

12.11 java基础复习

一、 基础语法

1.基本知识

- 1. 关键字 ： 被java语言赋予特定含义的单词，全部小写。
- 2. 标识符

给类、接口、方法、变量起名字的字符序列
组成规则：英文大小写字母；数字；\$和_
数字不能开头；不能是关键字；区分大小写
常用： 包全小写，用.隔开。
类或接口，每个单词首字母大写。
方法、变量：首字母小写，后面单词首字母大写
常量：全大写

3. 注释

单行注释：//
多行注释：/* */
文档注释：/** */

4. 常量

字符串；整数；小数；字符；布尔；空；
进制：0b二进制；0开头八进制；默认十进制；0x十六进制

5. 变量

程序执行过程中，值在某个范围内可以改变的值。
没有初始化的变量无法使用。在一行上建议只定义一个变量。
定义格式：数据类型 变量名 = 初始化值

6. 数据类型

基本数据类型：

A:整数	占用字节数	
byte	1	
short	2	
int	4	
long	8	
B:浮点数		
float	4	加F或者f
double	8	长整数要加L或者1
C:字符		
char	2	
D:布尔		
boolean	1	

引用数据类型：
默认转换(从小到大的转换)：
byte,short,char-int-long-float-double
byte,short,char相互之间不转换，他们参与运算首先转换为int类型
(大到小的转换)强制转换：目标数据类型 变量 = (目标数据类型) (被转换的数据)；
问题：byte b1 = 3；
byte b2 = 4；
byte b3 = b1 + b2;//编译有问题，类型先做提升。
byte b4 = 3 + 4；
byte b = (byte)130； b = -126;byte的范围：-128~127

Java中的参数传递问题：

基本类型：形式参数的改变对实际参数没有影响。

引用类型：数组【】 形式参数的改变直接影响实际参数。 数组、类

7. 运算符

算数运算符: + - * /商 %取余 ++ --

赋值运算符: =, +=, -=, *=, /=, %= a-=b 等价于 a = a-b

这里面有强制转换。

比较运算符: ==, !=, >, >=, <, <= 最终结果是布尔类型。

==不能写成=;

逻辑运算符: &, |, ^, !, &&, ||; 用于连接boolean类型的式子

逻辑与&: 有false则false

逻辑或|: 有true则true

逻辑异或^: 相同则false, 不同则true。

情侣关系。

逻辑非!: 非true则false, 非false则true

&&: 结果和&是一样的, 只不过有短路效果。左边是false, 右边不执行。

||: 结果和|是一样的, 只不过有短路效果。左边是true, 右边不执行。

位运算符(知道下就行): 计算机的运算都是二进制补码进行运算。

&位与运算: 有0则0。

|位或运算: 有1则1。

^位异或运算: 相同则0, 不同则1。

~按位取反运算符: 0变1, 1变0

<< 把<<左边的数据乘以2的移动次幂

System.out.println(3 << 2); // 3*2^2 = 3*4 = 12;

>> 把>>左边的数据除以2的移动次幂

System.out.println(24 >> 2); // 24 / 2^2 = 24 / 4 = 6

三元格式符: 比较表达式?表达式1:表达式2;

首先计算比较表达式的值, 看是true还是false。

如果是true, 表达式1就是结果。

如果是false, 表达式2就是结果。

2. 流程控制语句

1. 选择语句

1. if语句

(1) 三种格式

A: 格式1

```
if(比较表达式) {  
    语句体;  
}
```

执行流程:

判断比较表达式的值, 看是true还是false

如果是true, 就执行语句体

如果是false, 就不执行语句体

B: 格式2

```
if(比较表达式) {  
    语句体1;  
}  
else {  
    语句体2;  
}
```

执行流程:

判断比较表达式的值, 看是true还是false

如果是true, 就执行语句体1

如果是false, 就执行语句体2

C: 格式3

```
if(比较表达式1) {  
    语句体1;  
}  
else if(比较表达式2){  
    语句体2;
```

```

    }
    ...
    else {
        语句体n+1;
    }

```

执行流程:

```

    判断比较表达式1的值, 看是true还是false
    如果是true, 就执行语句体1
    如果是false, 就继续判断比较表达式2的值, 看是true还是false
    如果是true, 就执行语句体2
    如果是false, 就继续判断比较表达式3的值, 看是true还是false
    ...
    如果都不满足, 就执行语句体n+1

```

(2) 注意事项

- A: 比较表达式无论简单还是复杂, 结果是boolean类型
- B: if语句控制的语句体如果是一条语句, 是可以省略大括号的; 如果是多条, 不能省略。
建议: 永远不要省略。
- C: 一般来说, 有左大括号, 就没有分号, 有分号, 就没有左大括号。
- D: else后面如果没有if, 是不会出现比较表达式的。
- E: 三种if语句其实都是一个语句, 只要有一个执行, 其他的就不再执行。

2. switch语句

```

switch(表达式) {
    case 值1:
        语句体1;
        break;
    case 值2:
        语句体2;
        break;
    ...
    default:
        语句体n+1;
        break;
}

```

格式解释:

```

switch选择结构。
表达式 取值可以是byte ,short ,int ,char。
JDK1.7以后可以是枚举和string。
break; 中断结束switch语句。
default: 默认值。所有值都不匹配时。等if语句的else。

```

注意: case的值得是常量。不能是变量。
case后面的值不能是重复的。
default可以省略。也可以出现在任意位置。
switch语句运行到break位置, 或者是程序末尾。

2. 循环语句

```

for(初始化语句; 判断条件语句; 控制条件语句){
    循环体语句;
}
while(判断条件语句) {
    循环体语句;
}

```

扩展格式:

```

初始化语句;
while(判断条件语句){
    循环体语句;
    控制条件语句;
}
do {
    循环体语句;
}while(判断条件语句);

```

扩展格式:

```

初始化语句;
do {

```

```
    循环体语句;
    控制条件语句;
}while(判断条件语句);
```

while循环和for循环的区别?

使用区别: 如果你想在循环结束后, 继续使用控制条件的那个变量, 用while循环, 否则用for循环。不知道用for循环。
因为变量及早的从内存中消失, 可以提高内存的使用效率。

如果是一个范围的, 用for循环非常明确。

如果是不明确要做多少次, 用while循环较为合适。

do - while 至少会执行一次循环。

for里面的变量只在该区域内存在, 区域外无法引用。

3. 控制跳转语句

(1)break:中断的意思

A:用在循环和switch语句中, 离开此应用场景无意义。

B:作用

a:跳出单层循环

b:跳出多层循环, 需要标签语句的配合

(2)continue:继续

A:用在循环中, 离开此应用场景无意义。

B:作用

a:跳出单层循环的一次, 可以继续下一次

(3)return:返回

A:用于结束方法的, 后面还会在继续讲解和使用。

B:一旦遇到return, 程序就不会在继续往后执行。

3. 方法

就是完成特定功能的代码块。注意: 在很多语言里面有函数的定义, 而在Java中, 函数被称为方法。

1.格式:

```
修饰符 返回值类型 方法名(参数类型 参数名1, 参数类型 参数名2...) {方法体语句;
    return 返回值;
}
```

2.两个明确:

返回值类型: 结果的数据类型

参数列表: 参数的个数及对应的数据类型

3.方法调用

A:有明确返回值的方法

a:单独调用, 没有意义

b:输出调用, 不是很好,

4.方法注意事项:

方法不调用不执行

方法之间是平级关系, 不能嵌套定义

方法定义的时候, 参数是用, 隔开

5.方法重载

在同一个类中, 方法名相同, 参数列表不同。与返回值无关。

参数列表不同: 参数的个数不同。

参数的对应的数据类型不同。

4.数组

是存储同一种数据类型多个元素的集合。可以看成是一个容器 既可以存储基本数据类型，也可以存储引用数据类型。

数组格式

```
数据类型 [] 数组名; 推荐使用
数据类型 数组名 [];
int [] array;    定义一个int类型的数组a变量
int array [];    定义一个int类型的a数组变量
```

数组初始化

要初始化才能使用。
为数组元素分配内存空间，并为每个数组元素赋值。
两种方式：
 动态：先指定数组长度，后分配初始值
 数据类型[] 数组名= new 数据类型[数组长度]
 int []a = new int[3];
 静态：先指定初始值，系统后决定数组长度。 一般常用!
 数据类型[] 数组名 = new 数据类型[]{元素1,元素2,...};
 简化：数据类型[] 数组名 = {1,2,3};
动态和静态不能同时使用。

获取数组中的元素

数组名【索引】，从0开始，最大为长度-1。

数组操作的两个常见小问题

数组索引越界异常。
 数组下标最大值为长度-1； 访问了不存在的索引。
 int []arr = {1,2,3};
 System.out.println(arr[3]);

空指针异常：
 数组已经不再指向对应内存。
 arr = null;
 System.out.println(arr[0]);

二维数组

一个元素为一维数组的数组。

格式1：
 数据类型[][] 变量名= new 数据类型[m][n];
 int [][]arr = new int [3][2];
 三个一维数组arr[]，一维数组存放的是地址，每个数组里面有2个元素；
 int []x,y[]? x是一维数组，y是二维数组

格式2：
 数据类型[][]变量名= new 数据类型[m][]; m必给不然找不到分配空间地址。
 一维数组的元素动态给出。
 arr[] 数组名; arr[][] 为数组里的元素;
格式3：数据类型[][]变量名=new 数据类型[][]{元素}{元素}; 静态初始化;
 int [][]arr = {{1,2}{3,4}{5,6,7}};

二、面向对象

1.概述

类：一个模板。是一组相关的属性和行为的集合。是一个抽象的概念。

面向对象思想

A: 首先分析有哪些类
B: 接着分析每个类应该有什么
C: 最后分析类与类的关系

面向过程：每一个功能的步骤。

面向对象：强调的是对象，由对象去调用功能。更关注结果。

面向对象开发：不断的创建对象，使用对象，指挥对象做事。

面向对象设计：管理和维护对象之间的关系。

面向对象特征：封装、继承、多态

成员变量：事物的属性。
和变量的定义时一样的格式，但是在类方法外
成员方法：事物的行为
和方法的定义一样，去掉static。

对象
是类的一个实例。

如何创建对象使用

格式：类名 对象引用变量 = new 类名();
对象引用变量和对象是不一样的。但大多情况，这种差异是可以忽略

使用成员变量？

对象名.变量名

使用成员方法？

对象名.方法(参数名：参数类型);

2.关键字

匿名对象

只调用一次，调用完，就被当成垃圾回收。可以当做实际参数使用。
new Student().show()

封装

隐藏实现细节，提供公共的访问方式 提高代码的复用性，安全性。
private是封装的一种体现，修饰成员变量
封装原则：
将不需要对外提供的内容都隐藏起来。
把属性隐藏，提供公共方法对其访问。

private

修饰成员变量和成员方法
特点：修饰后的成员只能在本类中访问
提供get、set方法

this

代表当前类的引用对象，该方法内部的this就代表那个对象

构造方法

作用：对对象的数据进行初始化

格式：

1. 方法名和类名相同
2. 没有返回值类型,void也没有
3. 没有返回值

注意事项

- 1: 如果我们没写构造方法，系统将提供一个默认的无参构造方法
- 2: 如果我们给出了构造方法，系统将不再提供默认构造方法
如果这个时候，我们要使用无参构造方法，就必须自己给出。

推荐：永远手动自己给出无参构造方法。

给成员变量赋值的方式

- 1: setXxx()
- 2: 带参构造方法

形式参数的问题：

基本类型：形式参数的改变不影响实际参数

引用类型：形式参数的改变直接影响实际参数

Student s = new Student()的创建过程？

- (1) 把Student.class文件加载到内存
- (2) 在栈内存为s开辟空间
- (3) 在堆内存为学生对象申请空间
- (4) 给学生的成员变量进行默认初始化。null, 0
- (5) 给学生的成员变量进行显示初始化。林青霞, 27
- (6) 通过构造方法给成员变量进行初始化。陈遵胜, 30
- (7) 对象构造完毕，把地址赋值给s变量

成员变量和局部变量的区别？

- (1) 在类中的位置不同
 - 成员变量：类中方法外
 - 局部变量：方法定义中或者方法声明上
- (2) 在内存中的位置不同
 - 成员变量：在堆中
 - 局部变量：在栈中
- (3) 生命周期不同
 - 成员变量：随着对象的创建而存在，随着对象的消失而消失
 - 局部变量：随着方法的调用而存在，随着方法的调用完毕而消失
- (4) 初始化值不同
 - 成员变量：有默认值
 - 局部变量：没有默认值，必须定义，赋值，然后才能使用

static关键字

- (1) 静态的意思。可以修饰成员变量和成员方法。
- (2) 静态的特点：
 - A: 随着类的加载而加载
 - B: 优先与对象存在
 - C: 被类的所有对象共享
 - 这其实也是我们判断该不该使用静态的依据。
 - 举例：饮水机和水杯的问题思考
 - D: 可以通过类名调用
 - 既可以通过对象名调用，也可以通过类名调用，建议通过类名调用。
- (3) 静态的内存图
 - 静态的内容在方法区的静态区
- (4) 静态的注意事项：
 - A: 在静态方法中没有this对象
 - B: 静态只能访问静态
- (5) 静态变量和成员变量的区别
 - A: 所属不同
 - 静态变量：属于类，类变量
 - 成员变量：属于对象，对象变量，实例变量
 - B: 内存位置不同
 - 静态变量：方法区的静态区
 - 成员变量：堆内存
 - C: 生命周期不同
 - 静态变量：静态变量是随着类的加载而加载，随着类的消失而消失
 - 成员变量：成员变量是随着对象的创建而存在，随着对象的消失而消失
 - D: 调用不同
 - 静态变量：可以通过对象名调用，也可以通过类名调用
 - 成员变量：只能通过对象名调用

代码块: {}

局部代码块：局部位置，用于限定变量的生命周期。

静态代码块：在类成员位置，在构造代码块前加static，对类进行初始化，只执行一次。

构造代码块：调用一次构造方法，就调用一次构造代码块。对对象初始化。

静态代码块 > 构造代码块 > 构造方法。

主方法先于类的构造方法输出。

3.继承

多个类中有相同属性和行为时，将这些内容抽取到单独一个类中，那么其他类无需重复定义，只要继承即可。

继承：避免重复的定义。提高代码的复用性。

还可以添加自己的心成员。

什么时候考虑？谁是什么的一种。B是A的一种或者A是B的一种。

```
public class 子类 extends 父类{}
```

好处：

提高代码复用性；提高代码的维护性；

让类与类之间产生联系，是多态的前提。

开发的原则：低耦合，高内聚

耦合:类与类的关系

内聚：自己完成某件事的能力

java不支持多继承

```
public class 子类 extends 父类{} 父类{//错误
```

但是支持多层继承

```
public class 子类1 extends 父类{}
```

```
public class 子类2 extends 子类1{}
```

注意事项：

只能继承非私有的成员。

子类只能通过super去访问父类的构造方法

不要为了部分功能去使用继承。

继承中成员变量的关系：

A:子类中的成员变量和父类中的成员变量名称不一样，。

B:子类中的成员变量和父类中的成员变量名称一样，

在子类方法中访问一个变量的查找顺序：

a:在子类方法的局部范围找，有就使用

b:在子类的成员范围找，有就使用

c:在父类的成员范围找，有就使用

d:如果还找不到，就报错。

this 和 super的区别：

this代表本类的引用；

super代表父类存储空间标识；（可以调用父类的成员变量、构造方法、方法）

应用场景与this相同。

调用成员变量：super.成员变量

调用构造方法：super(构造方法)

调用方法：super.成员方法。

子类中所有的构造方法都会默认访问父类中空参数的构造方法。

子类会继承父类中的数据，可能还会使用父类的数据。

所以，子类初始化之前，一定要先完成父类数据的初始化。

每一个构造方法的第一条语句默认都是：super();

如何父类中没有构造方法，该怎么办呢？会报错。

子类通过super去显示调用父类其他的带参的构造方法

子类通过this去调用本类的其他构造方法

本类其他构造也必须首先访问了父类构造

一定要注意：

super(...)或者this(...)必须出现在第一条语句山

否则，就会有父类数据的多次初始化

继承中成员方法的关系：

A:子类中的方法和父类中的方法声明不一样，

B:子类中的方法和父类中的方法声明一样，
通过子类对象调用方法：
a:先找子类中，看有没有这个方法，有就使用
b:再看父类中，有没有这个方法，有就使用
c:如果没有就报错。

方法重载和方法重写？

方法覆盖（重写）：
子类中出现和父类中方法声明一样的方法。
原因：
子类对象调用方法的时候：
先找子类本身，再找父类。

注意：A:父类中私有方法不能被重写
因为父类私有方法子类根本就无法继承
B:子类重写父类方法时，访问权限不能更低 `public private`
最好就一致
C:父类静态方法，子类也必须通过静态方法进行重写
其实这个算不上方法重写

子类重写父类方法的时候，最好声明一模一样。

方法重载：
本类中出现的方法，参数类型不同，与返回值无关。
静态方法也能被继承。 但不能被覆盖。

方法覆盖override：
在子类中，出现和父类中一模一样的方法声明的现象。

方法重载overload：
同一个类中，出现的方法名相同，参数列表不同的现象。
方法重载能改变返回值类型，因为它和返回值类型无关。

4.多态、接口

final

final,最终的意思，可以修饰类、方法、变量

final类不能被继承；
final方法不能被覆盖；
final变量是常量，变量不能被重新赋值

final可以修饰局部变量。
方法内部中，该变量不可以被改变；
方法生命上，
基本类型：值不能发生改变
引用类型：该类型的地址值不能改变，但是该内存储存的值可以改变的。

final修饰变量时机：
被final修饰的变量只能赋值一次
在对象构造方法完毕前

多态

多态：某个事物，不同时刻变现出不同状态；
水（固液气）

前提： 有继承关系
要有方法覆盖，不然没意义!!
要有父类引用子类： `父f = new 子();`

成员访问特点：
成员变量： 编译和运行都看左边，即父类。
构造方法： 创建子类对象时，访问父类的构造方法，对父类数据进行初始化。
成员方法： 编译看左边 父类，运行看右边 子类
静态修饰：编译看父类，运行看父类。

成员方法：存在方法覆盖，所以多态运行看子类。

这也是多态的弊端： 不能使用子类的特有功能

如何使用子类的特有功能：

1. 创建子类对象调用方法（但是太占用内存空间）
2. 把父类的引用强制转换为子类的引用（向下转型）

对象间的转型：

向上转型：Fu f = new Zi()
从子到父，父类引用指向子类对象

向下转型： Zi z = (Zi)f //要求f必须是能够转换为zi的。
从父到子 父类引用转为子类对象
容易出现类型转换异常，

抽象

java中没有方法体的方法应该定义为抽象方法。

类中如果有抽象方法，该类必须定义为抽象类。

抽象类的特点：

格式：

```
abstract class 类名 {}  
public abstract void eat();
```

抽象类中不一定有抽象方法，但有抽象方法的类必须定义为抽象类

抽象类不能实例化。有构造方法，但不能实例化。

作用是用于子类访问父类数据的初始化

抽象的子类：

如果不想重写抽象方法，该子类是一个抽象类

重写所有的抽象方法，子类就变成了一个具体的类

抽象类可以实例化是靠具体的子类实现的，通过多态。（要有方法覆盖）

abstract不能和哪些关键字共存

a:final 冲突
b:private 冲突
c:static 无意义

接口

(1)为一开始的类添加个后期培养的功能，java提供了接口表示。

(2)接口的特点：

A:接口用关键字interface修饰

```
interface 接口名 {}
```

B:类实现接口用implements修饰

```
class 类名 implements 接口名 {}
```

C:接口不能实例化,按照多态的方式，由具体的子类实例化。其实这也是多态的一种，接口多态。

D:接口的实现类

a:是一个抽象类。

b:是一个具体类，这个类必须重写接口中的所有抽象方法。

(3)接口的成员特点：

A:成员变量

只能是常量

默认修饰符: public static final

B:构造方法

没有构造方法

C:成员方法

只能是抽象的

默认修饰符: public abstract 自己手动给出

(4)一些关系：

类与类：继承，不能多继承，可以多层继承

类与接口：实现关系，可以边继承边同时实现多个接口

接口与接口：继承关系，可以单继承，也可以多继承

抽象类和接口的区别：

A:成员区别

抽象类：

成员变量：可以变量，也可以常量
构造方法：有
成员方法：可以抽象，也可以非抽象
接口：
成员变量：只可以常量
成员方法：只可以抽象

抽象类 被继承体现的是：“is a”的关系。抽象类中定义的是该继承体系的共性功能。
接口 被实现体现的是：“like a”的关系。接口中定义的是该继承体系的扩展功能。

5.匿名内部类

导包

- (1)我们多次使用一个带包的类，非常的麻烦，这个时候，Java就提供了一个关键字import。
- (2)格式：
import 包名...类名;
另一种：
import 包名...*; (不建议)
- (3)package,import,class的顺序
package > import > class
- (4)包的定义
package 包名;
- 多级包用.分开。
- (5)注意事项：
A:package语句必须在文件中的第一条有效语句
B:在一个java文件中，只能有一个package
C:如果没有package，默认就是无包名

权限修饰符

- (1)权限修饰符
- | | 本类 | 同一个包下 | 不同包下的子类 | 不同包下的无关类 |
|-----------|----|-------|---------|----------|
| private | Y | | | |
| 默认 | Y | Y | | |
| protected | Y | Y | Y | |
| public | Y | Y | Y | Y |
- (2)这四种权限修饰符在任意时刻只能出现一种。
- ```
public class Demo {}
```

### 内部类

- (1)把类定义在另一个类的内部，该类就被称为内部类。  
举例：把类B定义在类A中，类B就被称为内部类。
- (2)内部类的访问规则  
A:可以直接访问外部类的成员，包括私有  
B:外部类要想访问内部类成员，必须创建对象
- (3)内部类的分类  
A:成员内部类  
B:局部内部类
- (4)成员内部类  
A:private 为了数据的安全性  
B:static 为了访问的方便性
- 成员内部类不是静态的：  
外部类名.内部类名 对象名 = new 外部类名.new 内部类名();  
成员内部类是静态的：  
外部类名.内部类名 对象名 = new 外部类名.内部类名();
- (5)成员内部类的面试题(填空)
- ```
30,20,10
```
- ```
class Outer {
 public int num = 10;
```

```

class Inner {
 public int num = 20;

 public void show() {
 int num = 30;

 System.out.println(num);
 System.out.println(this.num);
 System.out.println(Outer.this.num);
 }
}

```

#### (6)局部内部类

A:局部内部类访问局部变量必须加final修饰。

B:为什么呢?

因为局部变量使用完毕就消失，而堆内存的数据并不会立即消失。

所以，堆内存还是用该变量，而改变量已经没有了。

为了让该值还存在，就加final修饰。

通过反编译工具我们看到了，加入final后，堆内存直接存储的是值，而不是变量名。

#### (7)匿名内部类

A:是局部内部类的简化形式

B:前提

存在一个类或者接口

C:格式:

```

new 类名或者接口名() {
 重写方法;
}

```

D:本质:

其实是继承该类或者实现接口的子类匿名对象

#### (8)匿名内部类在开发中的使用

我们在开发的时候，会看到抽象类，或者接口作为参数。

而这个时候，我们知道实际需要的是一个子类对象。

如果该方法仅仅调用一次，我们就可以使用匿名内部类的格式简化。

```

interface Person {
 public abstract void study();
}

class PersonDemo {
 public void method(Person p) {
 p.study();
 }
}

class PersonTest {
 public static void main(String[] args) {
 PersonDemo pd = new PersonDemo();
 pd.method(new Person() {
 public void study() {
 System.out.println("好好学习，天天向上");
 }
 });
 }
}

```

#### 形式参数和返回值的问题

##### (1)形式参数:

类名: 需要该类的对象

抽象类名: 需要该类的子类对象

接口名: 需要该接口的实现类对象

##### (2)返回值类型:

类名: 返回的是该类的对象

抽象类名: 返回的是该类的子类对象

接口名：返回的是该接口的实现类的对象

(3)链式编程

对象.方法1().方法2().....方法n());

这种用法：其实在方法1()调用完毕后，应该返回一个对象；

方法2()调用完毕后，应该返回一个对象。

方法n()调用完毕后，可能是对象，也可以不是对象。

### 三、eclipse

- 1. 免费
- 2. 纯java语言编写
- 3. 免安装，绿色版
- 4. 扩展性强

src 源包  
bin编译的文件夹  
如何运行： ctrl + F11 run运行  
run as java aplication  
红色波浪线： 错误  
黄色波浪线： 警告  
Window--Show View—Console 找视窗  
修改代码区的字体大小颜色 window -- Preferences -- General -- Appearance -- Colors And Fonts -- Java修改 -- Java Edit Text Font  
行号的显示和隐藏：  
显示：在代码区域的最左边的空白区域，右键 -- Show Line Numbers即可。 隐藏：把上面的动作再做一次。

#### 快捷键

alt + / 内容辅助键 main + alt + / main方法  
ctrl + shift + f 代码格式化  
crtl + shift + o 导包  
注释 选中内容 ctrl + shift +/  
ctrl + /  
代码移动 选中代码alt+上下箭头  
看源码：选中类名，F3看源码。或者ctrl+点击  
alt + shift + s+对应的下划线字母 c无参，o带参自动生成构造方法，r为get()set()方法

#### jar包

就是多个class文件的压缩包  
用别人写好的东西  
如何打jar包？

选中项目-右键-Export-Java-jar-自己制定一个路径和名称-finish

如何做帮助文档？

针对源程序添加文档注释

选中项目-右键-export-java-javadoc-finish

使用？

复制jar包到项目路径并添加至构建路径中

#### eclipse中源码查看问题

attach source

external location

找到jdk安装目录文件夹中的src.zip文件。选中确认。

F3查看，或者选中类名ctrl+点击

## 删除项目

选中项目 – 右键 – 删除 从项目区域中删除；从硬盘上删除

## 导入项目

项目区域右键找import，general展开，找existing projects into workspace-next 选中要导入的项目（名称）

不能新建或导入同名的项目，随意建议的文件夹不能作为项目。

修改项目名得改配置.project 里面的名字。

## debug

双击有效程序行，设置断点。

debug as 调试

f6一行行运行调试

如何去断点：debug视图variables界面里的breakpoints 点击双叉

## 四、常用类

### Object

Object类是类结构层次的根类，每个类都直接或间接的继承Object

#### 1. Object类的方法：

public int hashCode():

返回该对象的哈希码值。根据地址值换算出来的

public final Class getClass():

返回此 Object 的运行时常类

Class类中的方法：public String getName()以 String 的形式返回此 Class 对象所表示的实体

public String toString():

返回该对象的字符串表示。默认是由类的全路径+'@'+哈希值的十六进制表示。

建议重写该方法: return 你想要的字符串。

也可以通过快捷键，自动生成toString

直接输出一个对象的名称，就是调用toString方法

public boolean equals(Object obj)

指示其他某个对象是否与此对象“相等”。默认比较的是地址。

this - s1 obj -s2

引用类型：这里的对象指的是地址。

重写？

一般都是用来比较对象的成员变量是否相同。

最终版alt + shift + s + h 重写hashCode和equal

protected void finalize()

当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。

子类重写 finalize 方法，以配置系统资源或执行其他清除。

protected Object clone()

创建并返回此对象的一个副本

标记接口

cloneable: 此类实现了cloneable接口，指示object.clone()方法可以合法地对该类实例进行字段复制。

克隆后，更改值不影响原来的值。

## 2. ==和equals()的区别？

A:==  
基本类型：比较的是值是否相同  
引用类型：比较的是地址值是否相同  
B:equals()  
只能比较引用类型。默认情况下，比较的是地址值是否相同。  
但是，我们可以根据自己的需要重写该方法。

## Scanner

```
Scanner sc = new Scanner(System.in)
has nextxxx 判断是否为某种类型
nextxxx 返回某种类型
public int nextInt()
public String nextLine()
```

## String

字符串是常量，一旦被赋值，就不能改变。

### 1. 构造方法：

```
常用 : String = "hello";
public String() 无参构造
public String(byte[] bytes) 把字节数组转换为字符串
public String(byte[] bytes,int offset,int length) 把字节数组的一部分转换为字符串
public String(char[] value) 把字符数组转换为字符串
public String(char[] value,int offset,int count) 把字符数组的一部分转换为字符串
public String(String original) 将字符常量转换为字符串
```

### 2. 判断功能

```
boolean equals(Object obj) 比较字符串内容是否相同
boolean equalsIgnoreCase(String str) 比较字符串内容是否相同，忽略大小写
boolean contains(String str) 判断大串里面是否含有小串
boolean startsWith(String str) 判断是否以某个字符串开头
boolean endsWith(String str) 判断是否以某个字符串结尾
boolean isEmpty() 判断字符串是否为空
```

### 3. 获取功能

```
int length() 获取字符串长度
char charAt(int index) 获取索引位置上的字符
int indexOf(int ch) 获取字符第一次出现的索引位置
int indexOf(String str) 获取字符串第一次出现的索引位置
int indexOf(int ch,int fromIndex) 从该位置开始搜索字符，并返回位置
int indexOf(String str,int fromIndex) 从该位置开始搜索字符串，并返回位置
String substring(int start) 从开始位置开始截取，并返回一个新的字符串
String substring(int start,int end) 返回一个新的字符串，它是此字符串的一个子字符串，包括start,不包括end
```

### 4. 转换功能

```
byte[] getBytes() 将String转换成byte，并存储到byte数组内
char[] toCharArray() 将String转化为char数组
static String valueOf(char[] chs) 字符数组转换为字符串
static String valueOf(int i) int类型转化为字符串
String toLowerCase() 产生一个新字符串，字符串转化为小写的
String toUpperCase() 产生一个新字符串，字符串转化为大写的
String concat(String str) 字符串连接
```

## 5. 其他功能

```
a:替换功能
 String replace(char old,char new) 字符替换
 String replace(String old,String new) 字符串替换
b:去空格功能
 String trim() 去除字符串两端空格
c:按字典比较功能 按ASCII表去比较顺序
 int compareTo(String str)
 int compareToIgnoreCase(String str) 忽略大小写比较
```

## String buffer

### 线程安全的可变字符串

每次String都会创建新对象，耗时，耗空间，StringBuffer可以解决这个问题。  
线程安全：线程同步 医院、银行网站  
不安全：不同步-效率高些 互联网网站、论坛

#### 1. toString():任何类型都可以通过这个方法，转换为字符串类型

```
String -- StringBuffer
 构造方法
StringBuffer -- String
 toString()方法
```

#### 2. 构造方法

```
public StringBuffer(int capacity) 指定容量值的缓冲区，默认为16
public StringBuffer() 造了一个缓冲区
public StringBuffer(String str) 指定字符创内容的缓冲区
length为实际值，
```

#### 3. 添加功能

```
public StringBuffer append() 添加任意类型到字符创缓冲区，并返回字符串缓冲区本身。可以链式编程
public StringBuffer insert(int offset,String str) 在指定位置后添加数据，并返回字符串本身
```

#### 4. 删除功能

```
public StringBuffer deleteCharAt(int index) 删除指定位置上的字符
public StringBuffer delete(int start,int end) 移除次序列的字符创中字符，不包括结束位置
StringBuffer delete(0,s.length());移除所有的数据
```

#### 5. 替换功能

```
public StringBuffer replace(int start,int end,String str) 用字符创替换指定位置。
```

#### 6. 反转功能

```
public StringBuffer reverse() 反转字符串
```

#### 7. 截取功能

```
返回值是String类型
public String substring(int start) 从指定位置截取
public String substring(int start,int end) 从指定位置上开始截取到截止位置，不包括截止位置。
```

判断是否为回文串：return new StringBuffer(s).reverse().toString().equals(s);

String、StringBuffer、StringBuilder区别？



String内容不可变，后面两者可变。  
StringBuffer：同步的，数据安全，效率低。  
StringBuilder：不同步的，数据不安全，效率高。

## StringBuffer和数组的区别？

都是容器，用来装其他的数据  
但是StringBuffer的数据最终是一个字符串数据  
数组的数据可以是多种，但最后是同一种的。

## 形式参数问题？

String、StringBuffer  
String作为形参，效果和基本类型相似。字符串是常量值。  
StringBuffer是引用类型。赋值时，不改变外部的。做操作时，会改变

## Arrays工具类

### 1. 排序

冒泡排序；快速排序；选择排序

public static void sort(数组):底层是快速排序  
Arrays.sort()

A:冒泡排序

相邻元素两两比较，大的往后放，第一次完毕，最大值出现在了最大索引处。同理，其他的元素就可以排好。

```
public static void bubbleSort(int[] arr) {
 for(int x=0; x<arr.length-1; x++) {
 for(int y=0; y<arr.length-1-x; y++) {
 if(arr[y] > arr[y+1]) {
 int temp = arr[y];
 arr[y] = arr[y+1];
 arr[y+1] = temp;
 }
 }
 }
}
```

B:选择排序

把0索引的元素，和索引1以后的元素都进行比较，第一次完毕，最小值出现在了0索引。同理，其他的元素就可以排好。

```
public static void selectSort(int[] arr) {
 for(int x=0; x<arr.length-1; x++) {
 for(int y=x+1; y<arr.length; y++) {
 if(arr[y] < arr[x]) {
 int temp = arr[x];
 arr[x] = arr[y];
 arr[y] = temp;
 }
 }
 }
}
```

### 2. 查找

基本查找；针对数组无序的情况

二分查找；针对数组有序的情况

public static int binarySearch(数组,key):  
Arrays.binarySearch();

A:基本查找

针对数组无序的情况

```
public static int getIndex(int[] arr,int value) {
 int index = -1;
```

```

 for(int x=0; x<arr.length; x++) {
 if(arr[x] == value) {
 index = x;
 break;
 }
 }

 return index;
 }
}

```

#### B:二分查找(折半查找)

针对数组有序的情况(千万不要先排序, 在查找)

```

public static int binarySearch(int[] arr,int value) {
 int min = 0;
 int max = arr.length-1;
 int mid = (min+max)/2;

 while(arr[mid] != value) {
 if(arr[mid] > value) {
 max = mid - 1;
 }else if(arr[mid] < value) {
 min = mid + 1;
 }

 if(min > max) {
 return -1;
 }

 mid = (min+max)/2;
 }

 return mid;
}

```

#### 转换成字符串

```

public static String toString():
 Arrays.toString(arr1)

```

## Integer

1. 为了让基本类型的数据进行更多的操作, Java就为每种基本类型提供了对应的包装类类型。可以在对象中定义更多的功能方法操作数据。

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

Integer类中有进制转换方法: 直接通过方法调用, 该指定的整数转换为对应进制。

static int parseInt(String s, int radix) 其余进制转为指定进制。这个方法也是将字符串转换为Integer类型。

static String toString(int i, int radix) 返回指定进制的整数, 进制范围2~36。0-9,a-z总共36个

#### 2. Integer的构造方法

```

Integer i = new Integer(int a);
Integer i = new Integer(string str); 这个字符必须是数字

```

#### 3. String和int的相互转换

```

A:String -- int

```

```
Integer.parseInt("100");
B:int -- String
String.valueOf(100);
```

#### 4. JDK5的新特性

```
Integer x = new Integer(4);可以直接写成
Integer x = 4;//自动装箱。
x = x + 5;//自动拆箱。通过Integer.valueOf方法。
```

自动装箱      基本类型--引用类型  
自动拆箱      引用类型--基本类型

Integer类有个缓冲池。针对-128到127之间的数据，做了一个数据缓冲池，如果数据是该范围内的，每次并不创建新的空间

注意：Integer的数据直接赋值，如果在-128到127之间，会直接从缓冲池里获取数据

#### Character

##### 构造方法

```
Character ch = new Character('a'); 类在对象中包装一个基本类型char的值。
```

##### 方法

```
public static boolean isUpperCase(char ch) 字符是否为大写
public static boolean isLowerCase(char ch) 是否为小写
public static boolean isDigit(char ch) 是否为数字
public static char toUpperCase(char ch) 字符转成大写
public static char toLowerCase(char ch) 字符转成小写
```