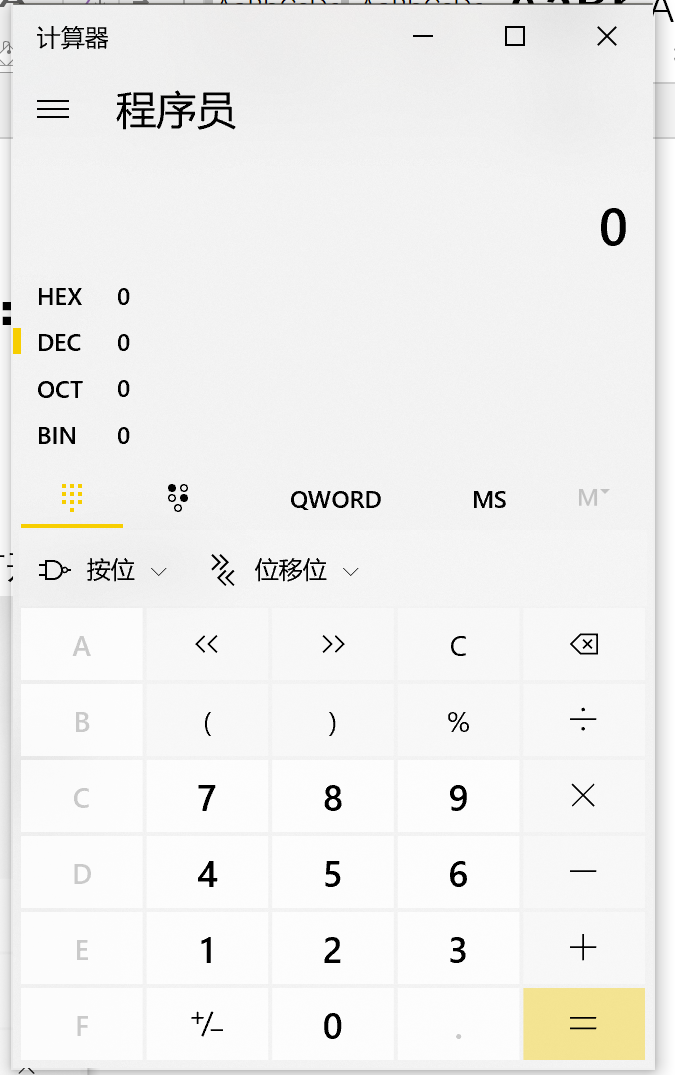
# 黑马Java

## Day 01

### 计算机基础知识：

### 进制转换

电脑自带的计算，可以打开程序员模式，HEX：16进制，DEC：十进制，OCT：八进制，BIN：2进制，可以在各个进制进行切换和计算。



### 常用DOS命令：

1. 如何在当前目录打开CMD

长按Shift，鼠标右键点击当前目录空白处，先打开Powershell窗口，输入 start cmd 回车

1. 

### HelloWorld小程序编译执行

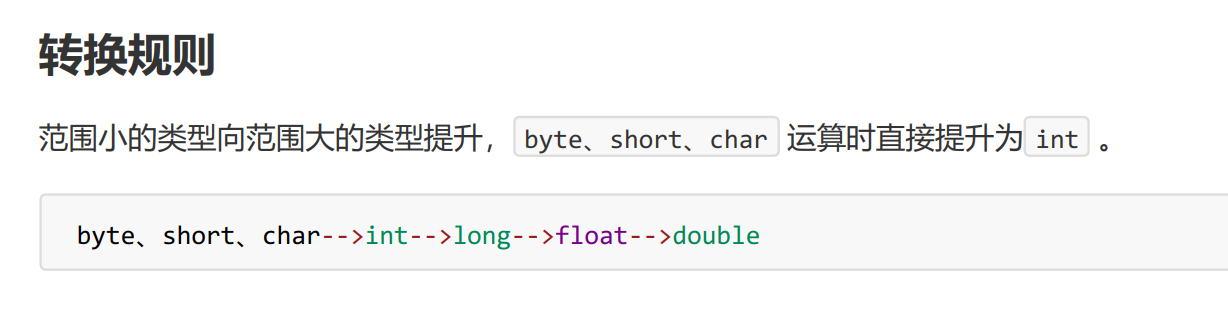
* 新建HelloWorld小程序，文件为HelloWorld.java
* 在当前目录打开cmd，运行javac HelloWorld.java，命令行无提示，但是在当前目录下生成了HelloWorld.class文件，该文件该文件就是编译后的文件，是Java的可运行文件，称为字节码文件，有了字节码文件，就可以运行程序了。
* cmd中使用命令 java HelloWorld 运行该程序

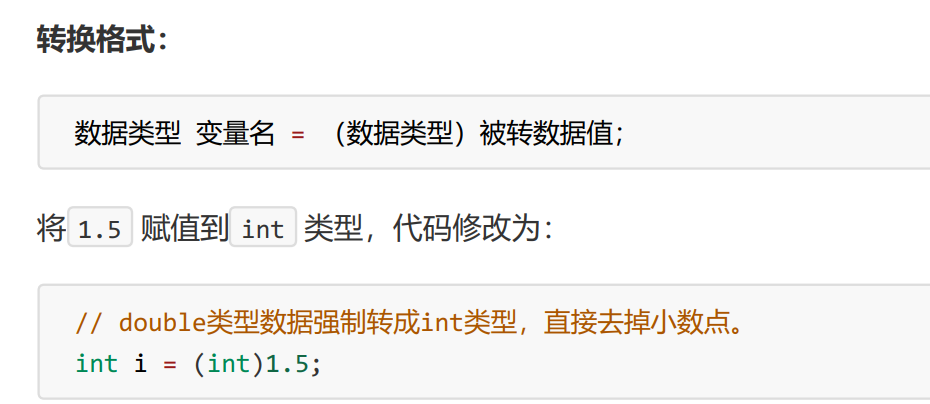
## Day 02

### 数据类型转换：

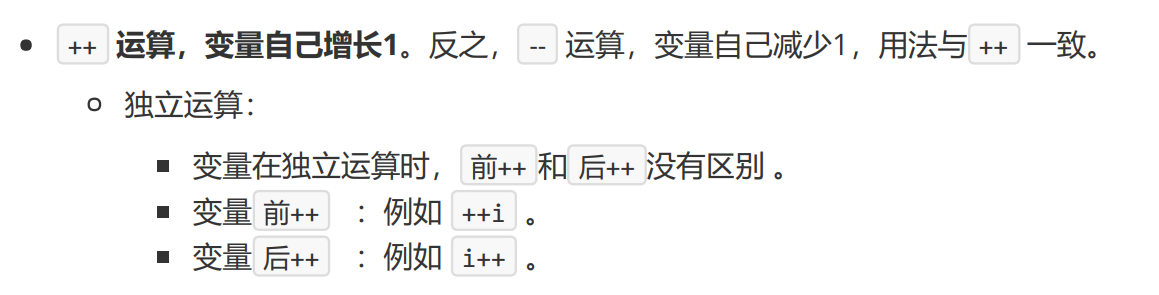
Java程序中要求参与的计算的数据，必须要保证数据类型的一致性，如果数据类型不一致将发生类型的转换

自动转换：将取值范围小的类型自动提升为取值范围大的类型 。

强制转换：将取值范围大的类型强制转换成取值范围小的类型 。（手工）



### ++运算和--运算





## Day 03

### 循环语句：

switch语句使用的注意事项：

1. 多个case后面的数值不可以重复。

2. switch后面小括号当中只能是下列数据类型：

基本数据类型：byte/short/char/int

引用数据类型：String字符串、enum枚举

3. switch语句格式可以很灵活：前后顺序可以颠倒，而且break语句还可以省略。

“匹配哪一个case就从哪一个位置向下执行，直到遇到了break或者整体结束为止。”

## Day 04

### IDEA快捷键：

自动生成代码：

* psvm 回车: public static void main(String[] args){}
* sout 回车：System.out.println();
* for循环代码生成：循环次数.for 回车，例如

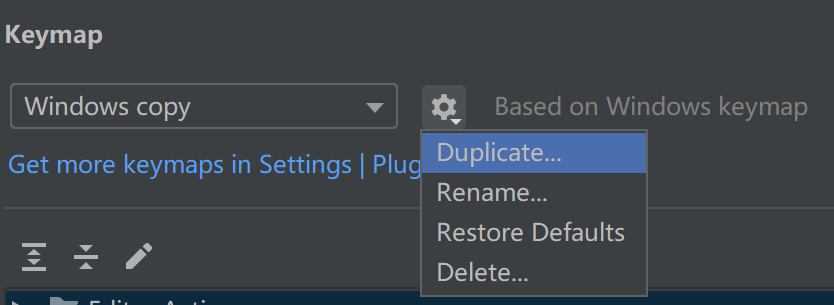
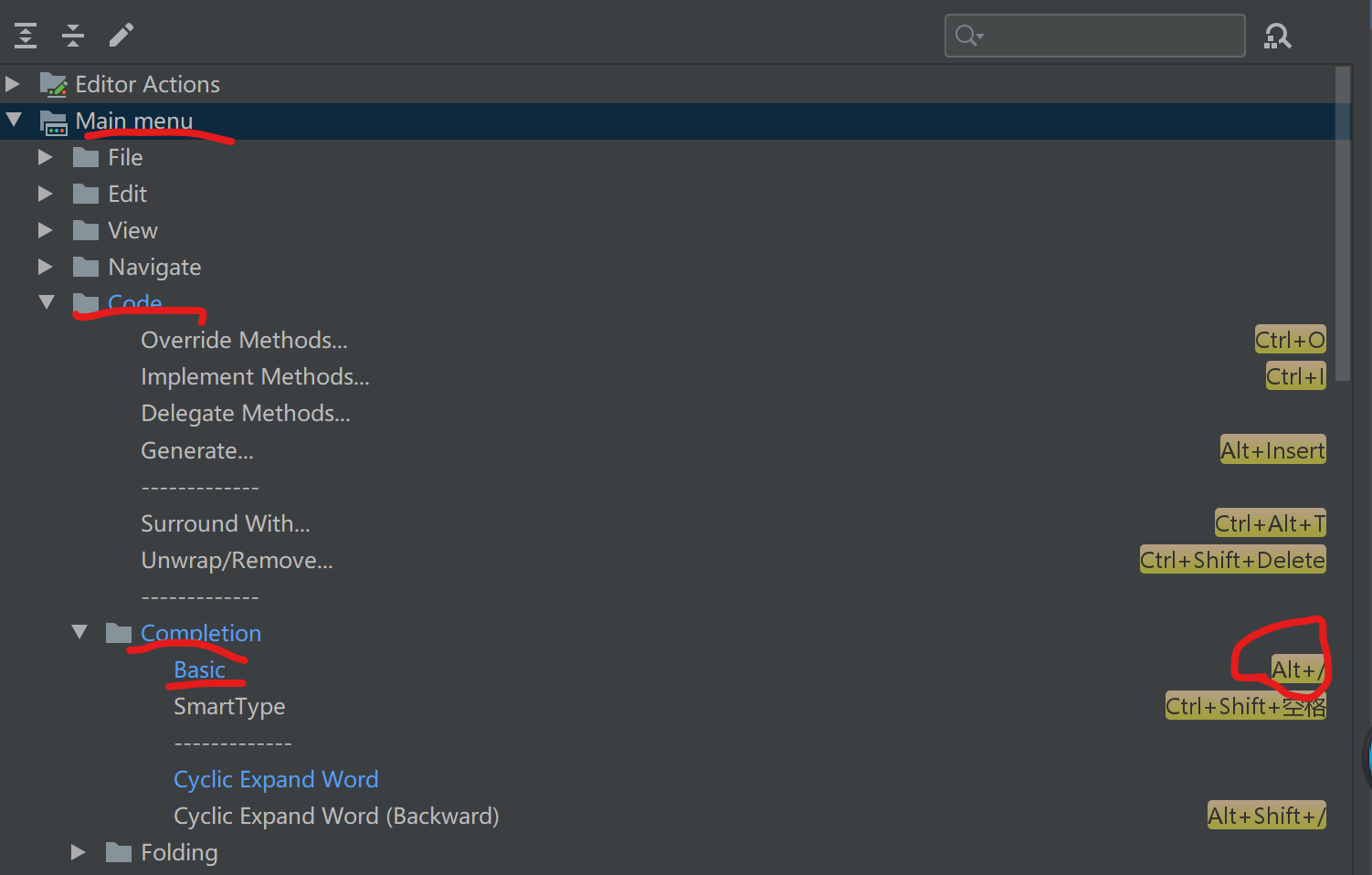
5.for 回车：for (int i = 0; i < 5; i++) { }

Alt 4：调出java编译执行结果界面

更改某些设置：

字体：settings🡪Eitor🡪font

快捷键补全设置：settings🡪Keymap🡪Duplicate🡪Main meun🡪Code🡪Comption🡪Basic



Alt+Enter:将光标放在报错位置，Alt+Enter，将会提出一些改错方式，尝试进行错误修复

Ctrl+Alt+L：自动对齐代码

同一项目下新建Module

### 定义方法的格式 ：



定义位置，类中方法外面。

返回值类型，必须要和 return 语句返回的类型相同，否则编译失败 。

### 调用方法格式



System.out.print("XXX"); 打印XXX

System.out.println("XXX"); 打印XXX并换行

### 方法重载：

方法重载：指在同一个类中，允许存在一个以上的同名方法，只要它们的参数列表不同即可，与修饰符和返回值类型无关。

参数列表：个数不同，数据类型不同，顺序不同。

重载方法调用：JVM通过方法的参数列表，调用不同的方法。

## Day 05

### 数组定义格式：

* 数组存储的数据类型[] 数组名字 = new 数组存储的数据类型[长度]；

int[] arr = new int[3];

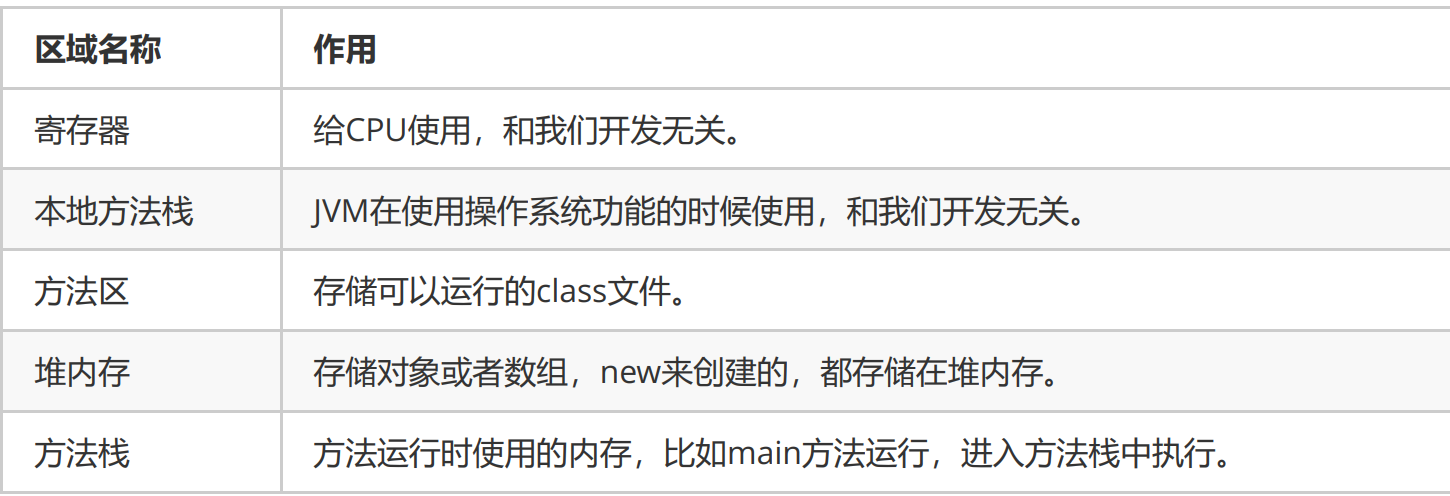
* 数据类型[] 数组名 = new 数据类型[]{元素1,元素2,元素3...};

int[] arr = new int[]{1,2,3,4,5};

* 数据类型[] 数组名 = {元素1,元素2,元素3...};

int[] arr = {1,2,3,4,5};

### JVM内存划分



## Day 06

### 类与对象

* 类：是一组相关属性和行为的集合。可以看成是一类事物的模板，使用事物的属性特征和行为特征来描述该类事物。现实中，描述一类事物：
* 属性：就是该事物的状态信息。
* 行为：就是该事物能够做什么。
* 对象：是一类事物的具体体现。对象是类的一个实例（对象并不是找个女朋友），必然具备该类事物的属性和行为。

现实中，一类事物的一个实例

#### 类的定义



#### 类的使用

通常，一个类不能直接使用，需要根据类创建一个对象。使用步骤如下：

1. 导包：指出所需要用的类的位置（同一个包下可以省略导包步骤）

import 包名称.类名称

1. 创建对象格式：

类名 对象名 = new 类名();（new 之后，没有赋值的成员变量为初始值）

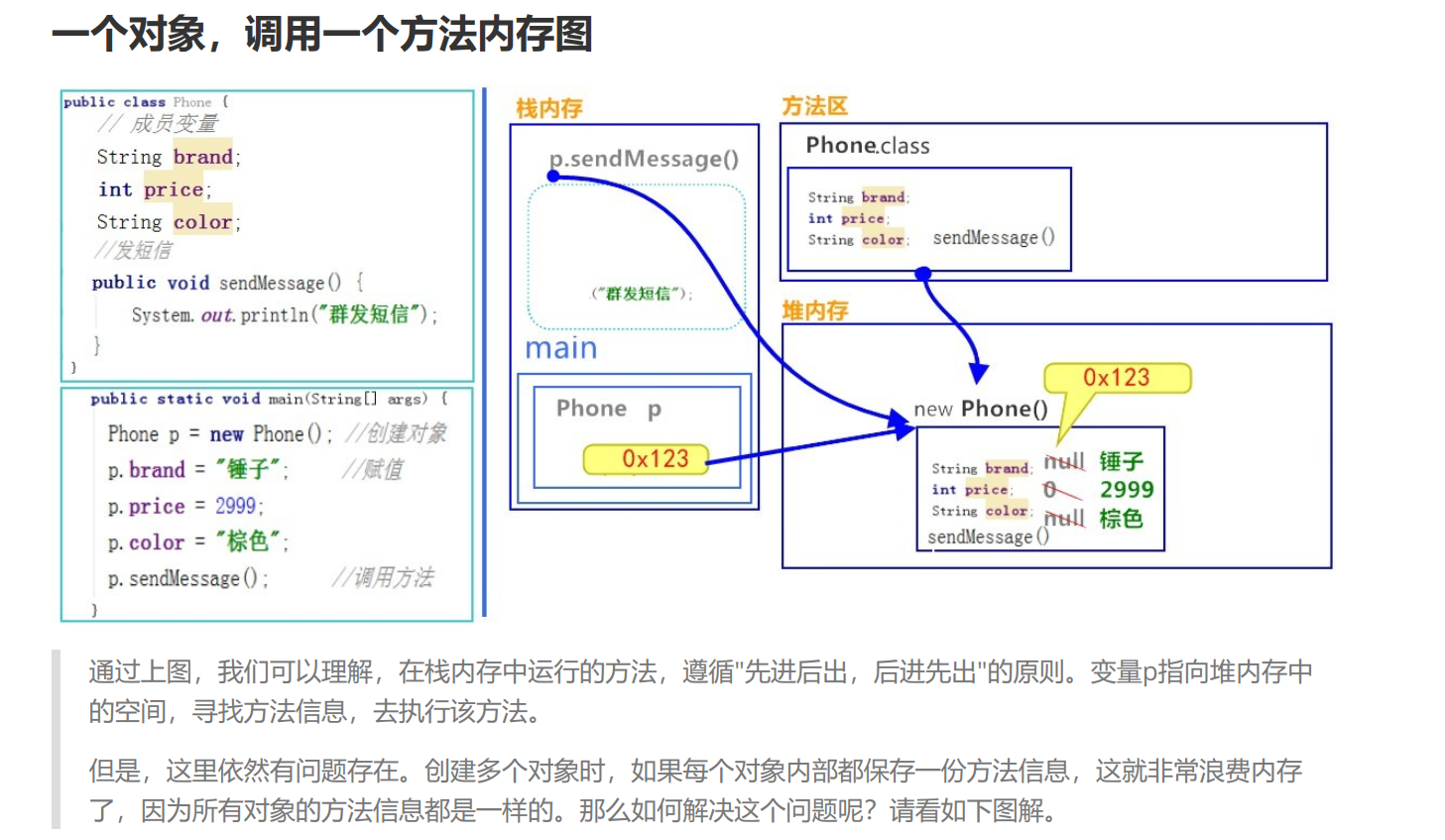


1. 使用类的成员变量或成员方法

对象名.成员变量；

对象名.成员方法([参数])；

#### 对象内存图



方法区内存储着类成员变量和成员方法，其中，成员方法有自己的地址A；运行中的方法存储在栈中，栈先进后出（压栈），程序执行从Main开始，因此，Main先进入栈，执行到new对象时，new 对象进入堆，此时 ，栈中存储对象在堆中的地址B；堆中存储初始化后的成员变量，以及成员方法的地址A；栈中继续执行对成员变量进行操作，操作 成员方法时，由B🡪A,找到该方法，执行完成后出栈。

多个不同对象（new 多个对象 ）也是同样的方法。

引用类型作为参数，传递的是地址值。

#### 成员变量和局部变量区别

* 在类中的位置不同 重点

成员变量：类中，方法外

局部变量：方法中或者方法声明上(形式参数)

* 作用范围不一样 重点

成员变量：类中

局部变量：方法中

* 初始化值的不同 重点

成员变量：有默认值

局部变量：没有默认值。必须先定义，赋值，最后使用

* 在内存中的位置不同 了解

成员变量：堆内存

局部变量：栈内存

* 生命周期不同 了解

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

### 封装

原则

将属性隐藏起来，若需要访问某个属性，提供公共方法对其访问。

封装的步骤

1. 使用 private 关键字来修饰成员变量。

2. 对需要访问的成员变量，提供对应的一对 getXxx 方法 、 setXxx 方法

注意：getXxx方法不能有传入参数，返回值类型和成员变量对应

SetXxx方法不能有返回值，传入参数和成员变量对应

若成员变量是Boolean类型，setXxx不变，对应的是isXxx,用法 同getXxx

### this关键字：在重名时，进行区分

当局部变量和成员变量重名时，默认就近原则，优先使用局部变量

如果需要访问本类中成员变量，使用关键词this.成员变量名

通过谁调用的方法，谁就是this，例如，person.getName()方法中，person调用的方法，则person就是this。

### 构造方法

用new关键字新建一个对象时，实际上就是使用该类的构造方法是 。

注意事项

1. 如果你不提供构造方法，系统会给出无参数构造方法。

2. 如果你提供了构造方法，系统将不再提供无参数构造方法。

3. 构造方法是可以重载的，既可以定义参数，也可以不定义参数。



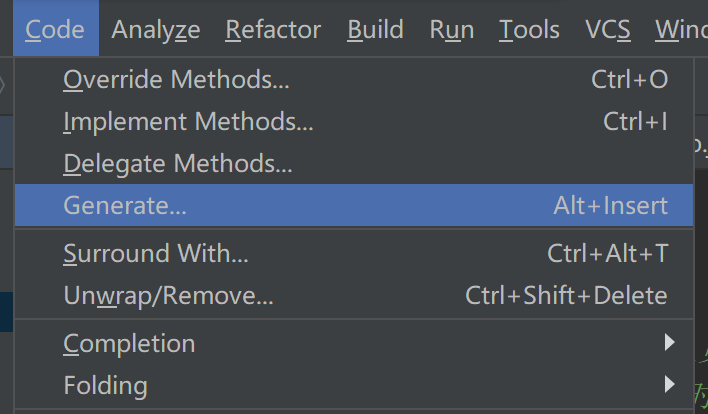
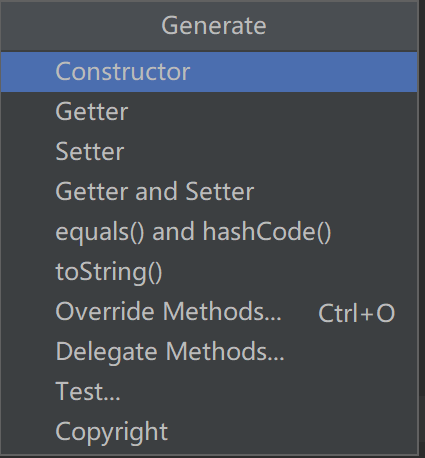
### 定义一个标准的类：JavaBean

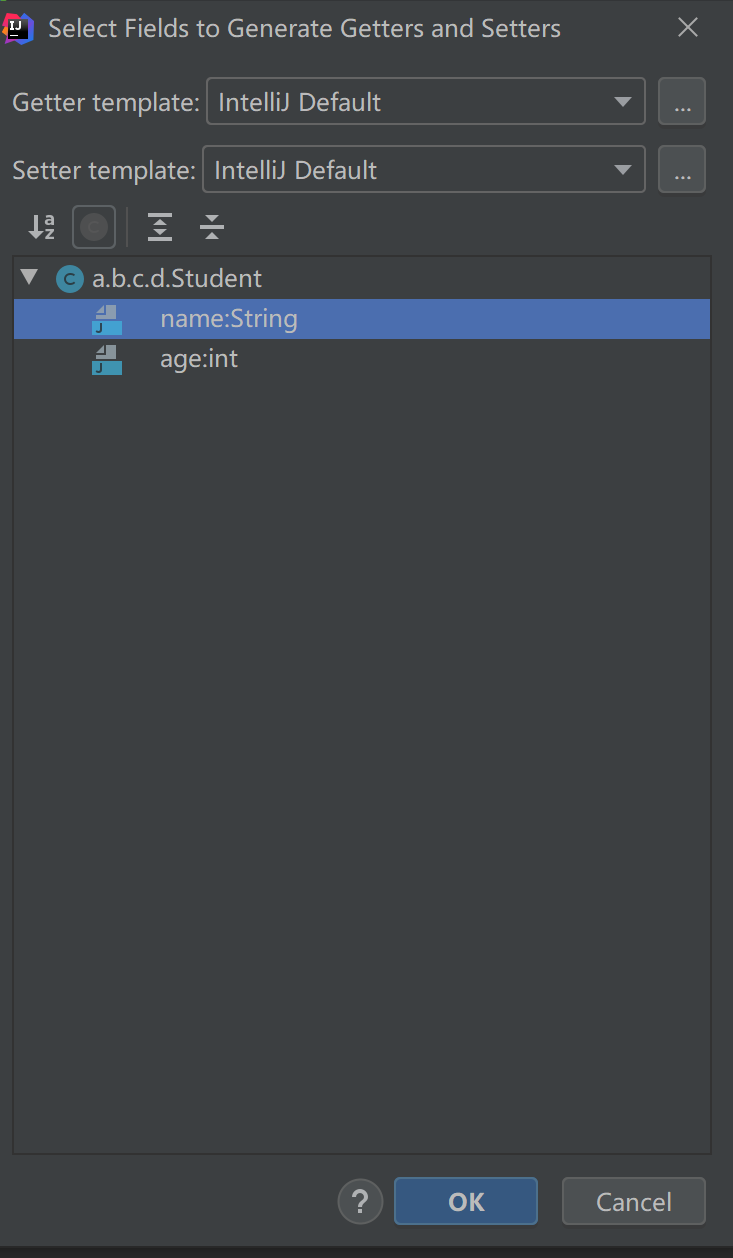
要求:

* 所有成员变量用private修饰
* 为每一个成员变量 编写 getter/setter方法
* 编写一个无参数构造方法
* 编写一个全参数构造方法

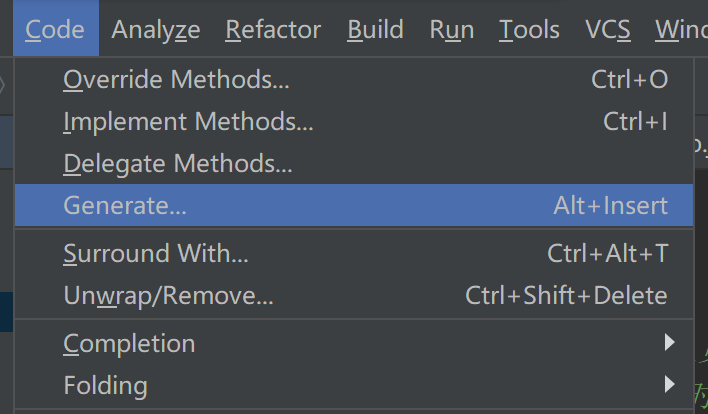
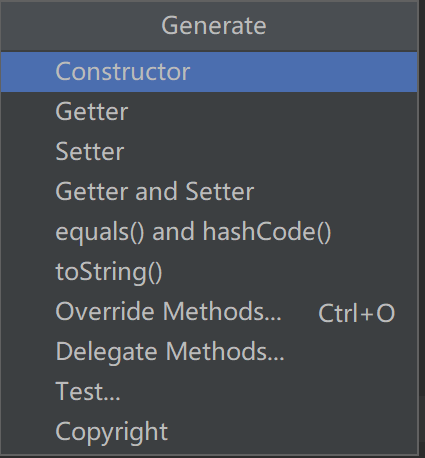
### getter/setter代码自动生成 ：

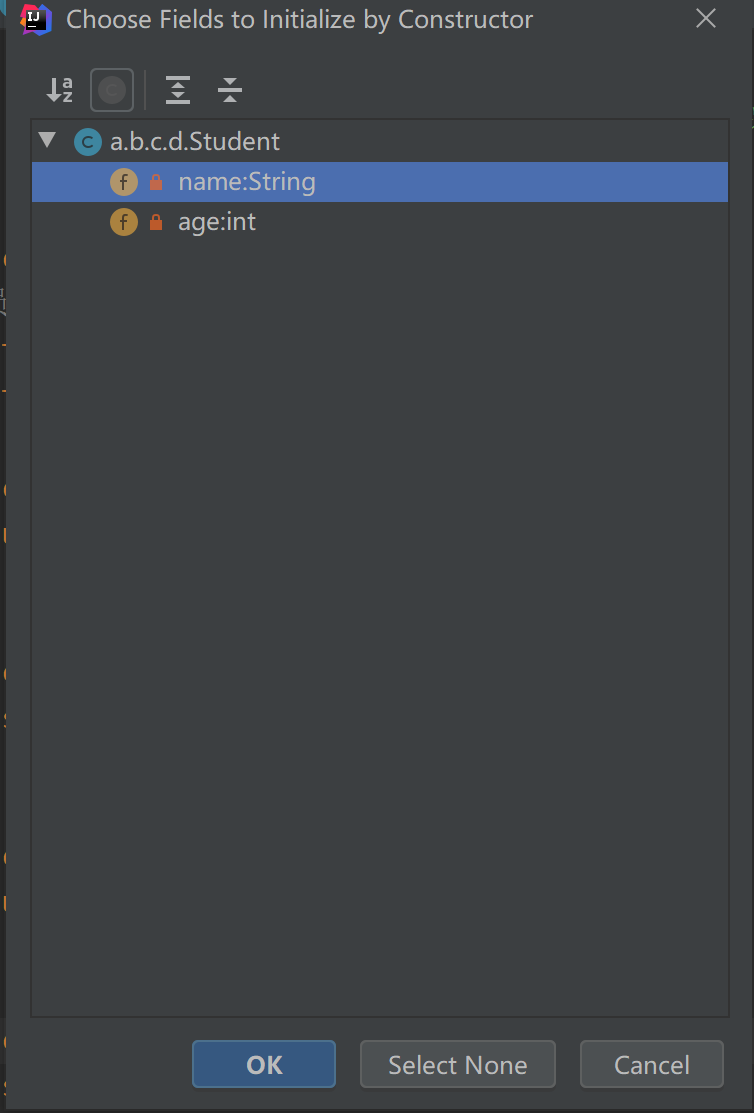
编写完private成员变量后 ，可以自动生成getter/setter方法，步骤如下：

  点击getter and setter

按住Shift选中想要的成员变量，点击OK即可

### 无参/全参数构造方法自动生成：

  选择Constructor

 无参构造方法直接点击Select None即可生成；全参构造方法按Shift多选，选择全部参数 ，点击 OK即可

## Day 07

API使用步骤

1. 打开帮助文档。

2. 点击显示，找到索引，看到输入框。

3. 你要找谁？在输入框里输入，然后回车。

4. 看包。java.lang下的类不需要导包，其他需要。

5. 看类的解释和说明。

6. 学习构造方法。

7. 使用成员方法

API文档

### 引用类型一般使用步骤：

* 导包

使用import关键字导包，在类的所有代码之前导包，引入要使用的类型，java.lang包下的所有类无需导入。 格式：import 包名.类名;

* 创建对象

使用该类的构造方法，创建一个该类的对象。 格式：数据类型 变量名 = new 数据类型(参数列表);

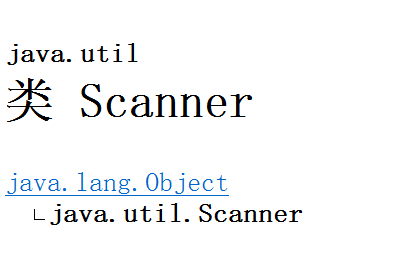
* 调用方法

调用该类的成员方法，完成指定功能。 格式：变量名.方法名();

### Scanner类

使用方法：

1. 导包：



import java.util.Scanner;

1. 创造类：

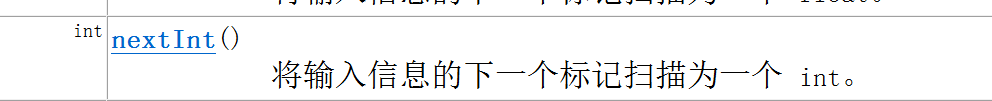


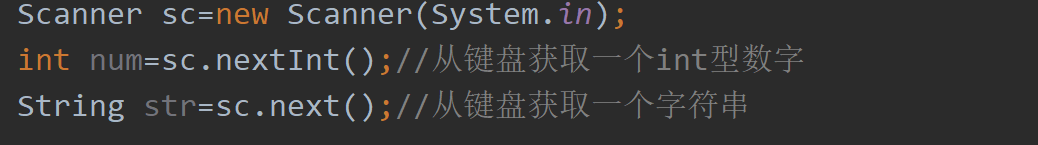
例如：

Scanner sc=new Scanner(System.in);

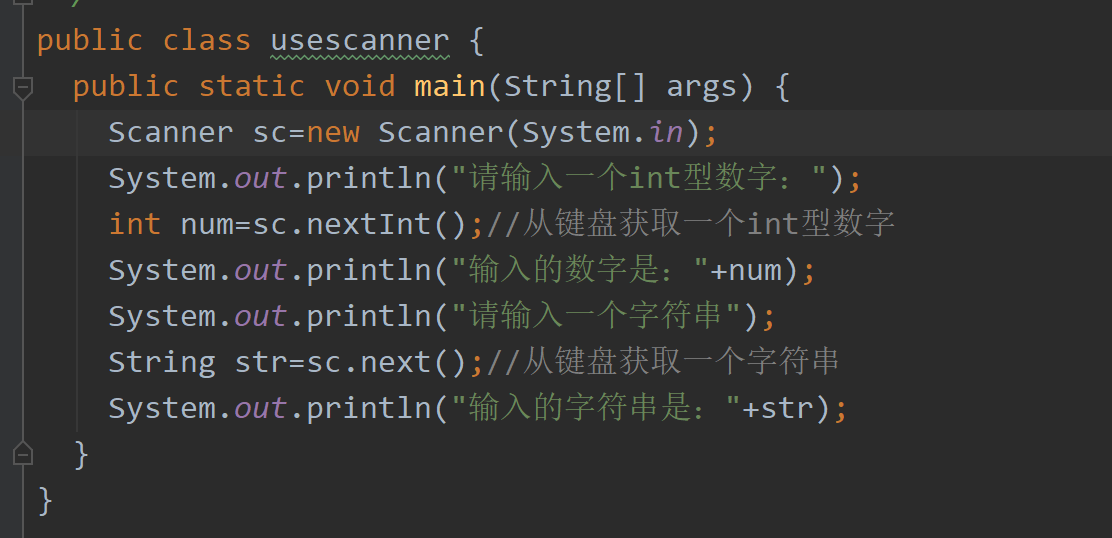
System.in表示从键盘输入

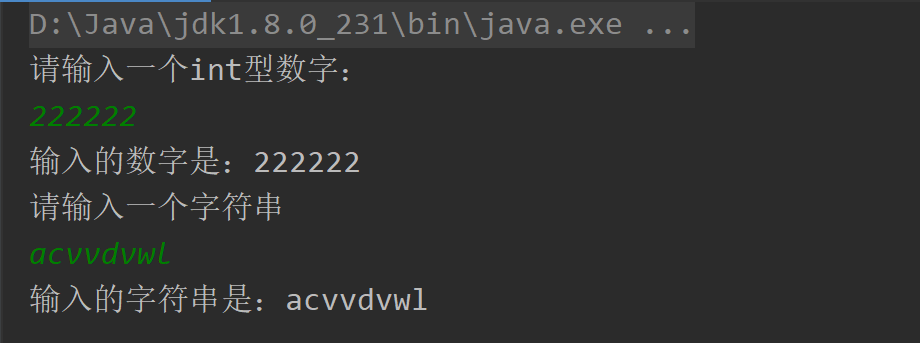
1. 使用，调用API中的成员方法，查阅API文档，例如：





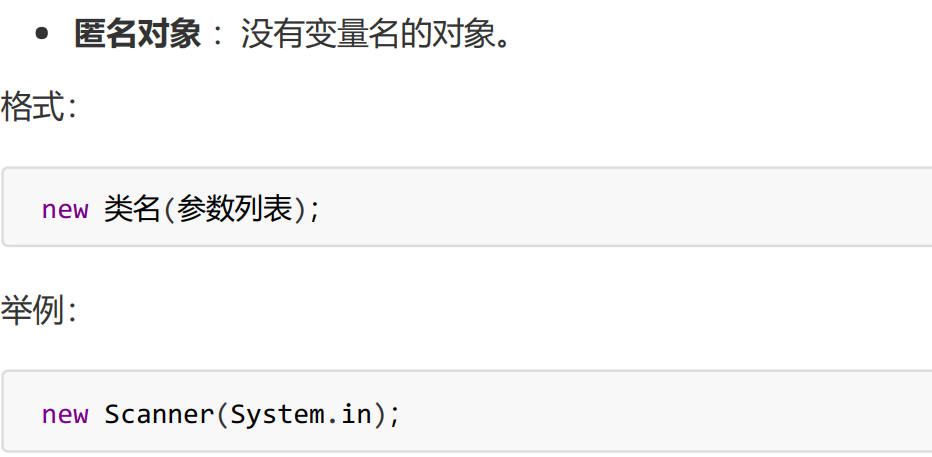
输出结果：绿色表示键盘输入，白色是系统输出





### 匿名对象

使用建议：若确定对象至使用一次，可以使用匿名对象



应用场景

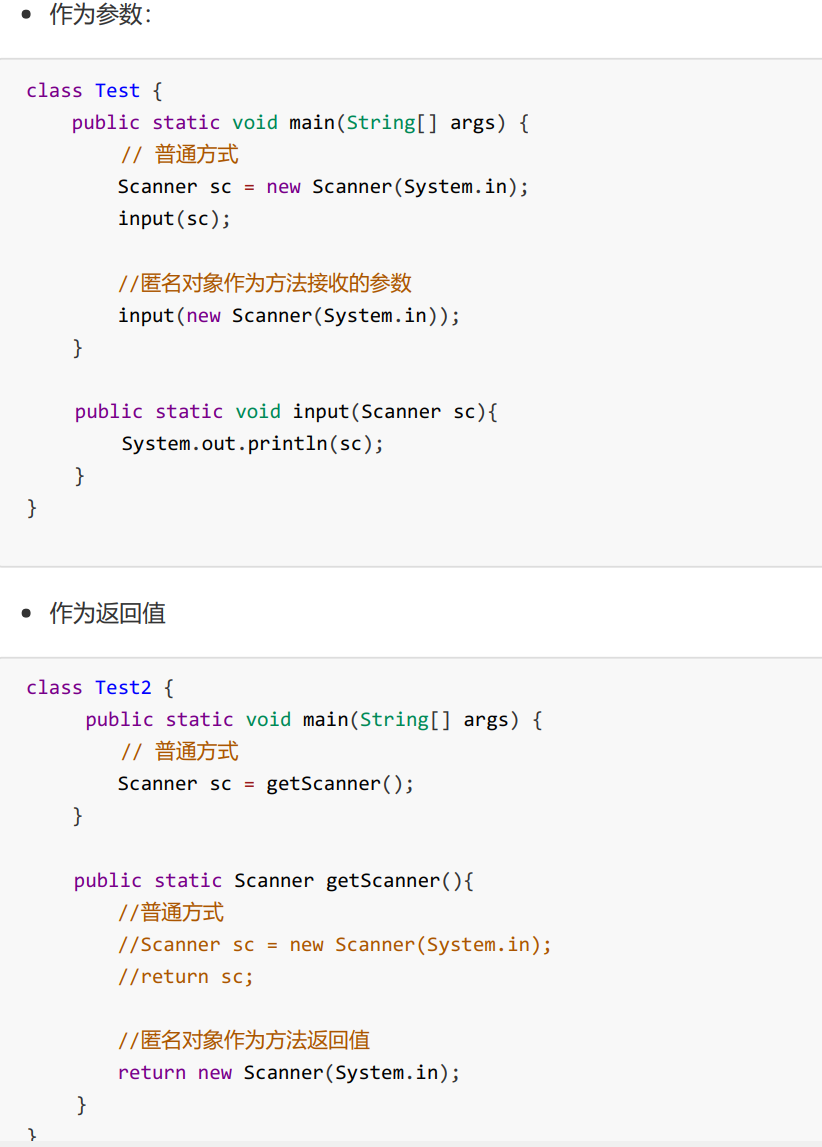
1. 创建匿名对象直接调用方法，没有变量名

new Scanner(System.in).nextInt();

1. 一旦调用两次方法，就是创建了两个对象，造成浪费

小贴士：一个匿名对象，只能使用一次。

1. 匿名对象可以作为方法的参数和返回值



### Random使用



创建一个Random对象，每次调用nextInt（）方法都会生成一个随机数；

例如：r.nextInt(n),取值范围 ：[0,n)

### Array 对象数组

数组内可以存储任何类型数据，包括对象；

缺点：创建之后，程序运行期间长度不可更改；

到目前为止，我们想存储对象数据，选择的容器，只有对象数组。而数组的长度是固定的，无法适应数据变化的需求。为了解决这个问题，Java提供了另一个容器 java.util.ArrayList 集合类,让我们可以更便捷的存储和操作对象数据。

### ArrayList 集合类

* java.util.ArrayList 是大小可变的数组的实现，存储在内的数据称为元素。此类提供一些方法来操作内部存储的元素。 ArrayList 中可不断添加元素，其大小也自动增长。
* 使用步骤：



注意：出现<E>的地方，必须使用某一引用类型代替E，不能用基本类型代替它，以及，确定了一个引用类型，则不可以使用别的引用类型。

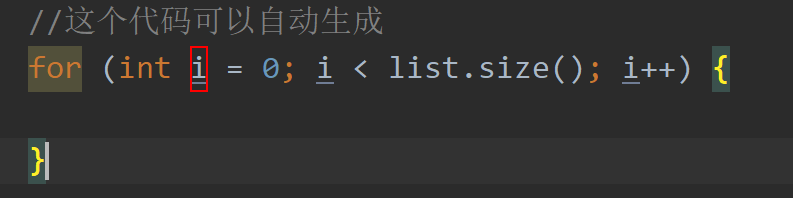
对于ArrayList，若不进行赋值直接打印，输出的不是地址值而是空的，会得到 :[]

#### ArrayList常用方法

* public boolean add(E e) ：将指定的元素添加到此集合的尾部。

对于ArrayList来说，add添加一定会成功，所以返回值可用可不用；对于其他几何，add不一定成功，因此，需要 一个返回值来确定添加动作是否成功。

* public E remove(int index) ：移除此集合中指定位置上的元素。返回被删除的元素。
* public E get(int index) ：返回此集合中指定位置上的元素。返回获取的元素。
* public int size() ：返回此集合中的元素数。遍历集合时，可以控制索引范围，防止越界。
* list.fori :自动生成集合list的遍历for代码



#### ArrayList存储基本类型数据的方法

ArrayList对象不能存储基本类型，只能存储引用类型的数据。类似 <int> 不能写，但是存储基本数据类型对应的包装类型是可以的。所以，想要存储基本类型数据， <> 中的数据类型，必须转换后才能编写，转换写法如下：



从JDK1.5以后，实现了基本类型到包装类型的自动装箱和自动拆箱

## Day 08

### String类

* 类 String 中包括用于检查各个字符串的方法，比如用于比较字符串，搜索字符串，提取子字符串以及创建具有翻译为大写或小写的所有字符的字符串的副本
* 特点：
* 字符串不变：字符串的值在创建后不能被更改
* 因为String对象是不可变的，所以它们可以被共享。
* "abc" 等效于 char[] data={ 'a' , 'b' , 'c' }

#### 使用步骤



#### 常用方法

* ==和equals

对于基本对象，==比较的是两者的数值

对于引用类型，==比较的是两者的地址值

字符串常量池：有且仅有直接用双引号创建的字符串，保存在字符串常量池中

若需要进行字符串内容的比较，可以使用如下方法：

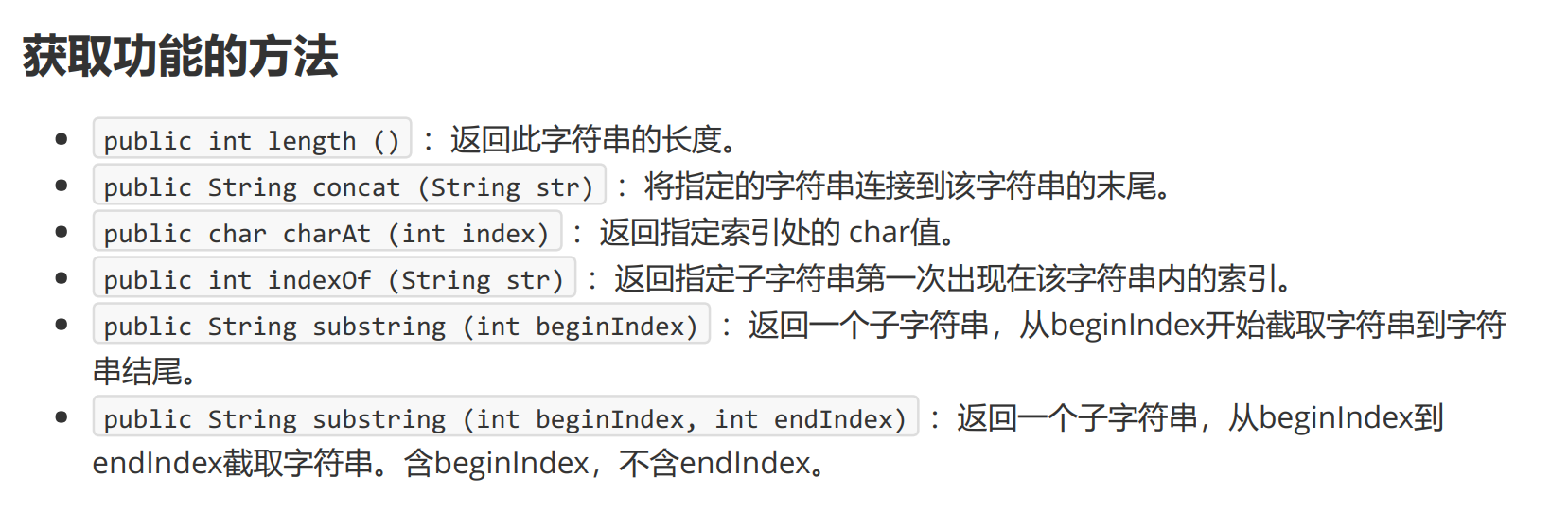
public Boolean equals(Object obj):参数可以是任何对象

equals方法具有对称性，但是，若是比较双方有一个是常量，推荐常量.equlas(变量)

若是进行忽略英文字符大小写的字符串内容比较，可以使用：

public Boolean equalsIgnore(String str)

* 获取功能的方法

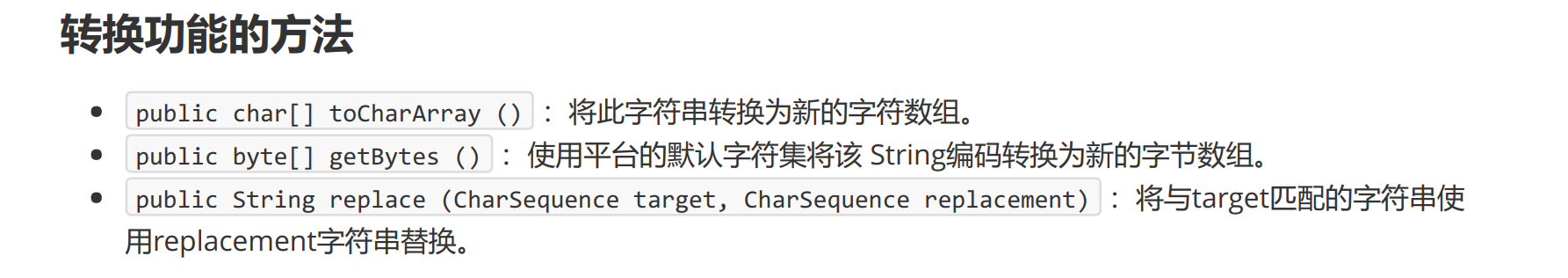


* 字符串截取方法

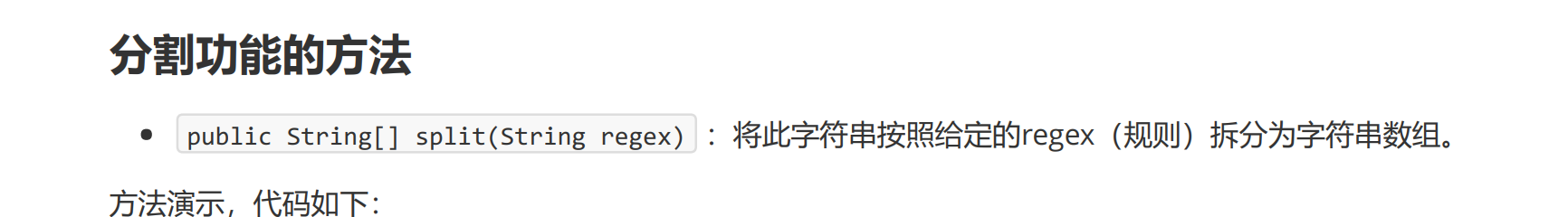
public String substring(int index):截取从参数位置到字符串末尾，返回一个新的字符串

public String substring(int begin,int end),截取【begin，end）字符串，返回一个新的字符串

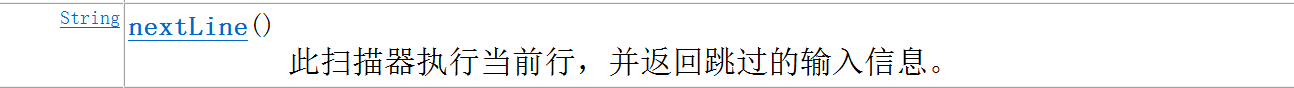
* 字符串转换



* 字符串切割



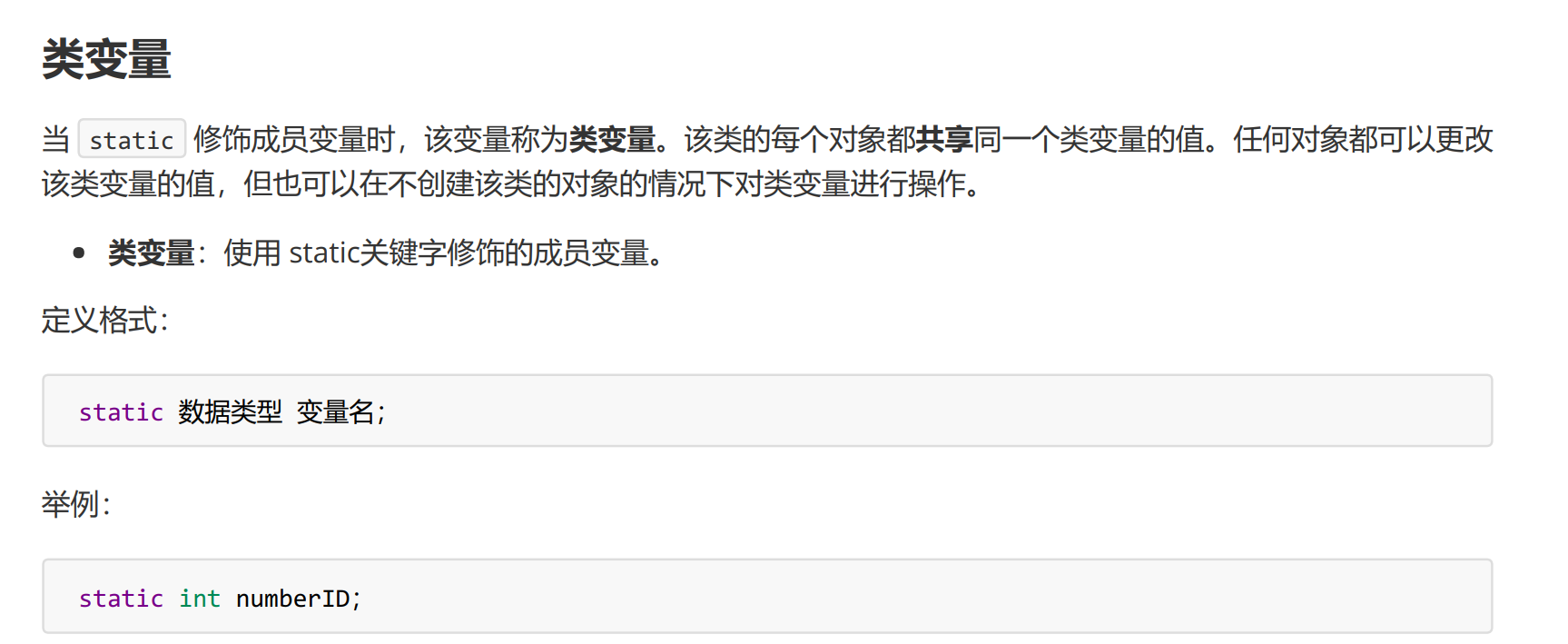
练习二中使用到的Scanner方法



### Static关键字

关于 static 关键字的使用，它可以用来修饰的成员变量和成员方法，被修饰的成员是属于类的，而不是单单是属于某个对象的，本类对象共享该属性。也就是说，既然属于类，就可以不靠创建对象来调用了。

* static 成员变量



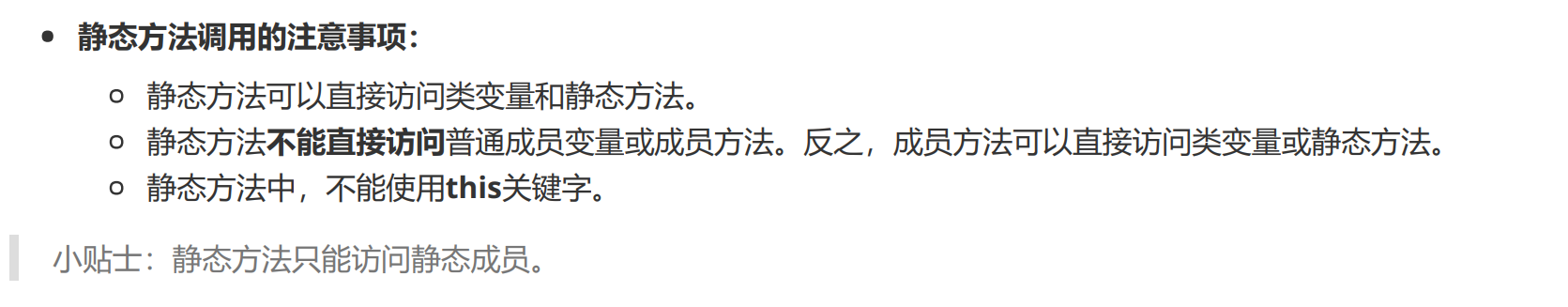
* static 成员方法



调用方式：

1. 通过对象名称调用：正确不推荐
2. 通过类名称调用：正确且推荐

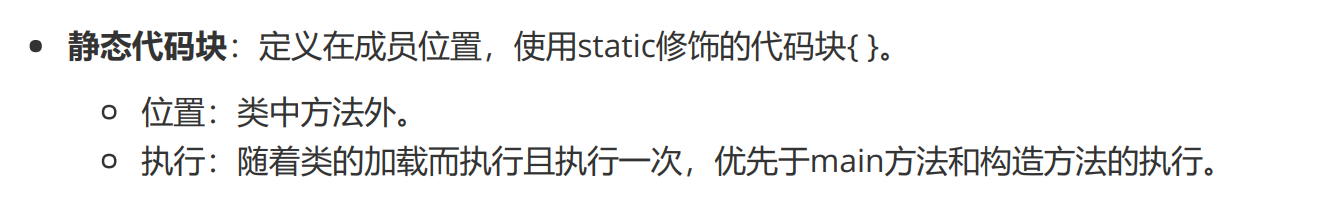
* 注意事项



静态方法可以访问静态成员和静态变量；不可以直接访问普通成员变量或者普通成员方法；

普通成员方法可以访问静态成员变量和静态成员方法；也可以访问普通成员变量和普通成员方法。

* 静态代码块：用来一次性对静态变量赋值





### Arrays

java.util.Arrays 此类包含用来操作数组的各种方法，比如排序和搜索等。其所有方法均为静态方法，调用起来非常简单。

### 操作数组的方法

#### public staticString toString（int[] a）将输入数组参数转换成String



Math类

#### public static void sort(int[] a) ：对指定的 int 型数组按数字升序进行排序

注意：不生成新数组，只是将原数组进行排序

不仅是对int型数组排序，String型也可以，数字按照数字升序，字符串按照字母升序。如果是自定义的类型，那么这个自定义的类型需要有Comparable或者Comparator接口的支持。



### Math 类

java.lang.Math 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。类似这样的工具类，其所有方法均为静态方法，并且不会创建对象，调用起来非常简单。

#### public static double abs(double a) ：返回 double 值的绝对值

double d1 = Math.abs(‐5); //d1的值为5

double d2 = Math.abs(5); //d2的值为5

#### public static double ceil(double a) ：返回大于等于参数的最小的整数

double d1 = Math.ceil(3.3); //d1的值为 4.0

double d2 = Math.ceil(‐3.3); //d2的值为 ‐3.0

double d3 = Math.ceil(5.1); //d3的值为 6.0

#### public static double floor(double a) ：返回小于等于参数最大的整数

double d1 = Math.floor(3.3); //d1的值为3.0

double d2 = Math.floor(‐3.3); //d2的值为‐4.0

double d3 = Math.floor(5.1); //d3的值为 5.0

#### public static long round(double a) ：返回最接近参数的 long。(相当于四舍五入方法)

long d1 = Math.round(5.5); //d1的值为6.0

long d2 = Math.round(5.4); //d2的值为5.0

## Day 09

### 继承：共性抽取

* 父类：超类（superclass）或者基类

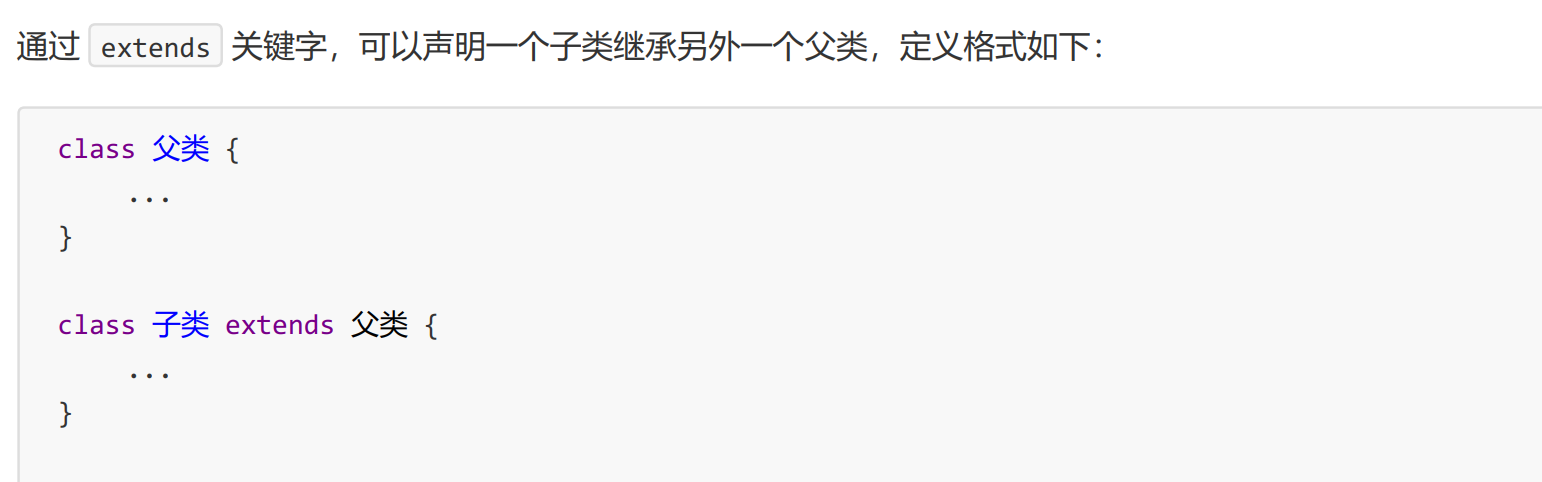
子类：派生类

* 继承：就是子类继承父类的属性和行为，使得子类对象具有与父类相同的属性、相同的行为。子类可以直接访问父类中的非私有的属性和行为。
* 好处：

1. 提高代码的复用性。

2. 类与类之间产生了关系，是多态的前提。

* 子类定义格式



#### 父类、子类同名成员变量（方法）访问优先级

* 直接调用看调用者：父类.同名成员变量就是访问父类；子类.同名成员变量就是访问子类；
* 间接调用（使用成员方法调用）：成员方法属于谁优先访问谁，没有则向上找

#### 局部变量、本类成员变量、父类成员变量重名访问区分

局部变量：直接写

本类成员变量：this.成员变量

父类成员变量：super.成员变量

#### 重写（Override）与重载（Overlord）

* 重写【覆盖 、覆写】发生在继承中，子类中出现与父类一模一样的方法时（返回值类型，方法名和参数列表都相同），会出现覆盖效果，也称为重写或者复写。声明不变，重新实现。

特点：创建的是子类对象，则优先使用子类方法

* 注意事项：

1. 子类方法覆盖父类方法，返回值类型、函数名和参数列表都要一模一样
2. 子类当中返回值必须小于等于父类返回值范围
3. 子类方法覆盖父类方法，必须要保证权限大于等于父类权限修饰符（public>protected>(default)>private 注意：default不是关键字default，而是不写，留空）

* 重载发生在方法名相同，参数列表不相同的时候。

#### 注解@Override

@Override 卸载方法前面，用来检测是不是有效的正确的重写

这是一种可选的安全检测手段，不写程序也不会报错。

#### 父类和子类构造方法调用优先级与注意事项

1. 构造方法的名字是与类名一致的。所以子类是无法继承父类构造方法的。

2. 构造方法的作用是初始化成员变量的。所以子类的初始化过程中，必须先执行父类的初始化动作。子类的构造方法中默认有一个 super() ，表示调用父类的无参构造方法，父类成员变量初始化后，才可以给子类使用。

3. 若是父类仅仅有有参数的构造方法，则子类会报错，此时必须在子类中通过super重载父类构造方法,程序才能正常执行。

4. super的父类构造调用，必须是子类构造方法的第一条语句。且只有子类构造方法，才能调用父类构造方法。

#### super关键字的三种用法

1. 在子类成员方法中，访问父类成员变量
2. 在子类成员方法中，访问父类成员方法
3. 在子类的构造方法中，访问父类的构造方法super（）：没有. 直接（）

#### this关键字的三种用法

1. 在本类成员方法中，访问本类成员变量
2. 在本类成员方法中，访问本类另一个成员方法
3. 构造方法的重载调用：在本类的构造方法中，访问本类的另一个构造方法this（） 没有.直接（）

例如，本类无参构造调用本类有参构造，那就是this（参数），这里，this（……）也必须是构造方法的第一条语句且是唯一一个。

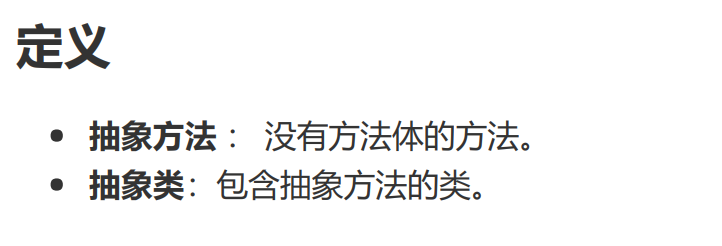
super和this构造调用不能同时使用。

#### 继承的特点：

1. Java只支持单继承，不支持多继承：一个类最多只有一个直接父类
2. Java支持多层继承(继承体系) 。顶层父类是Object类。所有的类默认继承Object，作为父类。
3. 子类和父类是一种相对的概念。

### 抽象类

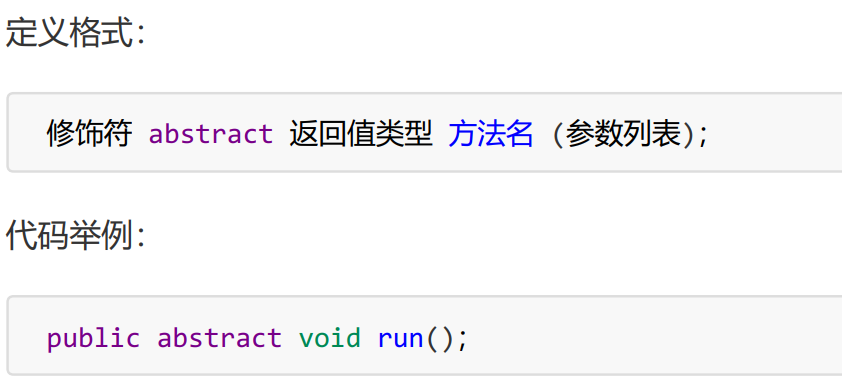
#### 定义



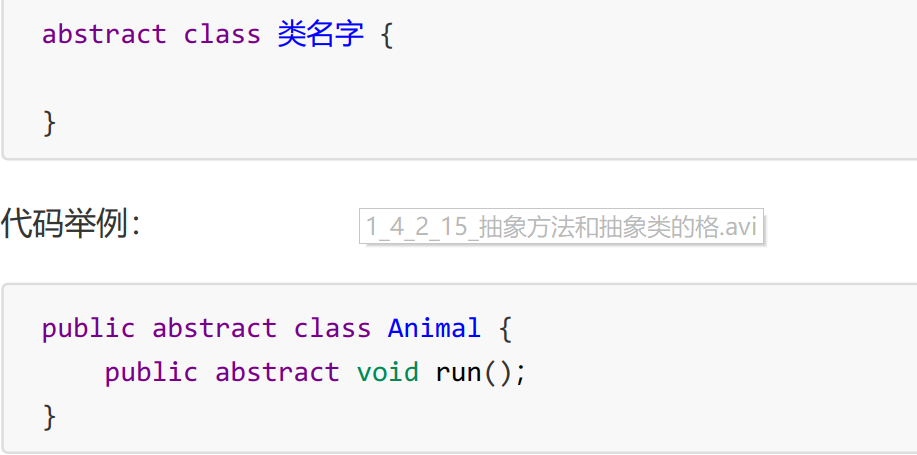
如果父类不确定某一方法具体实现，可以只声明一个方法名称（即没有方法体，抽象方法），此时，父类包含了抽象方法，成为一个抽象类；子类继承后，重写该方法，写方法具体实现。

#### 抽象方法和抽象类使用格式：abstract关键字

抽象方法：



抽象类：



因为包含抽象方法的类是抽象类，因此，若是类中存在抽象方法，则类名一定要用abstract进行修饰，否则会报错。

#### 抽象类的使用：

继承抽象类的子类必须重写父类所有的抽象方法。否则，该子类也必须声明为抽象类。最终，必须有子类实现该父类的抽象方法，否则，从最初的父类到最终的子类都不能创建对象，失去意义。

#### 使用注意事项：

1. 不能直接new抽象类对象，如果创建，编译无法通过而报错。只能new其非抽象子类的对象。

理解：假设创建了抽象类的对象，调用抽象的方法，而抽象方法没有具体的方法体，没有意义。

1. 抽象类的子类，必须重写抽象父类中所有的抽象方法，否则，编译无法通过而报错。除非该子类也是抽象类。覆盖重写：去掉父类抽象方法的abstract关键字，补上方法体。

理解：假设不重写所有抽象方法，则类中可能包含抽象方法。那么创建对象后，调用抽象的方法，没有意义

1. 抽象类中，可以有构造方法，是供子类创建对象时，初始化父类成员使用的。

理解：子类的构造方法中，有默认的super()，需要访问父类构造方法。

1. 抽象类中，不一定包含抽象方法，但是有抽象方法的类必定是抽象类。

理解：未包含抽象方法的抽象类，目的就是不想让调用者创建该类对象，通常用于某些特殊的类结构设计。

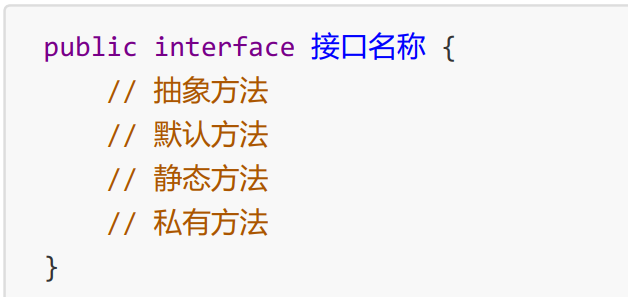
## Day 10

### 接口：多个类的公共规范，关键字：interface

#### 概述：

接口，是Java语言中一种引用类型，是方法的集合，接口的内部主要就是封装了方法，包含抽象方法（JDK 7及以前），默认方法和静态方法（JDK 8），私有方法（JDK 9）。

#### 接口定义



##### 定义抽象方法

所有版本Java中，接口都能定义抽象方法：

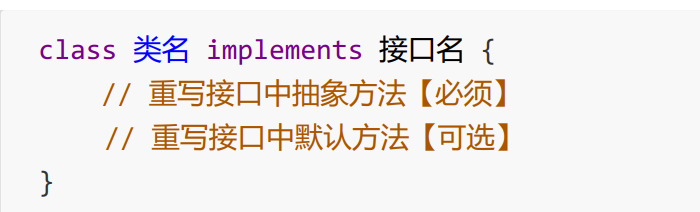
格式：public abstract 返回值类型 方法名称（参数列表）

注意事项：接口中的抽象方法，public abstract 是固定的关键字，可以选择性忽略不写，但即使不写或者不写全，也还是抽象方法。

#### 接口使用步骤

接口不能直接使用，要有一个“实现类”来实现接口

格式如下：



接下来，创建实现类的对象，实现对于接口的使用。

注意事项：1.若实现类并没有覆盖重写接口中所有的抽象方法，那么这个实现类自己必须是抽象类。

##### 从Java 8 中，接口中可以定义默认方法 格式如下：

public default 返回值类型 方法名称（参数列表）{

方法体

}

备注：接口中的默认方法，可以解决接口升级的问题

也就是说，如果想在接口中新增抽象方法，所有实现了改接口的实现类都必须要修改；此时可以将方法设定为default方法（这里的public可以省略不写），这样，可以不用修改其他实现类。

注意事项：1.接口的默认方法，可以通过接口实现类对象直接调用

2.接口的默认方法，也可以被接口实现类覆盖重写

##### 从Java 8 中，接口中可以定义静态方法

格式如下：

public static 返回值类型 方法名称（参数列表）{

方法体}

接口中静态方法的使用：

1. 不能通过接口实现类的对象直接调用
2. 正确方法：通过接口名称，直接调用其中的静态方法，即：

接口名称.静态方法（参数）

#### 从Java 9开始，接口中允许定义私有方法

问题描述：需要一个接口内共有的方法，用来解决多个默认方法之间重复代码的问题，且这个方法只允许接口内使用，不允许接口的实现类使用。

接口内的私有方法分成两类：

1. 普通私有方法：解决多个默认方法之间重复代码的问题

格式：

private 返回值类型 方法名称（参数列表）{

方法体

}

1. 静态私有方法：解决多个静态方法之间重复代码问题

格式：

private static 返回值类型 方法名称（参数列表）{

方法体

}

#### 接口中的成员变量

1. 接口中可以定义成员变量，必须使用public static final 固定修饰，就算不写修饰词直接定义也是同样的属性
2. 接口中的常量必须进行赋值，且一旦赋值之后就不可以进行修改
3. 接口中常量的名称，应该使用完全大写的字母，用下划线进行分割

#### 接口使用注意事项：

1. 接口没有静态代码块或者构造方法
2. 一个类的直接父类是唯一的，但是一个类可以同时实现多个接口

格式：

public class 实现类名称 inplements 接口名称1，接口名称2[,接口名称3]{

//覆盖重写接口中所有方法

}

1. 若实现类实现的多个接口中有重复的抽象方法，那么只需要覆盖重写一次就可以
2. 若实现类没有覆盖重写接口中所有的抽象方法，那么该实现类是一个抽象类
3. 若实现类实现的多个接口中，存在重复的默认方法，那么实现类一定要对冲突的默认方法就行覆盖重写
4. 一个类，若是直接父类当中的方法和接口中的默认方法产生了冲突，优先用父类中的方法

### 多态： 是指同一行为，具有多个不同表现形式

多态是继封装、继承之后，面向对象的第三大特性。

继承（extends） 接口（implements）实现是多态性的前提。

* 代码中的多态性：父类引用指向子类对象

格式：

父类名称 对象名=new 子类名称（）；

或者：

接口名称 对象名=new 实现类名称（）

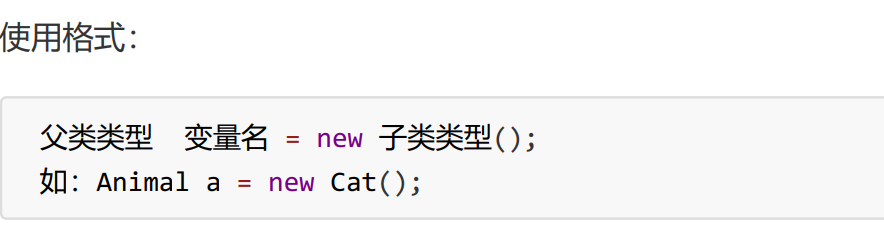
* 多态中，成员方法的访问规则是：

new 的是谁就优先用谁，没有向上找

#### 多态的转型：

* 向上转型：多态本身是子类类型向父类类型向上转换的过程，这个过程是默认的。

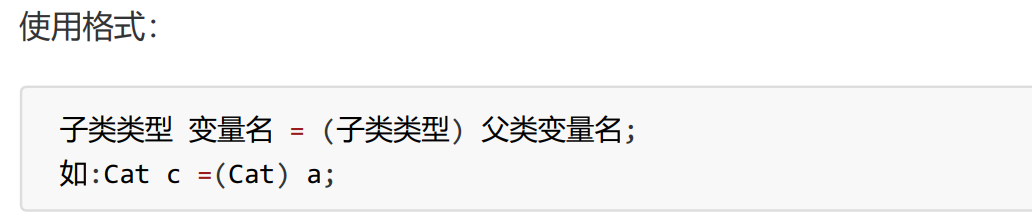
当父类引用指向一个子类对象时，便是向上转型。



* 向下转型：父类类型向子类类型向下转换的过程，这个过程是强制的

一个已经向上转型的子类对象，将父类引用转为子类引用，可以使用强制类型转换的格式，便是向下转型。或者说：

将父类对象还原成为本来的子类对象



向上转型一定是安全的，但缺点是：如果对象一旦向上转型为父类，那么就无法调用子类的原有的内容。

解决方法：用对象的向下转型【还原】

## Day 11

### final 关键字

有四种用法，下面逐一介绍：

对于类和方法来说，abstract和final不能同时使用，矛盾

#### final修饰一个类：最终类

格式 ：

public final class 类名称 { }

被final修饰的类不可以有子类（太监类），也因此，其中所有的成员方法都不可以被覆盖重写。

但是 ，由于final类可以有父类，因此，final类可以覆盖重写父类的成员方法

#### final修饰一个方法：最终方法

格式 ：

修饰符 final 返回值类型 方法名称（参数列表）{

//方法体

}

被final修饰的方法不可以被子类覆盖重写

#### final修饰一个局部变量

final修饰一个局部变量，这个变量只能进行一次赋值操作，且赋值之后不再改变。“一次赋值，终生不变”

对于基本类型来说，不可变指的是变量中的数据不可以变化

对于引用类型来说，不可变指的是变量中的地址值不可以改变，地址值指向的对象内容可以发生变化

#### final修饰一个成员变量

对于成员变量来说，使用final来修饰，也是不可以发生改变的

但是，同局部变量相比，有区别：

1. 由于成员变量是有初始值的，因此，final类的成员变量必须进行手动赋值；
2. 对于final成员变量，要么使用直接赋值 ，要么通过构造方法赋值（用了直接赋值就不可以有构造方法或者setter方法；同理，用构造方法，就不可以直接赋值。这里有一个问题，构造方法有无参构造和有参数构造，同时还有setter方法，若是使用构造方法，有参数和无参数在方法体那里都需要实现对final类的赋值且不可以有setter方法 ，用setter 方法，无参和有参的 构造函数都不可以再对final类进行赋值不可以不用构造方法，反而直接使用setter）



### 权限修饰符 public>protected>(default)> private

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 位置\修饰符 | public | protected | （default） | private |
| 同一个类中 | Y | Y | Y | Y |
| 同一个包中 | Y | Y | Y | N |
| 不同包中的子类中 | Y | Y | N | N |
| 不同包非子类 | Y | N | N | N |

可以访问：Y 不可以访问：N

### 内部类：

内部类分为两种：成员内部类和局部内部类（包括匿名内部类）

#### 成员内部类

定义格式：

修饰符 class 外部类名称{

修饰符 class内部类名称{

//……方法体

}

//方法体

}

注意：内用外可以随意调用，外用内，必须通过内部类对象

使用成员内部类有两种方式：

1. 间接方式：在外部类方法中使用内部类；然后main只是调用外部类的方法
2. 直接方式，公式如下：

外部类名称.内部类名称 对象名=new 外部类名称（）. new 内部类名称（）；

若内部类的成员变量和外部类的成员变量重名，在内部类中调用外部类的成员变量的格式为：

外部类名称.this.外部类成员变量名

#### 局部内部类：

若一个类定义在另一个类的方法内部，那么这就是一个局部内部类。

“局部”：指的是只有当前所属的方法才能使用它，出了这个方法外面就不能用了

定义格式：

修饰符 class 外部类名称{

修饰符 返回值类型 外部类方法名称（参数列表）{

class 局部内部类名称{

//……

}

}

}

### 类的权限修饰规则：

1. 外部类：public /(default)
2. 成员内部类：public/protected/(default)/private
3. 局部内部类：什么都不能写，但是和（default ）不一样
4. 如局部内部类希望访问所在方法中的局部变量，那么这个局部变量必须是有效final从Java开始，只要局部变量事实上没有发生变化，可以省略final关键字。要么直接声明为final类。

原因：解决生命周期问题：new出来的对象在堆内存中，局部变量跟着方法走，在栈内存中，方法运行结束会立刻出栈，此时局部变量也会消失，但是，new出来的对象会持续存在，直到垃圾回收。

### 匿名内部类

如果接口的实现类（或者父类的子类）只需要使用唯一的一次，那么这种情况下就可以省略掉该类的定义，而改为使用【匿名内部类】

格式：

接口名称 对象名 =new接口名称（）{

//覆盖重写所有抽象方法

}

注意事项：

1. 匿名内部类，在创建的时候只能使用唯一一次。若是希望多次使用且内容一样，不推荐使用匿名内部类，可以使用单独定义的实现类。
2. 匿名对象，在调用方法的时候，只能调用唯一一次，如果需要同一个对象调用多次方法，则必须给对象起一个名字
3. 匿名内部类是省略了【实现类/子类】名称，匿名对象是省略了对象名称,这是两者的区别

类可以作为成员变量

接口也可以作为成员变量

接口可以作为方法的参数和返回值

## Day 01

### equals方法重写

如果没有覆盖重写equals方法，那么Object类中默认进行`==`运算符的对象地址比较，只要不是同一个对象，结果必然为false。

如果希望进行对象的内容比较，即所有或指定的部分成员变量相同就判定两个对象相同，则可以覆盖重写equals方法。IDEA可以自动生成equals方法，其中用到了`java.util.Objects`类。这是JDK7增加了Objects类，由一些静态的实用方法组成，这些方法是null-save（空指针安全的）或null-tolerant（容忍空指针的），用于计算对象的hashcode、返回对象的字符串表示形式、比较两个对象。

### Date（）

* `public Date()`：分配Date对象并初始化此对象，以表示分配它的时间（精确到毫秒）。
* `public Date(long date)`：分配Date对象并初始化此对象，以表示自从标准基准时间（称为“历元（epoch）”，即1970年1月1日00:00:00 GMT）以来的指定毫秒数。
* `public long getTime()` 把日期对象转换成对应的时间毫秒值。

### DateFormat类

`java.text.DateFormat` 是日期/时间格式化子类的抽象类，我们通过这个类可以帮我们完成日期和文本之间的转换,也就是可以在Date对象与String对象之间进行来回转换。

\* \*\*格式化\*\*：按照指定的格式，从Date对象转换为String对象。日期🡪文本

\* \*\*解析\*\*：按照指定的格式，从String对象转换为Date对象。 文本🡪日期

### Date小结：

Date now= new Date(); 新建一个Date对象，获取系统当前日期

long time=now.getTime()； 将日期转换为毫秒，在此基础上可以进行日期的计算

System.out.println(now); 可直接打印输出，Fri Dec 27 00:24:14 CST 2019

DateFormat df = new SimpleDateFormat("yyyy年MM月dd日");固定日期格式

Date past = df.parse(birthday); 将字符串转化为对应格式的日期

String str = df.format(date); 将对应格式的日期转换为字符串

日期和字符串之间的转化需要引入以下三个包：

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

使用parse方法需要额外引入：

import java.text.ParseException;

且，主函数要写成public static void main(String[] args) throws ParseException { //……}

### Calendar类

#### 创建对象：getInstance()方法

`java.util.Calendar`是日历类，为抽象类，由于语言敏感性，Calendar类在创建对象时并非直接创建，而是通过静态方法创建，返回子类对象，如下：Calendar静态方法

public static Calendar getInstance()； 使用默认时区和语言环境获得一个日历

代码实现：Calendar cal = Calendar.getInstance();//使用父类接受子类对象，多态



#### 常用方法：

* public int get(int field)：返回给定日历字段的值。

int year=cal.get(Calendar.YEAR);

int mouth=cal.get(Calendar.MONTH);

int date=cal.get(Calendar.DATE);

int date1=cal.get(Calendar.DAY\_OF\_MONTH);

…………

* `public void set(int field, int value)`：将给定的日历字段设置为给定值。

cal.set(Calendar.YEAR,2000);

cal.set(Calendar.MONTH,9);

cal.set(1111,2,22);//同时设置年月日

* `public abstract void add(int field, int amount)`：根据日历的规则，为给定的日历字段添加或减去指定的时间量。

cal.add(Calendar.YEAR,2);

cal.add(Calendar.YEAR,-2);

* public Date getTime()`：返回一个表示此Calendar时间值（从历元到现在的毫秒偏移量）的Date对象。

cal.getTime();

### System

`java.lang.System`类中提供了大量的静态方法，可以获取与系统相关的信息或系统级操作，在System类的API文档中，常用的方法有：

#### public static long currentTimeMillis()`：返回以毫秒为单位的当前时间。

#### public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`：将数组中指定的数据拷贝到另一个数组中。



### StringBuilder类

如果对字符串进行拼接操作，每次拼接，都会构建一个新的String对象，既耗时，又浪费空间。为了解决这一问题，可以使用`java.lang.StringBuilder`类。

查阅`java.lang.StringBuilder`的API，StringBuilder又称为可变字符序列（String类是不可变 的），它是一个类似于 String 的字符串缓冲区，通过某些方法调用可以改变该序列的长度和内容。

原来StringBuilder是个字符串的缓冲区，即它是一个容器，容器中可以装很多字符串。并且能够对其中的字符串进行各种操作。

它的内部拥有一个数组用来存放字符串内容，进行字符串拼接时，直接在数组中加入新内容。StringBuilder会自动维护数组的扩容。

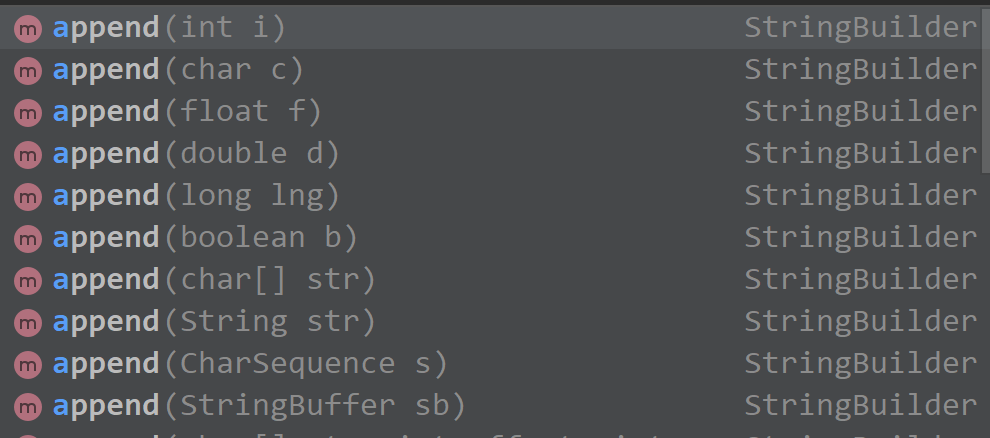
#### 构造方法

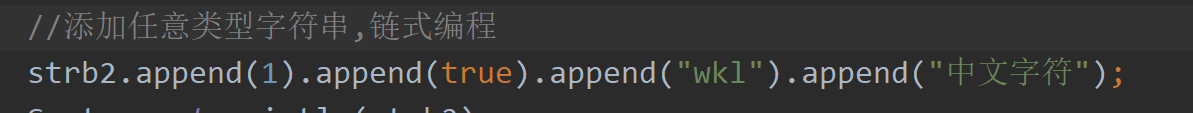
* public StringBuilder()：构造一个空的StringBuilder容器。
* public StringBuilder(String str)：构造一个StringBuilder容器，并将字符串添加进去。

#### 常用方法

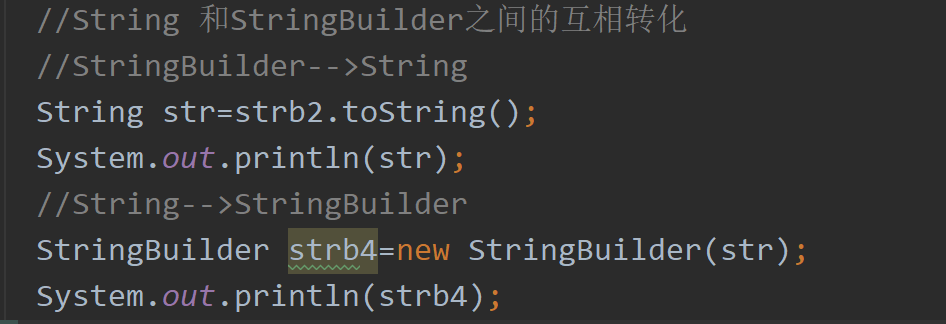
* public StringBuilder append(...)：添加任意类型数据的字符串形式，并返回当前对象自身。

此方法不需要接收返回值 ，可以添加任意类型数据且可以实现链式编程



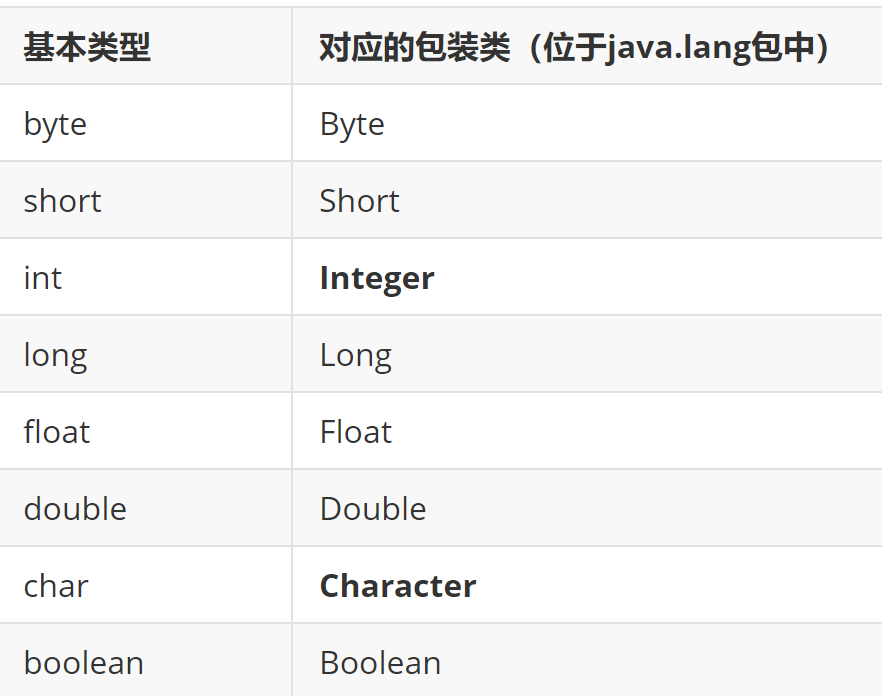


* StringBuilder🡪String : 将当前StringBuilder对象转换为String对象。
* String🡪StringBuilder: 使用StringBuilder的构造方法public StringBuilder(String str)：构造一个StringBuilder容器，并将字符串添加进去。



### 包装类

Java提供了两个类型系统，基本类型与引用类型，使用基本类型在于效率，然而很多情况，会创建对象使用，因为对象可以做更多的功能，如果想要我们的基本类型像对象一样操作，就可以使用基本类型对应的包装类



基本类型与对应的包装类对象之间，来回转换的过程称为”装箱“与”拆箱“：

\* \*\*装箱\*\*：从基本类型转换为对应的包装类对象。（Integer举例）

构造方法：

Interger(int value):构造一个Integer对象，表示指定的int值

Integer（String s）：同上，表示String参数所指示的int值，这里的s必须是基本类的字符串，可以使 “100”，不能是”a”.

静态方法：

static Integer valueOf（int i）返回一个表示指定int 值得Integer实例。

static Integer valueOf（String s）返回表示指定String值的Integer对象

\* \*\*拆箱\*\*：从包装类对象转换为对应的基本类型。

成员方法：

int intValue() 以int类型返回该Integer的值

自动装箱和拆箱：自动进行基本类型和包装类之间的转换，从 JDK1.5之后

### 基本类型数据和字符串之间进行转换

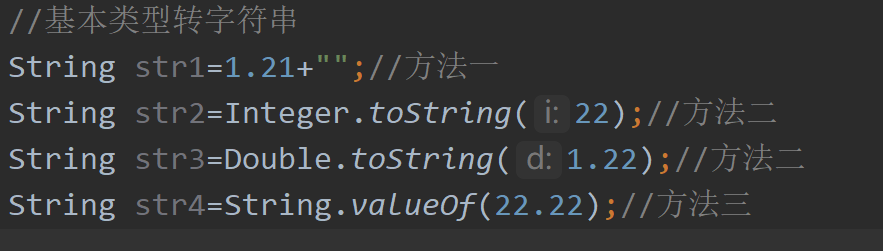
#### 基本类型转换为String：三种方法

* 基本类型的值+” ” (推荐)
* 包装类的静态方法toString(参数)，不是Object类的toString()重载

static String toString(int i) 返回一个表示指定整数的String对象

* String类的静态方法valueOf（参数）

static String valueOf（int i）返回int参数的字符串表示形式

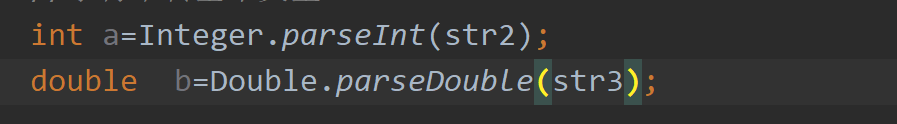


#### 字符串（String）转换成基本类型

使用包装类的静态方法parseXXX(“数值类型的字符串”)

Integer类： static int parseInt(String s )

Double类：static double parseDouble(String s)



## Day 02

### 集合

\* \*\*集合\*\*：集合是java中提供的一种容器，可以用来存储多个数据。

集合和数组既然都是容器，它们有啥区别呢？

\* 数组的长度是固定的。集合的长度是可变的。

\* 数组中存储的是同一类型的元素，可以存储基本数据类型值。集合存储的都是对象。而且对象的类型可以不一致。在开发中一般当对象多的时候，使用集合进行存储。

#### 集合的分类：

继承：子类共性抽取 形成父类（接口）

* List接口 ：1.有序的集合：存储和取出元素顺序相同； 2.允许有重复的元素； 3.有索引可以使用普通的 for循环遍历

包括：Vector集合 ArrayList集合 LinkedList集合

* Set接口：1. 不允许存储重复数据 2.没有索引 ，没有带索引方法，不能用for循环遍历

包括：TreeSet集合 、HashSet集合（以及LinkedHashSet集合）

* Collection接口

由List接口和Set接口共性抽取 ，定义的是单列集合共性的方法。没有带索引的方法。

单列集合类的根接口，用于存储一系列符合某种规则的元素，它有两个重要的子接口，分别是`java.util.List`和`java.util.Set`。其中，`List`的特点是元素有序、元素可重复。`Set`的特点是元素无序，而且不可重复。`List`接口的主要实现类有`java.util.ArrayList`和`java.util.LinkedList`，`Set`接口的主要实现类有`java.util.HashSet`和`java.util.TreeSet`。

#### Collection 常用共性方法

* `public boolean add(E e)`： 把给定的对象添加到当前集合中 。

一般都返回true，不需要特意接受返回值

* `public void clear()` :清空集合中所有的元素。

但是不删除集合

* `public boolean remove(E e)`: 把给定的对象在当前集合中删除。

返回true说明删除该元素成功；返回false说明没有指定的元素，删除失败

* `public boolean contains(E e)`: 判断当前集合中是否包含给定的对象。

包含返回true，不包含返回false

* `public boolean isEmpty()`: 判断当前集合是否为空。

集合为空返回 true，不为空返回false

* `public int size()`: 返回集合中元素的个数。

* `public Object[] toArray()`: 把集合中的元素，存储到数组中。

#### Iterator接口

* 概述

`Collection`接口与`Map`接口主要用于存储元素，而`Iterator`主要用于迭代访问（即遍历）`Collection`中的元素，因此`Iterator`对象也被称为迭代器。

获取迭代器的方法：

\* `public Iterator iterator()`: 获取集合对应的迭代器，用来遍历集合中的元素的。

* 常用方法：

public E next():返回迭代的下一个元素。

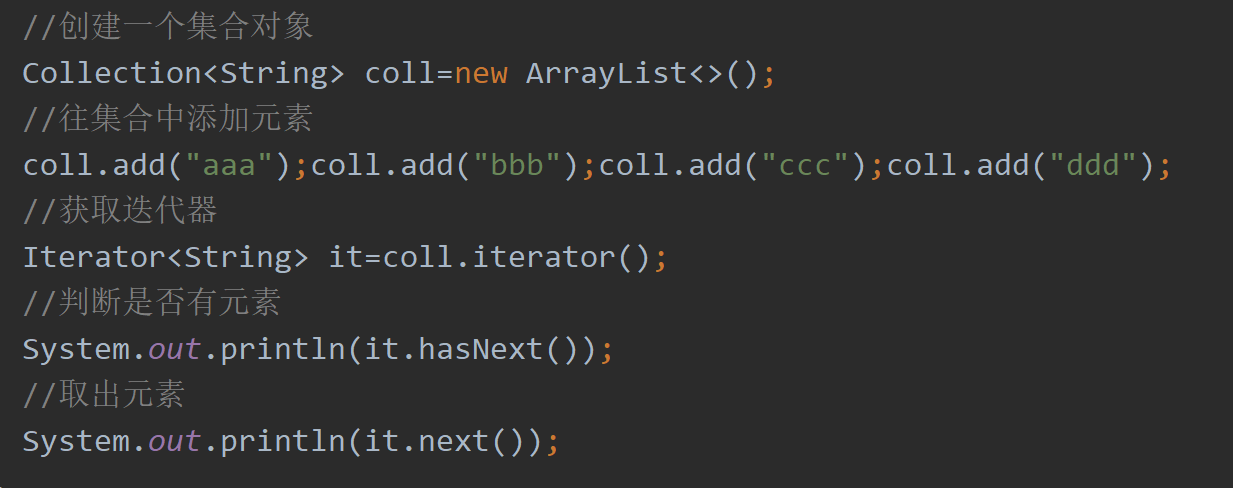
public boolean hasNext():如果仍有元素可以迭代，则返回 true。

* 使用步骤：

1. 使用集合中的方法iterator()获取迭代器的实现类对象，使用Iterator接口接受（多态）

Iterator<E>接口含有泛型，迭代器泛型同集合

1. 使用Iterator接口中的方法hasNext判断是否还有下一个元素
2. 使用Iterator接口中的方法next取出集合中的下一个元素（若没有元素，会抛出NoSuchElemmentException 没有元素异常）



### 增强for（for each）

* 概述

增强for循环(也称for each循环)是\*\*JDK1.5\*\*以后出来的一个高级for循环，专门用来遍历数组和集合的。它的内部原理其实是个Iterator迭代器，所以在遍历的过程中，不能对集合中的元素进行增删操作。

Collection<E> extends Iterable<E>:所有的单列集合都可以使用增强for

public interface Iterable<T> 实现这个接口允许对象成为“foreach”语句的目标

* 格式：

for（集合/数组的数据类型 变量名：集合名/数组名）{

sout（变量名）；

}

