已自动生成

**package** d1603;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

R1 r1 = **new** R1();

Thread t1 = **new** Thread(r1);

// r1.run(), r1.add(),r1.add(),r1.add(),r1.add(),r1.add(),

t1.start();

//main

// r1.get(), r1.get(),r1.get(),r1.get(),r1.get(),r1.get(),

R1 r2 = **new** R1();

**while**(**true**) {

**int** i = r2.*get*();

**if**(i % 2 == 1) {

System.***out***.println(i);

System.*exit*(0);//关闭虚拟机

}

}

}

**static** **class** R1 **implements** Runnable {

**static** **int** *i*=0;

**public** **static** **synchronized** **void** add() {

*i*++;

*i*++;

}

**public** **static** **synchronized** **int** get() {

**return** *i*;

}

@Override

**public** **void** run() {

**while**(**true**) {

*add*();

}

}

}

}

* 1. **生产者、消费者模型**

线程之间传递数据

徽标

中度可信度描述已自动生成

Stack

**package** d1603;

**public** **class** Stack {

**private** **char**[] a = **new** **char**[5];

**private** **int** index;

**public** **void** push(**char** c) {

**if**(index == 5) {

**return**;

}

a[index] = c;

index++;

}

**public** **char** pop() {

**if** (index == 0) {

**return** ' ';//用空格字符，表示没有数据

}

index--;

**char** c = a[index];

**return** c;

}

}

Producer

**package** d1603;

**import** java.util.Random;

**public** **class** Producer **extends** Thread {

**private** Stack stack;

**public** Producer(Stack stack) {

**this**.stack = stack;

}

@Override

**public** **void** run() {

**while**(**true**) {

// 'a' - 'z'

**char** c =

(**char**) ('a' + **new** Random().nextInt(26));

**synchronized** (stack) {

stack.push(c);

System.***out***.println("压入 << "+c);

}

}

}

}

Consumer

**package** d1603;

**public** **class** Consumer **extends** Thread {

**private** Stack stack;

**public** Consumer(Stack stack) {

**this**.stack = stack;

}

@Override

**public** **void** run() {

**while**(**true**) {

**synchronized** (stack) {

**char** c = stack.pop();

System.***out***.println("弹出 >> "+c);

}

}

}

}

Test3

**package** d1603;

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

Stack stack = **new** Stack();

Producer p = **new** Producer(stack);

Consumer c = **new** Consumer(stack);

p.start();

c.start();

}

}

* 1. **等待和通知**

Object 的方法

wait()

notify()

notifyAll()

* 必须在 synchronized 代码内调用
* 等待和通知的对象，必须是加锁的对象

图示

描述已自动生成

* 1. **同步监视器**

遇到 synchronized 关键字，会在加锁的对象上，关联一个同步监视器

图示

描述已自动生成

1. **网络**

win + r， 输入 cmd

输入命令：

ipconfig

图片包含 文本

描述已自动生成

ping 192.168.21.xxx

如果ping不通，要关闭防火墙

图片包含 文本

描述已自动生成

* 1. **Socket 通信**

Socket - 网络套接字

* 通过 ip 地址，两台主机可以互相找到对象
* 在两台主机上，各选择一个端口号
  1. **服务器端**

被动等待客户端发起连接

ServerSocket 在服务器端，选择一个端口号， 在指定端口上等待客户端发起连接

ServerSocket ss = new ServerSocket(8000);

等待客户端发起连接，建立连接通道，并返回连接通道的服务器端插头对象

Socket s = ss.accept();

* 1. **客户端**

主动去连接服务器

Socket s = new Socket(ip, port);

取出双向的流

InputStream in = s.getInputStream();

OutputStream out = s.getOutputStream();

* 1. **超时**

socket.setSoTimeout(毫秒值)

设置接收数据，等待超时时长，超时会出现 SocketTimeoutException

1. **网络**

项目：d1701\_网络

类：d1701.Server1

**package** d1701;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public** **class** Server1 {

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

ServerSocket ss = **new** ServerSocket(8000);

System.***out***.println(

"服务已经在 8000 端口上启动");

//暂停等待客户端发起连接

System.***out***.println("等待客户端连接");

Socket s = ss.accept();

System.***out***.println("客户端已连接");

//从插头对象，获取输入流和输出流

InputStream in = s.getInputStream();

OutputStream out = s.getOutputStream();

/\*

\* 通信协议，通信流程，数据格式

\*

\* 服务器端：

\* 1. 接收 "hello"

\* 2. 发送 "world"

\*/

**for** (**int** i = 0; i < 5; i++) {

**char** c = (**char**) in.read();

System.***out***.print(c);

}

out.write("world".getBytes());

out.flush(); //刷出内存缓存

s.close();//断开连接

ss.close();//停止服务，释放端口

}

}

Client1

**package** d1701;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.Socket;

**public** **class** Client1 {

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

Socket s = **new** Socket("旁边同学ip", 8000);

InputStream in = s.getInputStream();

OutputStream out = s.getOutputStream();

/\* 客户端：

\* 1. 发送 "hello"

\* 2. 接收 "world"

\*/

out.write("hello".getBytes());

out.flush(); //刷出内存缓存

**for** (**int** i = 0; i < 5; i++) {

**char** c = (**char**) in.read();

System.***out***.print(c);

}

s.close();

}

}

* 1. **阻塞操作**

**ss.accept()**

暂停，等待客户端发起连接

**in.read()**

对方不发送数据，暂停，死等数据

* 1. **服务器端的线程模型**

图示, 示意图

描述已自动生成

1. **回声**

客户端发送到服务器的数据，原封不动的发回客户端

项目：d1702\_回声

类：d1702.EchoServer

EchoClient

EchoServer

**package** d1702;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public** **class** EchoServer {

**public** **void** launch() {

**new** Thread() {

@Override

**public** **void** run() {

**try** {

ServerSocket ss = **new** ServerSocket(8000);

System.***out***.println("服务已在 8000 端口上启动");

**while**(**true**) {

System.***out***.println("等待下一个客户端连接");

Socket s = ss.accept();

System.***out***.println("客户端已连接");

TongXinThread t = **new** TongXinThread(s);

t.start();

}

} **catch** (Exception e) {

System.***out***.println(

"无法在 8000 端口上启动服务，或服务已停止");

}

}

}.start();

}

**class** TongXinThread **extends** Thread {

Socket s;

**public** TongXinThread(Socket s) {

**this**.s = s;

}

@Override

**public** **void** run() {

/\*

\* 通信协议，流程和格式

\* UTF-8编码的字符串，

\* 每段字符串末尾添加换行

\*

\* BR--ISR--网络输入流

\* PW--OSW--网络输出流

\*/

**try** {

BufferedReader in =

**new** BufferedReader(

**new** InputStreamReader(

s.getInputStream(), "UTF-8"));

PrintWriter out =

**new** PrintWriter(

**new** OutputStreamWriter(

s.getOutputStream(),"UTF-8"));

String line;

**while**((line = in.readLine()) != **null**) {

out.println(line);

out.flush();

}

//断开

} **catch** (Exception e) {

//断开

}

System.***out***.println("一个客户端端已断开");

}

}

**public** **static** **void** main(String[] args) {

EchoServer s = **new** EchoServer();

s.launch();

}

}

EchoClient

**package** d1702;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.io.UnsupportedEncodingException;

**import** java.net.Socket;

**import** java.util.Scanner;

**public** **class** EchoClient {

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

Socket s = **new** Socket("找一个人启动服务器", 8000);

BufferedReader in =

**new** BufferedReader(

**new** InputStreamReader(

s.getInputStream(), "UTF-8"));

PrintWriter out =

**new** PrintWriter(

**new** OutputStreamWriter(

s.getOutputStream(),"UTF-8"));

**while**(**true**) {

System.***out***.print("输入：");

String str = **new** Scanner(System.***in***).nextLine();

out.println(str);

out.flush();

String echo = in.readLine();

System.***out***.println("回声："+echo);

System.***out***.println("-----------------");

}

}

}

1. **聊天室**

项目：d1703\_聊天室

类：d1703.ChatServer

ChatClient

ChatServer

**package** d1703;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.util.ArrayList;

**public** **class** ChatServer {

**private** ArrayList<TongXinThread> list =

**new** ArrayList<>();

**public** **void** launch() {

//启动服务线程

**new** Thread() {

@Override

**public** **void** run() {

**try** {

ServerSocket ss = **new** ServerSocket(8000);

System.***out***.println("聊天室服务器已启动");

**while**(**true**) {

Socket s = ss.accept();

TongXinThread t =

**new** TongXinThread(s);

t.start();

}

} **catch** (Exception e) {

System.***out***.println(

"服务无法在 8000 端口上启动，或服务已经停止");

}

}

}.start();

}

**class** TongXinThread **extends** Thread {

Socket s;

BufferedReader in;

PrintWriter out;

**private** String name;

**public** TongXinThread(Socket s) {

**this**.s = s;

}

**public** **void** send(String msg) {

out.println(msg);

out.flush();

}

**public** **void** sendAll(String msg) {

**synchronized** (list) {

**for** (TongXinThread t : list) {

t.send(msg);

}

}

}

@Override

**public** **void** run() {

**try** {

//UTF-8, 换行

in = **new** BufferedReader(**new** InputStreamReader(s.getInputStream(), "UTF-8"));

out = **new** PrintWriter(**new** OutputStreamWriter(s.getOutputStream(),"UTF-8"));

//接收客户端的昵称

**this**.name = in.readLine();

//把当前通信线程实例，加入集合

**synchronized** (list) {

list.add(**this**);

}

//发送欢迎信息

send("欢迎进入激情聊天室");

//群发上线消息

sendAll(name+"进入了聊天室，在线人数："+list.size());

String line;

**while**((line = in.readLine()) != **null**) {

sendAll(name+"说："+line);

}

//断开

} **catch** (Exception e) {

//断开

}

//删除当前通信线程实例

**synchronized** (list) {

list.remove(**this**);

}

//群发离线消息

sendAll(name+"离开了聊天室，在线人数："+list.size());

}

}

**public** **static** **void** main(String[] args) {

ChatServer s = **new** ChatServer();

s.launch();

}

}

ChatClient

**package** d1703;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.net.Socket;

**import** java.util.LinkedList;

**import** java.util.Scanner;

**public** **class** ChatClient {

**private** Socket s;

**private** BufferedReader in;

**private** PrintWriter out;

**private** String name;

**private** LinkedList<String> list = **new** LinkedList<>();

**private** **boolean** flag; //开关

**public** **void** launch() {

**try** {

s = **new** Socket("192.168.21.61", 8000);

in = **new** BufferedReader(**new** InputStreamReader(s.getInputStream(), "UTF-8"));

out = **new** PrintWriter(**new** OutputStreamWriter(s.getOutputStream(),"UTF-8"));

//昵称

System.***out***.print("起一个昵称： ");

name = **new** Scanner(System.***in***).nextLine();

out.println(name);

out.flush();

//接收线程

**new** Thread() {

@Override

**public** **void** run() {

receive();

}

}.start();

//输入线程

**new** Thread() {

@Override

**public** **void** run() {

input();

}

}.start();

//打印线程

**new** Thread() {

@Override

**public** **void** run() {

print();

}

}.start();

} **catch** (Exception e) {

System.***out***.println("无法连接聊天室服务器");

e.printStackTrace();

}

}

**protected** **void** print() {

**while**(**true**) {

**synchronized** (list) {

**while** (list.isEmpty() || flag) {

**try** {

list.wait();

} **catch** (InterruptedException e) {

}

}

String msg = list.removeFirst();

System.***out***.println(msg);

}

}

}

**protected** **void** input() {

System.***out***.println("按回车输入聊天内容");

**while**(**true**) {

**new** Scanner(System.***in***).nextLine();

flag = **true**;//打开开关

System.***out***.print("输入聊天内容：");

String s = **new** Scanner(System.***in***).nextLine();

out.println(s);

out.flush();

flag = **false**;//关闭开关

//通知打印线程可以继续打印

**synchronized** (list) {

list.notify();

}

}

}

**protected** **void** receive() {

**try** {

String line;

**while**((line = in.readLine()) != **null**) {

**synchronized** (list) {

list.add(line);

//通知打印线程已经有数据可以打印了

list.notify();

}

}

} **catch** (Exception e) {

}

System.***out***.println("已经与服务器断开连接");

}

**public** **static** **void** main(String[] args) {

ChatClient c = **new** ChatClient();

c.launch();

}

}

图示, 示意图

描述已自动生成

1. **反射 Reflect**

config.txt

--------------------------

d1801.A;a

d1801.B;b

d1801.C;c

d1801.D;d

加载到方法区的“类对象”的反射操作，可以

* 获得一个类的定义信息
* 反射新建实例
* 反射调用成员
  1. **获取“类对象”**

Class.forName("d1801.A")

A.class

a1.getClass()

* 1. **获得包名、类名**

包名

c1.getPackage().getName()

完整类名

c1.getName()

类名，不含包名

c1.getSimpleName()

* 1. **成员变量定义信息**

**getFields()**

获得所有可见的成员变量，包括继承的变量

**getDeclaredFields()**

获得本类定义的成员变量，包括私有

不包括继承的变量

**getField(变量名)**

**getDeclaredField(变量名)**

* 1. **构造方法定义信息**

**getConstructors()**

获得所有可见的构造方法

**getDeclaredConstructors()**

获得本类定义的构造方法，包括私有

**getConstructor(参数类型列表)**

**getDeclaredConstructor(int.class, String.class)**

* 1. **方法定义信息**

**getMethods()**

获得所有可见的方法，包括继承的方法

**getDeclaredMethods()**

获得本类定义的方法，包括私有

不包括继承的方法

**getMethod(方法名,参数类型列表)**

**getDeclaredMethod(方法名, int.class, String.class)**

* 1. **反射新建实例**
     1. **新建实例时，执行无参构造**

Object obj = c.newInstance();

* + 1. **新建实例时，执行有参构造**

获取构造方法

Constructor t = c.getConstructor(int.class, String.class);

新建实例，并执行该构造方法

Object obj = t.newInstance(6, "abc");

* 1. **反射调用成员变量**

获取变量

Field f = c.getDeclaredField(变量名);

使私有成员允许访问

f.setAccessible(true);

反射给变量赋值

为指定实例的变量赋值，静态变量，第一参数给 null

f.set(实例, 值);

反射访问变量的值

访问指定实例的变量的值，静态变量，第一参数给 null

Object v = f.get(实例);

* 1. **反射调用成员方法**

获取方法

Method m = c.getDeclaredMethod(方法名, 参数类型列表);

使私有方法允许被调用

m.setAccessible(true)

反射调用方法

让指定的实例来执行该方法

Object returnValue = m.invoke(实例, 参数数据)

1. **反射**

项目：d1801\_反射

类：d1801.Test1

A

B

C

D

Test1

**package** d1801;

**public** **class** Test1 {

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

//A

Class<A> c1 = A.**class**;

//B

Class<B> c2 =

(Class<B>) Class.*forName*("d1801.B");

//C

C c = **new** C();

Class<C> c3 = (Class<C>) c.getClass();

System.***out***.println(c1);

System.***out***.println(c2);

System.***out***.println(c3);

}

}

Test2

**package** d1801;

**import** java.lang.reflect.Constructor;

**import** java.lang.reflect.Field;

**import** java.lang.reflect.Method;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** Test2 {

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

System.***out***.println("输入类名：");

String s = **new** Scanner(System.***in***).nextLine();

Class<?> c = Class.*forName*(s);

System.***out***.println(c.getPackage().getName());

System.***out***.println(c.getName());

System.***out***.println(c.getSimpleName());

/\*

\* d1801.A

\* java.lang.String

\* java.util.ArrayList

\* java.io.File

\*/

System.***out***.println("\n--成员变量------------------");

*f1*(c);

System.***out***.println("\n--构造方法------------------");

*f2*(c);

System.***out***.println("\n--方法------------------");

*f3*(c);

}

**private** **static** **void** f1(Class<?> c) {

// private static final int v1;

//Field实例，封装一个变量的定义信息

Field[] a = c.getDeclaredFields();

**for** (Field f : a) {

String t = f.getType().getSimpleName();

String n = f.getName();

System.***out***.println(t+" "+n);

}

}

**private** **static** **void** f2(Class<?> c) {

//public A(int i, String s) throws ...

//Constructor实例，封装构造方法的定义信息

Constructor<?>[] a =

c.getDeclaredConstructors();

**for** (Constructor<?> t : a) {

String n = c.getSimpleName();

Class<?>[] p = t.getParameterTypes();

System.***out***.println(

n+"("+ Arrays.*toString*(p) +")");

}

}

**private** **static** **void** f3(Class<?> c) {

//public static String f1(int i) throws ...

//Method实例，封装方法的定义信息

Method[] a = c.getDeclaredMethods();

**for** (Method t : a) {

String n = t.getName();

Class<?>[] p = t.getParameterTypes();

System.***out***.println(

n+"("+ Arrays.*toString*(p) +")");

}

}

}

Test3

**package** d1801;

**import** java.lang.reflect.Constructor;

**import** java.util.Scanner;

**public** **class** Test3 {

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

/\*

\* 反射新建两个实例

\* 1. 无参构造

\* 2. int参数构造

\*/

System.***out***.println("输入类名：");

String s = **new** Scanner(System.***in***).nextLine();

Class<?> c = Class.*forName*(s);

Object o1 = **null**;

Object o2 = **null**;

System.***out***.println("--执行无参构造------------");

**try** {

o1 = c.newInstance();

System.***out***.println(o1);

} **catch** (Exception e) {

System.***out***.println("不能执行无参构造");

}

System.***out***.println("--执行有参构造------------");

**try** {

Constructor<?> t = c.getConstructor(**int**.**class**);

o2 = t.newInstance(6);

System.***out***.println(o2);

} **catch** (Exception e) {

System.***out***.println("不能执行int参数构造方法");

}

}

}

Test4

**package** d1801;

**import** java.lang.reflect.Field;

**import** java.lang.reflect.Method;

**public** **class** Test4 {

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

A a = **new** A();

Class<A> c = (Class<A>) a.getClass();

Field f = c.getDeclaredField("i");

f.setAccessible(**true**);//私有可访问

f.set(a, 6);//反射赋值

System.***out***.println(a.getI());

**int** i = (**int**) f.get(a);//反射取值

System.***out***.println(i);

////

Method geti = c.getMethod("getI");

Method seti = c.getMethod("setI", **int**.**class**);

**int** r = (**int**) geti.invoke(a);

System.***out***.println(r);

seti.invoke(a, 666);

System.***out***.println(a.getI());

}

}

* 1. **反射用来做动态编程**

根据配置文件的配置来执行

1. **动态编程**

d:/config.txt

--------------------------

d1801.A;a

d1801.B;b

d1801.C;c

d1801.D;d

**package** d1801;

//运行器，根据配置文件来执行

**import** java.io.BufferedReader;

**import** java.io.FileInputStream;

**import** java.io.InputStreamReader;

**import** java.lang.reflect.Method;

**import** java.util.ArrayList;

**public** **class** Runner {

**private** **static** ArrayList<String> *list* =

**new** ArrayList<>();

**static** {

**try** {

BufferedReader in =

**new** BufferedReader(

**new** InputStreamReader(

**new** FileInputStream("d:/config.txt"), "GBK"));

String line;

**while**((line = in.readLine()) != **null**) {

line = line.replaceAll(" ", "");

**if**(line.length() == 0) {

**continue**;

}

*list*.add(line);

}

in.close();

System.***out***.println(*list*);

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

}

**public** **static** **void** launch() **throws** Exception {

**for** (String s : *list*) {

//"d1801.A;a" --> ["d1801.A", "a"]

// 0 1

String[] a = s.split(";");//拆分字符串

Class<?> c = Class.*forName*(a[0]);//获取“类对象”

Object obj = c.newInstance();//新建实例执行无参构造

Method m = c.getMethod(a[1]);//获取方法

m.invoke(obj);//反射调用该方法

}

}

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

Runner.*launch*();

}

}

1. **注解 annotation**

注解是为其他开发工具，或为其他 java 程序提供代码的额外信息

@Override

public String toString() {

}

@Override 编译器编译代码时，会根据该注解，检查方法定义是否是正确的方法重写语法

自定义注解，需要自己编写处理程序来处理注解

1. **注解**

定义 @Test 注解，

A类中的方法上添加 @Test 注解

@Test

void a() {}

void b() {}

@Test

void c() {}

Runner 运行器，从A类中，自动发现、运行有@Test注解的方法

项目：d1802\_注解

类：d1802.Test

A

Runner

Test

**package** d1802;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

/\*

\* 元注解：对注解的注解

\*

\* @Target 设置注解目标：类、方法、成员变量、参数变量...

\*

\* @Retention 保留范围：源码、字节码、运行期内存

\* 源码：编译成字节码时被丢弃

\* 字节码：类被加载到内存时丢弃

\* 运行期内存：在内存中保留

\*/

//@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD, ElementType.PARAMETER})

@Target(ElementType.***METHOD***)

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** Test {

// 如果没有默认值，使用注解时，就必须赋值

// 有默认值，可以不赋值

**int** id() **default** 0;

String title() **default** "";

/\*

\* 特殊的属性名 value，

\* 有特殊待遇，单独赋值时，可以不写value=

\* 可以作为一个有意义属性名的别名

\* @Test("sdfsdfs")

\* @Test(id=5, value="dfgsdf")

\* @Test(id=5, title="dfgsdf")

\*/

//title的别名

String value() **default** "";

}

A

**package** d1802;

**public** **class** A {

@Test(id=1, title="测试a")

**public** **void** a() {

System.***out***.println("A.a");

}

**public** **void** b() {

System.***out***.println("A.b");

}

@Test("测试b")

**public** **void** c() {

System.***out***.println("A.c");

}

@Test(id=3)

**public** **void** d() {

System.***out***.println("A.d");

}

}

Runner

**package** d1802;

**import** java.lang.reflect.InvocationTargetException;

**import** java.lang.reflect.Method;

**import** javax.swing.border.TitledBorder;

**public** **class** Runner {

//在指定的类中，自动发现自动运行有@Test注解的方法

**public** **static** **void** launch(Class c) **throws** Exception{

Object obj = c.newInstance();

Method[] a = c.getMethods();

**for** (Method m : a) {

//在方法上，是否存在Test类型的注解

**if** (m.isAnnotationPresent(Test.**class**)) {

//获取 Test 注解数据

Test t = m.getAnnotation(Test.**class**);

System.***out***.println("id: "+t.id());

System.***out***.println("title: "

+(t.title().length()!=0?t.title():t.value()));

m.invoke(obj);

System.***out***.println("\n\n-------------------");

}

}

}

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

Runner.*launch*(A.**class**);

}

}

1. **JUnit单元测试框架**

JUnit 单元测试框架是第三方的开源工具，需要下载开发包，引入到项目中，才能使用

JUnit 是Java单元测试的事实标准，eclipse也集成了 JUnit

1. **JUnit**

项目：d1803\_JUnit

类： d1803.Test1

右键点击项目 -- Build path -- add library/Other -- 选择 JUnit -- JUnit4

**package** d1803;

**import** org.junit.Test;

**public** **class** Test1 {

@Test

**public** **void** a() {

System.***out***.println("a");

}

**public** **void** b() {

System.***out***.println("b");

}

@Test

**public** **void** c() {

System.***out***.println("c");

}

@Test

**public** **void** d() {

System.***out***.println("d");

}

}