Java基础

Constant常量

> 不可更改的量称之为常量，例如数字“1,2,3”等均为常量，字符串“adfad”也是常量
>
> final修饰称之为符号常量
>
> 常量命名：大写字母+下划线

数据类型

1. **分类，数字为所占字节**

2. **整型：**

表2-4整型数据类型		
类型	**占用存储空间**	**表数范围**
byte	1字节	-2^7 ~ 2^7-1（-128~127）
short	2字节	-2^{15} ~ $2^{15}-1$（-32768~32767）
int	4字节	-2^{31} ~ $2^{31}-1$ (-2147483648~2147483647)约21亿
long	8字节	-2^{63} ~ $2^{63}-1$

3. **浮点型：不精确**

表2-5浮点型数据类型		
类型	占用存储空间	表数范围
float	4字节	-3.403E38~3.403E38
double	8字节	-1.798E308~1.798E308

类

float类型又被称作单精度类型，尾数可以精确到7位有效数字，在很多情况下，float类型的精度很难满足需求。而double表示这种类型的数值精度约是float类型的两倍，又被称作双精度类型，绝大部分应用程序都采用double类型。浮点型常量默认类型也是double。

Java浮点类型常量有两种表示形式

- 十进制数形式，例如:3.14　　314.0　　0.314
- 科学记数法形式，如314e2　　314E2　　314E-2

> 对精度有要求，可以使用math包下的BigInteger和BigDecimal进行任意精度的运算
>
> 注意：不要使用浮点数进行比较

4. **字符型变量**

> 转义字符：

表2-6转义字符		
转义符	含义	Unicode值
\b	退格（backspace）	\u0008
\n	换行	\u000a
\r	回车	\u000d
\t	制表符（tab）	\u0009
\ "	双引号	\u0022
\ '	单引号	\u0027
\\	反斜杠	\u005c

5. **布尔型变量**

> 只占一位（1 bit），java不可以用0与1表示

6. **运算符分类**

算术运算符	二元运算符	+ , - , * , / , %
	一元运算符	++ , --
赋值运算符		=
扩展运算符		+= , -= , *= , /=
关系运算符		> , < , >= , <= , == , != instanceof
逻辑运算符		&&, ||, ! , ^
位运算符		& , | , ^ , ~ , >> , << , >>>
条件运算符		? :
字符串连接符		+

> **算数运算符的运算规则：**
>
> 两个操作数有一个为Long/Double，则结果为Long/Double；
>
> 如果操作数没有Long,则整型运算结果均为int，但浮点运算只有两个操作数都为float结果才为float

> **取模运算：**

> 结果符号与左边操作数相同：7%3=1,-7%3=-1

> 注意++a与a++在赋值时的区别：
>
> b=++a，先自加在赋值；
>
> b=a++，先将a赋值给b，a再自增；
>
> ***a=b+3 //a=a*(b+3)**

> **关系运算符：**
>
> ==，!= 基本与引用数据类型均适用
>
> “>”、>=、<、<=仅适用于数值类型

> 逻辑运算符：短路与（&&）短路或（||）能提高效率

> **位运算符：**
>
> 左移（<<），左移一位相当于成2；
>
> 右移（>>），右移一位相当于除2；

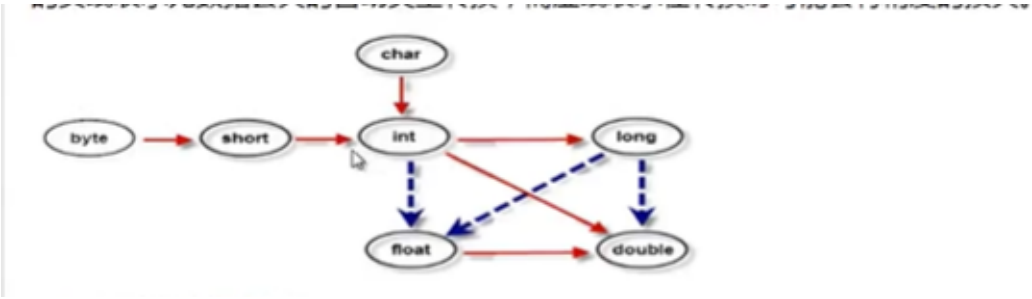
> **字符串连接符**
>
> “+”，+左右两边有一个字符串，则+当做字符串连接符使用，而不是当做数值运算符使用
>
> 若 a=“3”，b=2，c=5
>
> 则a+b=32，但b+c+a=73
>
> 需要注意的是，单引号内为char型，双引号为字符串String，char型加法是当做数值运算的

> **运算符优先级**
>
> 算数>关系>逻辑>赋值
>
> 在逻辑运算符中（逻辑非>逻辑与>逻辑或）
>
> 可以用括号来避免优先级带来的问题

优先级	运算符	类	结合性
1	()	括号运算符	由左至右
2	!、+（正号）、-（负号）	一元运算符	由左至右
2	~	位逻辑运算符	由右至左
2	++、--	递增与递减运算符	由右至左
3	*、/、%	算术运算符	由左至右
4	+、-	算术运算符	由左至右
5	<<、>>	位左移、右移运算符	由左至右
6	>、>=、<、<=	关系运算符	由左至右
7	==、!=	关系运算符	由左至右
8	&	位运算符、逻辑运算符	由左至右
9	^	位运算符、逻辑运算符	由左至右
10	|	位运算符、逻辑运算符	由左至右
11	&&	逻辑运算符	由左至右
12	||	逻辑运算符	由左至右
13	?:	条件运算符	由右至左
14	=、+=、-=、*=、/=、%=	赋值运算符、扩展运算符	由右至左

7. **自动类型转换**

> 容量小的数据类型可以自动转换为容量大的数据类型
>
> 红线无精度损失，但紫色虚线有进度损失



8. **强制类型转化**

> (type)var，会产生精度丢失

9. **存在的问题**

```
int money=10000000000;
int year = 20;//total*money=20亿超出int表示范围
int total=money*total;//发生溢出，无法获得正确值
long total2=money*total;//统一发生溢出，int*int其结果首先是int,然后才自动转化为long,转化中精度发生丢失
long total3=money*(long)year;//这样才是正确的
```

10. **Scanner获取键盘输入**

```
Scanner scanner = new Scanner(System.in);
int num = scanner.nextInt
此外还有scanner.nextLint等其他方法
```

11. **扩展**

> 关于定义了 int a; 不赋值情况下a的值:
>
> 如果a是类的成员变量，那么a是0，如果a是临时变量，则不会是0，输出会报错

流程控制语句

1. **类型**

> 顺序，选择，循环

2. **一些知识点**

> math.ramdom返回的是【0,1）之间的随机数
>
> if做单选则可以不写{}，但一般不会不写
>
> switch在一个case中若没有break，将会执行其后面的其他case，知道碰到break或者结尾
>
> break强制退出循环，不执行剩下的循换语句，continue退出本次循环，还将继续执行其他循环

方法

1. 一些知识点:

> 普通方法的调用需要通过对象来调用
>
> java方法中的参数传递是值传递，传的是数据的副本
>
> 基本数据类型与引用数据类型的传递都是传的值的copy

2. 重载的条件

> 形参类型、形参个数、形参顺序三个出现任意一个不同，即构成重载；
>
> 只有返回值不同、形参名不同不构成方法的重载；
>
> 注：static方法调用时可以不用new一个对象，非static调用需new对象

递归

1. 一句话总结：自己调用自己
2. 递归结构

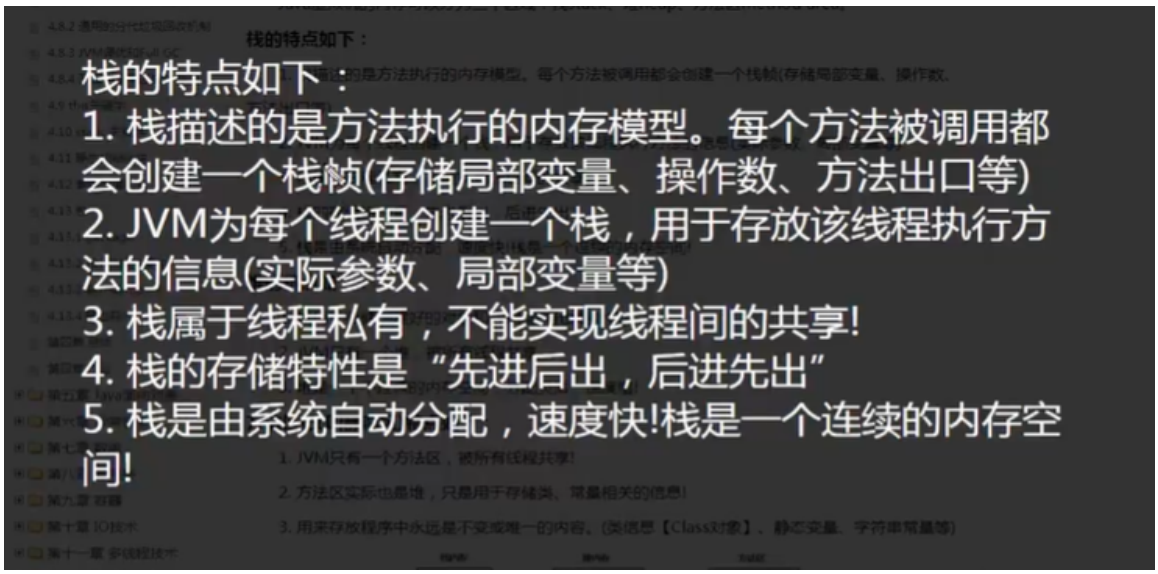
> 递归头：递归的出口
>
> 递归体：什么时候调用自身

面向对象的内存分析

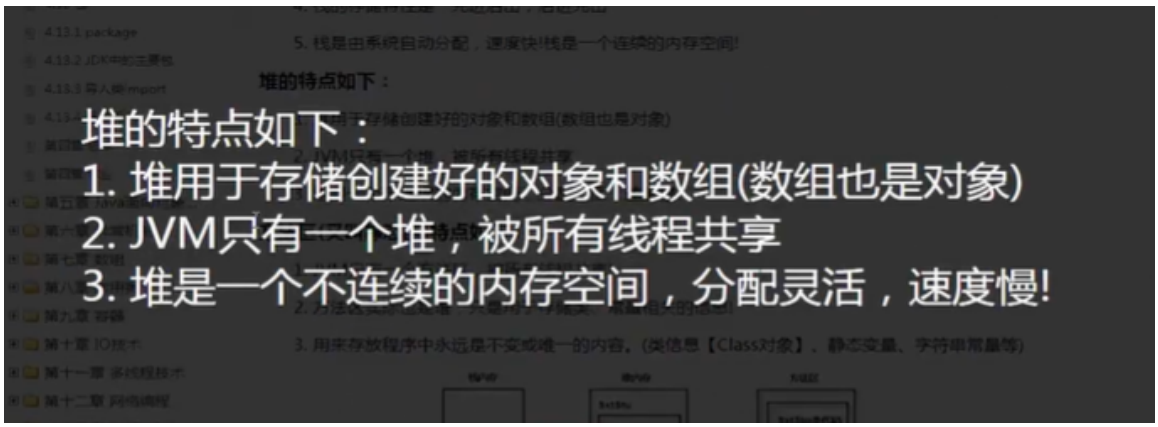
1. **jvm的内存划分：**

> 三个区域：栈stack，堆heap，方法区method area（方法区也在堆中，只是作用比较特殊）

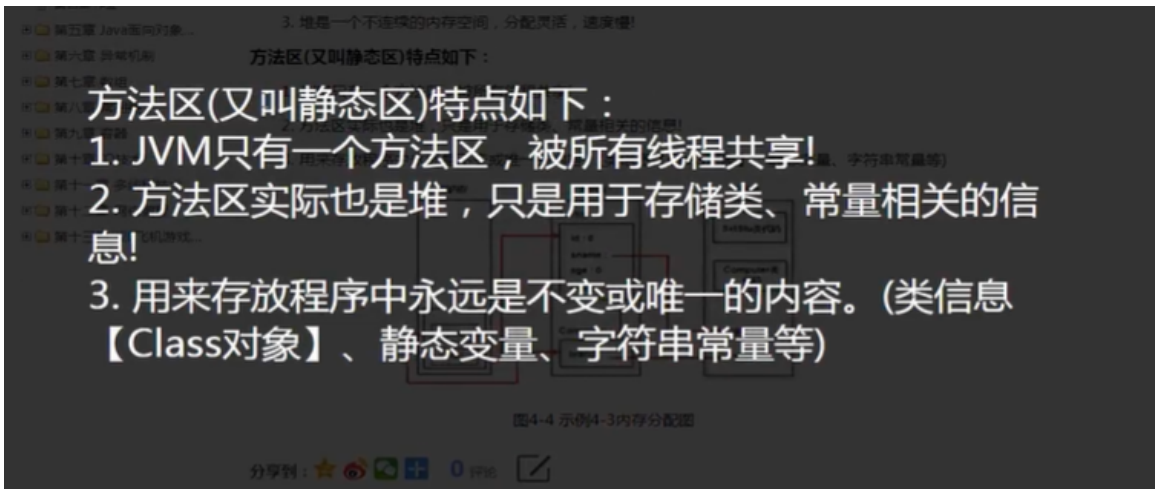
2. **栈：**

> 
>

3. **堆**



4. **方法区：**



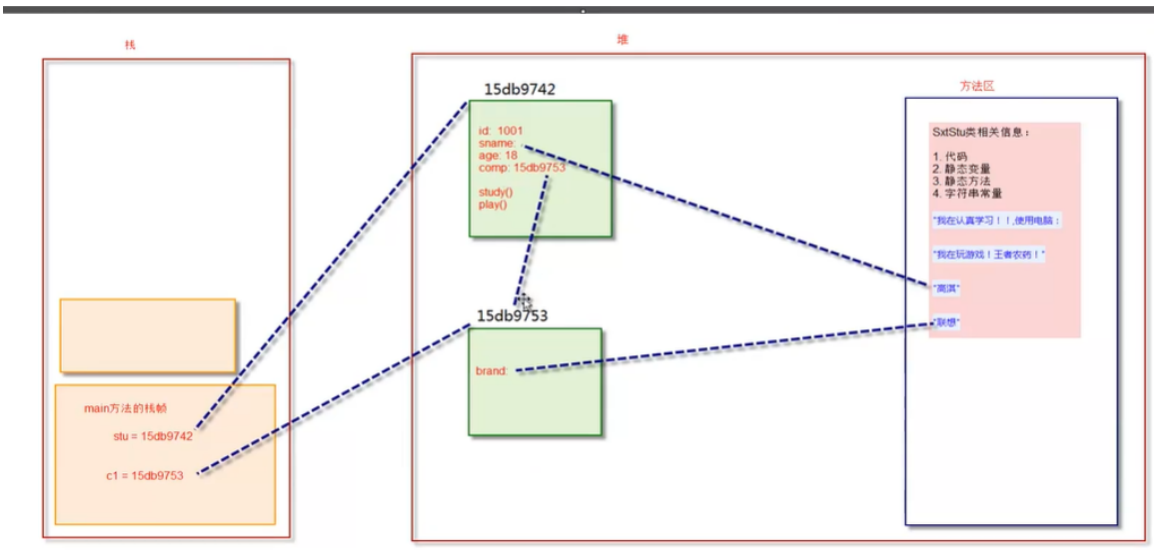
5. 代码执行的顺序：第64讲15min~~

首先，将类代码加载到内存方法区，并存储该类的静态变量，静态方法和常量（即所有不会变的内容）；

调用main方法，在栈中开辟main方法的栈帧，之后，每调用一个方法，都会建立该方法特定的栈帧；

执行方法时，new了一个对象，jvm对应在堆中存放该对象，并存储其信息（成员变量，方法）

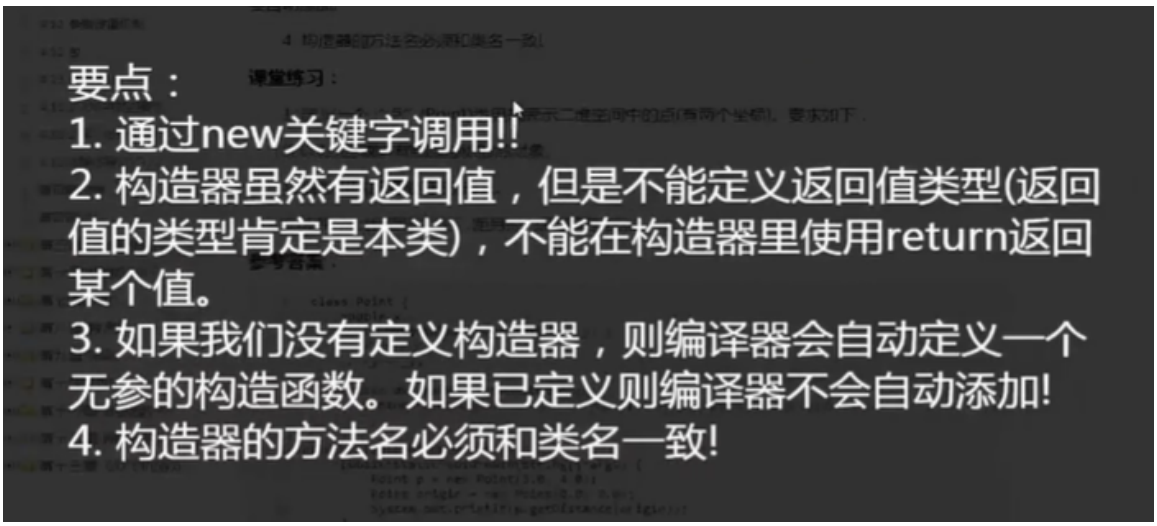
```
24   //程序执行的入口，必须要有
25   //javac  Sxtstu.java  ，  java Sxtstu
26   public static void main(String[] args) {
27      SxtStu  stu = new SxtStu();  //创建一个对象
28      stu.id=1001;
29      stu.sname= "高淇";
30      stu.age = 18;
31
32      Computer  c1 = new Computer();
33      c1.brand = "联想";
34
35      stu.comp = c1;
36
37      stu.play();
38      stu.study();
39
40   }
41 }
```



构造器

1. **特点**

> 如果自己定义了一个有参构造器，那么系统将不会自动添加无参构造器，要使用无参构造器需要自己自定



垃圾回收机制（GC garbage collection）

1. 基本事件

> 发现无用的对象；
>
> 回收无用的对象占用的内存空间；

2. 两个发现垃圾的算法（详解垃圾回收.md）

> 引用计数算法；
>
> 可达性分析算法；

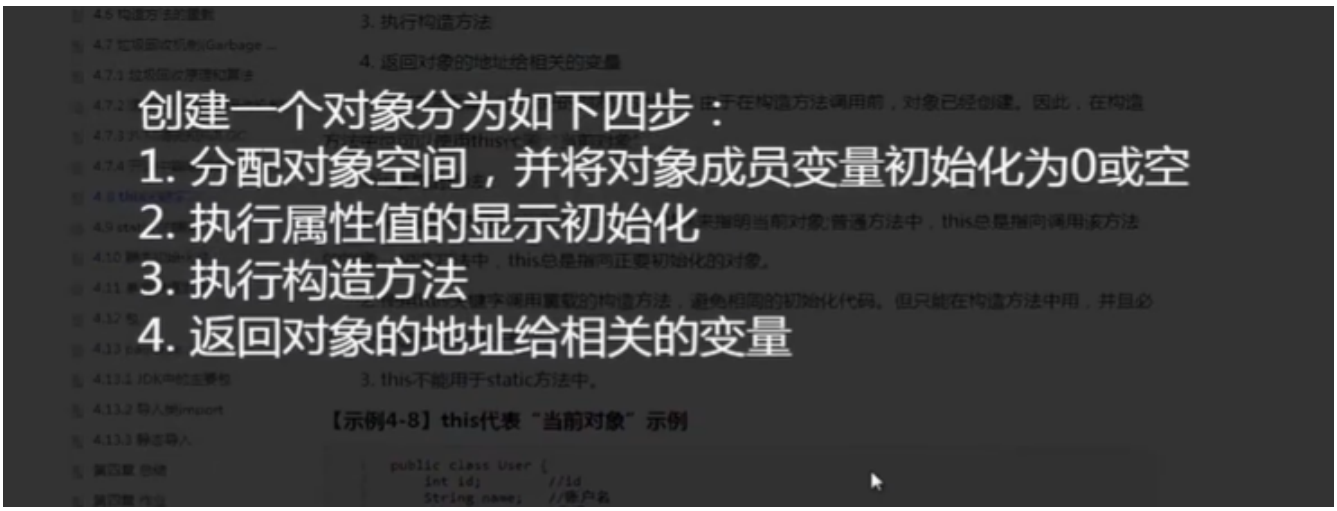
3. 通用的分代垃圾回收机制

> 即年轻代、年老代，持久代的划分，jvm将内存分为Eden、survivor、old分区，survivor还划分为from和to；
>
> 垃圾回收的流程：
>
> 1. 新创建的对象，多数放入Eden区；
> 2. 当Eden满了不能创建新的对象，触发minor GC，将无用对象清理掉，剩余对象放入survivor中的from区，同时清空Eden；
> 3. 当Eden再次满，同时将Eden中的不能清除的对象复制到from，然后将from区的不能清空的对象存入到to区，最后清空Eden和from区（from区与to区不存在实际差别，他们两在每次只需minorGC是会交换职责，如第一次minorGC将Eden和from区存货对象赋值到to区，那么第二次minorGC将会将Eden与to区的存活对象赋值到from区，依次类推）
> 4. 重复多次（默认最大15次），survivor区中任然没有被清理的对象，将会复制到old区中
> 5. 当old区满了（或达到某个比例），将会触发major GC,触发stop-the-world，清空old区
> 6. note:还有一个full GC,同时清理年轻代，年老代区域，成本较高，会对系统性能产生影响；导致full GC的原因：年老代写满；持久代写满；System.gc()的显示调用；上一次GC之后Heap的各域分配策略动态变化

开发中容易造成内存泄露的操作

1. 创建大量无用对象；
2. 静态集合类的使用：HashMap、Vector、List等；
3. 各种连接对象（IO流对象，数据库连接对象、网络连接对象）未关闭；
4. 监听器的使用：释放对象时未释放相应的监听器；

创建一个对象的步骤



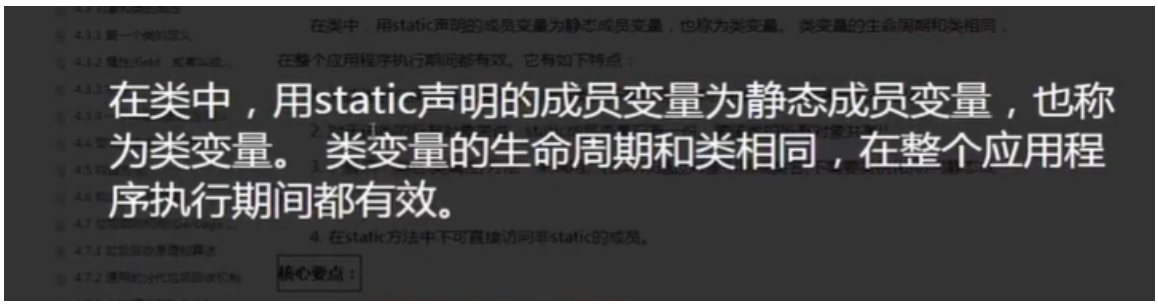
this指代的就是当前的对象

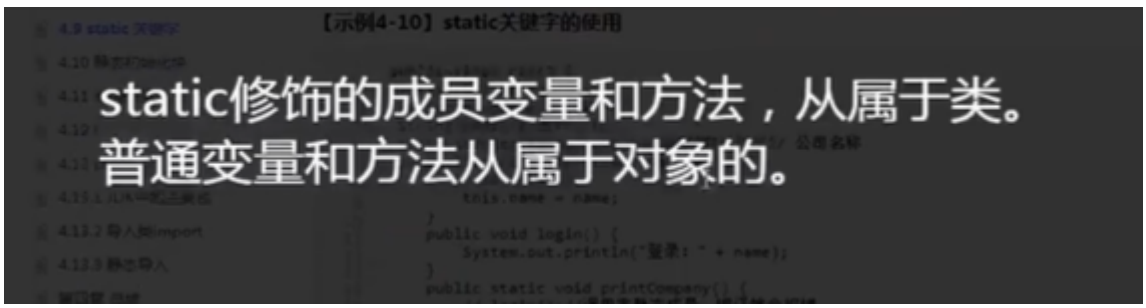
在构造器a中调用其他重载构造器b，使用this(参数列表)，且必须将此调用放在a构造器内部的第一句；

在static方法中无法调用this，因为static方法位于heap的method area中，属于类的信息，不存在对象；

static关键字

1. 定义与特征





2. 主义

> 在静态方法中，无法调用非静态的方法或变量（因为他们所处的区域不同）；
>
> 但是非静态方法可以调用静态的方法与静态变量，因为非静态方法属于对象，对象是类的实例化；
>
> 类就像图纸，而对象就像汽车，有了图纸不一定有车，但是有了车就一定有图纸；
>
> 注意：static所修饰的是静态变量，而不是常量。所以其值是可修改的且始终保持最新的值，其成为静态是因为他的值不会随着方法的进入退出而改变，且其在内存中始终只占一块内存；只有final修饰的量才叫常量，static final修饰的叫做静态常量；

静态初始化块

1. 作用：

> 用于类的初始化操作（类似于构造方法用于对象的初始化），在静态初始化块中不能直接访问非static成员；

2. 执行顺序：

> 依次向上追溯其父类，先执行其父类的静态初始化块，在向下执行子类的静态初始化块（构造方法的执行顺序同）

参数传递机制

1. 定义

> java中所有的参数传递都是值传递，即传递的是值的copy副本

2. 基本数据类型传递

> 传递的是值的副本，副本不会改变原件

3. 引用类型的传递

> 也是传递的值的副本，但是引用类型传的是对象的地址，因此副本和原参数都指向同一个地址，改变副本指向地址的对象的值，也意味着原参数指向地址的值改变

包

1. 包的作用类似于文件夹对于文件的作用，用来管理类
2. 包名：

> 域名倒着写，再加上模块名，便于内部管理

3. import的小问题

> 比如多个包下均存在Date类，那么导入以及使用时就会出现小问题，那么可以在使用时通过java.util.Date date = new java.utilDate()的方法来避免，同时增加可读性

4. 静态导入

> 用于导入指定类的静态属性，方便直接使用静态属性
>
> 语法：import static 包名.类名.静态属性

面向对象的提高

1. **继承**

> 实现代码重用，使用extends关键字实现继承；
>
> java中的类只有单继承，c++中的类可以多继承，但是java的接口可以多继承；
>
> 子类继承父类，可以得到父类的全部属性和方法，除了父类的构造方法；
>
> 一个类没有继承，那么他的父类默认为Object类，可以说，所有的类都是Object类的子类；
>
> instanceof是一个二元运算符，左边是对象，右边是类，当对象是右面类或其子类所创建的对象时，返回true，否则返回false；

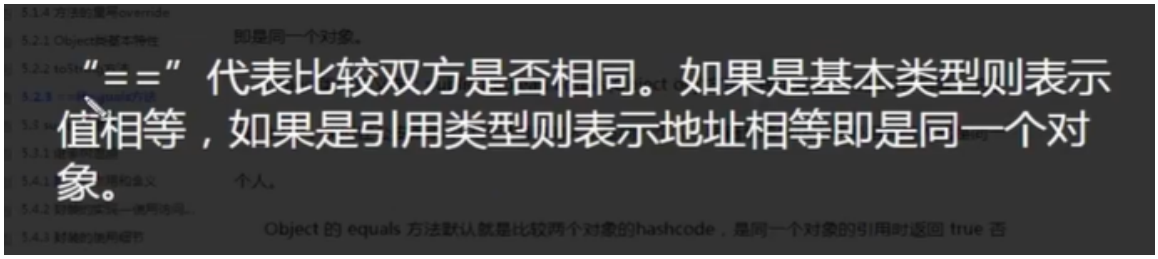
2. override的访问权限

> "==":方法名，参数列表相同；
>
> “<=”:返回值类型和声明异常类型，子类小于等于父类；
>
> “>=”:访问权限，子类大于等于父类

3. Object类

> 所有类的父类；

4. ==与equals方法：



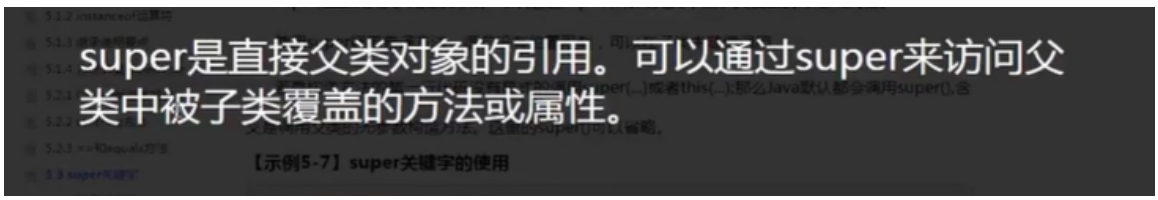
equals方法是由Object类提供的一个方法：

equals源码：

```
public boolean equals(Object obj){
    return (this==obj);
}
```

某种程度上说，equals方法用来判断两个对象的引用是否相同，但是很多类重写了equals方法用来判断值相同

5. super



通过super.function，super.value来访问父类在子类被覆盖的方法和变量；

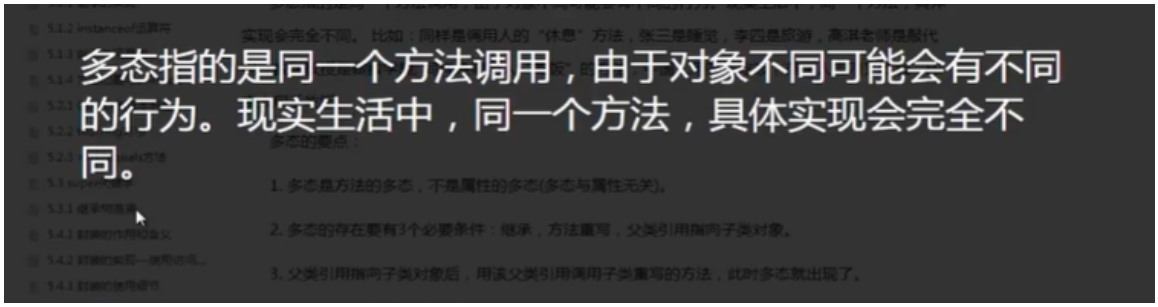
构造方法第一句总是super（）来调用父类的构造方法，所以流程就是先向上追溯到object，然后依次向下执行类的初始化块和构造方法，直到当前子类

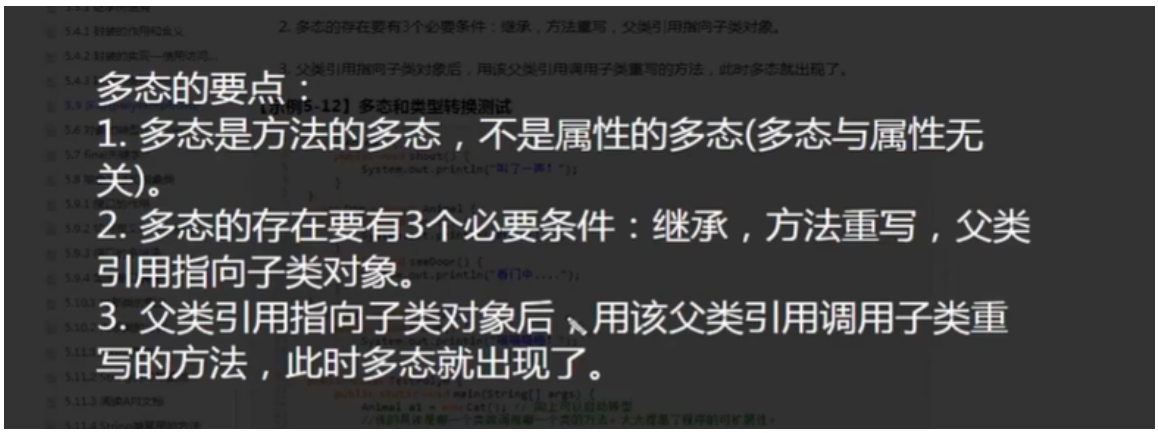
6. 封装的细节

类的属性处理：一般使用private，提供setter/getter方法访问

7. 多态

定义：



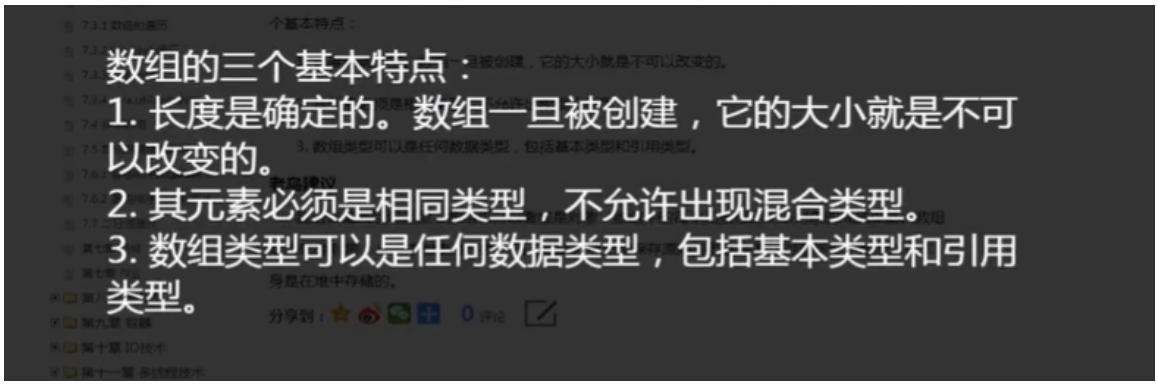


8. 转型

> 向上转型是自动的；
>
> 向下转型是

数组

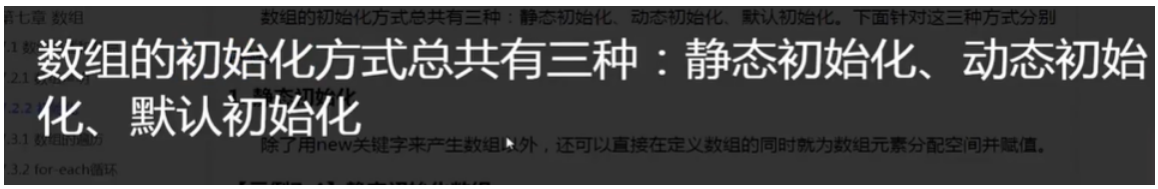
1. 特点



2. 申明方式

> type[] array;
>
> type array[];

3. 初始化



> 静态初始化：int[] a ={1,2,3,4,5};
>
> 默认初始化：int[] b = new int[3];int型数组默认初始化为0，Boolean默认初始化为false，string默认初始化为null；
>
> 动态初始化：通过下标直接赋值；

4. 数组的遍历：

> for循环；
>
> foreach：专门用来读取数组或集合中的所有元素，无法进行修改,针对int数组a
>
> for(int m : a){ syso(m); }即：
>
> for(数据类型：对象)

抽象方法

1. 定义

> abstract关键字

2. 要点：

> 没有方法体；
>
> 包含抽象方法的类必须是抽象的；
>
> 要求子类必须实现抽象方法

3. 意义

> 为子类提供统一规范的模板，子类必须实现相关抽象方法

接口

1. 定义

> 接口内不存在任何实现，其内部所有方法都是抽象的
>
> **声明格式：**
>
> ```
> [访问修饰符] interface 接口名 [extends 父接口1，父接口2…] {
> 常量定义；
> 方法定义；
> }
> ```
>
> **定义接口的详细说明：**
>
> 1. 访问修饰符：只能是public或默认。
> 2. 接口名：和类名采用相同命名机制。
> 3. extends：接口可以多继承。
> 4. 常量：接口中的属性只能是常量，总是：public static final 修饰。不写也是。
> 5. 方法：接口中的方法只能是：public abstract。 省略的话，也是public abstract。

2. 要点

> 1. 子类通过implements来实现接口中的规范。
>
> 2. 接口不能创建实例，但是可用于声明引用变量类型。
>
> 3. 一个类实现了接口，必须实现接口中所有的方法，并且这些方法只能是public的。
>
> 4. JDK1.7之前，接口中只能包含静态常量、抽象方法，不能有普通属性、构造方法、普通方法。
>
> 5. JDK1.8后，接口中包含普通的静态方法。

内部类

1. 定义：详情见网页内部类的分类

> 在Java中内部类主要分为成员内部类(非静态内部类、静态内部类)、匿名内部类、局部内部类

2. 成员非静态内部类：

> 特点：
>
> i. 非静态内部类必须寄存在一个外部类对象里。因此，如果有一个非静态内部类对象那么一定存在对应的外部类对象。非静态内部类对象单独属于外部类的某个对象。
>
> ii. 非静态内部类可以直接访问外部类的成员，但是外部类不能直接访问非静态内部类成员。
>
> iii. 非静态内部类不能有静态方法、静态属性和静态初始化块。
>
> iv. 外部类的静态方法、静态代码块不能访问非静态内部类，包括不能使用非静态内部类定义变量、创建实例。
>
> 形式：

```
class Outer {
 private int age = 10;
 //内部类inner
 class Inner {
     int age = 20;
     public void show() {
         int age = 30;
         System.out.println("内部类方法里的局部变量age:" + age);// 30
         System.out.println("内部类的成员变量age:" + this.age);// 20
         System.out.println("外部类的成员变量age:" + Outer.this.age);// 10
     }
 }
}
```

> 如何创建内部类对象

```
Outer.Inner inner = new Outer().new Inner()//这是和静态内部类的区别
```

3. 成员静态内部类

特点：

> 1. 当一个静态内部类对象存在，并不一定存在对应的外部类对象。 因此，静态内部类的实例方法不能直接访问外部类的实例方法。
>
> 2. 静态内部类看做外部类的一个静态成员。 因此，外部类的方法中可以通过： “静态内部类.名字”的方式访问静态内部类的静态成员，通过 new 静态内部类()访问静态内部类的实例。

构造：

```
class Outer{
 //相当于外部类的一个静态成员
 static class Inner{
 }
}
public class TestStaticInnerClass {
 public static void main(String[] args) {
     //通过 new 外部类名.内部类名() 来创建内部类对象,不在依托于外部类
     Outer.Inner inner =new Outer.Inner();
 }
}
```

4. 匿名内部类

特点：适合那种只需要使用一次的类，比如：键盘监听操作等等。

语法：

```
new  父类构造器(实参类表) \实现接口 () {
        //匿名内部类类体!
}
```

使用：

```
this.addWindowListener(new WindowAdapter(){
     @Override
     public void windowClosing(WindowEvent e) {
         System.exit(0);
     }
 }
);
this.addKeyListener(new KeyAdapter(){
     @Override
     public void keyPressed(KeyEvent e) {
         myTank.keyPressed(e);
     }
```

```
        @Override
        public void keyReleased(KeyEvent e) {
            myTank.keyReleased(e);
        }
    }
);
```

注意：

匿名内部类没有访问修饰符；

匿名内部类没有构造方法。因为它连名字都没有那又何来构造方法呢；

5. 方法内部类：

使用极少

```
public class Test2 {
    public void show() {
        //作用域仅限于该方法
        class Inner {
            public void fun() {
                System.out.println("helloworld");
            }
        }
        new Inner().fun();
    }
    public static void main(String[] args) {
        new Test2().show();
    }
}
```

String类

1. 特点

不可变字符序列，只能被初始化一次，每次更改都是创建一个新的对象并引用；

位于lang包

String是对象，不是基本数据类型

2. 常用方法

方法	解释说明
char charAt(int index)	返回字符串中第index个字符
boolean equals(String other)	如果字符串与other相等，返回true；否则，返回false。
boolean equalsIgnoreCase(String other)	如果字符串与other相等（忽略大小写），则返回true；否则，返回false。
int indexOf(String str)	返回从头开始查找第一个子字符串str在字符串中的索引位置。如果未找到子字符串str，则返回-1。
lastIndexOf()	返回从末尾开始查找第一个子字符串str在字符串中的索引位置。如果未找到子字符串str，则返回-1。
int length()	返回字符串的长度。
String replace(char oldChar,char newChar)	返回一个新串，它是通过用 newChar 替换此字符串中出现的所有oldChar而生成的。
boolean startsWith(String prefix)	如果字符串以prefix开始，则返回true。
boolean endsWith(String prefix)	如果字符串以prefix结尾，则返回true。
String substring(int beginIndex)	返回一个新字符串，该串包含从原始字符串beginIndex到串尾。
String substring(int beginIndex,int endIndex)	返回一个新字符串，该串包含从原始字符串beginIndex到串尾或endIndex-1的所有字符。
String toLowerCase()	返回一个新字符串，该串将原始字符串中的所有大写字母改成小写字母。
String toUpperCase()	返回一个新字符串，该串将原始字符串中的所有小写字母改成大写字母。
String trim()	返回一个新字符串，该串删除了原始字符串头部和尾部的空格。

数组的拷贝

1. 方法

> arraycopy(src,start,target,targetpos,length)
>
> 从src的start位置开始，赋值length个字符到target从targetpos开始的位置

2. java.util.Arrays工具类的使用

> Arrays.toString(Array[])：输出
>
> Arrays.sort(Array[])：排序
>
> 数组元素是引用类型的排序需要基础Comparable接口重写sort方法
>
> binarysearch()：二分查找

二维数组

1. 定义

> int [] [] a=new int[10] [];
>
> 赋值 a[0]=new int[]{20,30};

算法

1. 冒泡排序

> 两两比较，一轮比较后将比出一个最大/最小的，第二轮从第一个比较到倒数第二个，第三轮从第一个比较到倒数第三个，以此类推
>
> 优化：设置flag，如果以此循环没有发生交换，则排序完成，跳出循环

2. 二分查找/折半查找

> 前提：数组是排好序的
>
> 顺序：找到中轴，将待查询数与中轴比较，划分查找的左右区间

常用类

1. 包装类

基本数据类型	包装类
byte	Byte
boolean	Boolean
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

> 除了Character和Boolean包装类都是Number抽象类的子类；
>
> 常用方法：
>
> 基本类型转化为包装类：new Integer(3)/Integer.valueOf(3);
>
> 包装类转化为基本类型：int a = b.intValue();//Integer b;
>
> 字符串转包装类：Integer e = new Integer("999");
>
> 包装类转字符串：String str = e.toString();

> 包装类包含的常见常量：
>
> MAX_VALUE、MIN_VALUE

2. 自动装箱与拆箱

> Integer a=234 // Integer a = Integer.valueOf(234); 装箱
>
> int b = a; // int b = a.intvalue(); 拆箱
>
> Integer c=null; int d = c; //自动拆箱，空指针异常

3. 包装类的缓存问题：

> 整型、char类型所对应的包装类，在自动装箱时，对于-128~127之间的值会进行缓存处理，其目的是提高效率。
>
> 即包装类在初始化时，通过静态块将一定范围内的值自动封装成包装类对象缓存数组放入堆中，调用valueOf的时候首先从缓存数组中检查，如果在缓存数组中可以直接引用而提高效率
>
> 测试代码：

```
public class Test3 {
 public static void main(String[] args) {
     Integer in1 = -128;
     Integer in2 = -128;
     System.out.println(in1 == in2);//true 因为123在缓存范围内
     System.out.println(in1.equals(in2));//true
     Integer in3 = 1234;
     Integer in4 = 1234;
     System.out.println(in3 == in4);//false 因为1234不在缓存范围内
     System.out.println(in3.equals(in4));//true
 }
}
```

String类源码相关内容

1. 定义

> String 类对象代表不可变的Unicode字符序列，因此我们可以将String对象称为“不可变对象”。 那什么叫做“不可变对象”呢?指的是对象内部的成员变量的值无法再改变。我们打开String类的源码；
>
> 其核心是private final char value[];只能在初始化时赋值一起，无法再更改；
>
> 字符串比较值用equals方法，==比较的是引用是否相同

2. StringBuilder和StringBuffer和String

> StringBuilder和StringBuffer继承自abstractStringBuilder,其核心是char value[]，所以是可变字符序列，修改前后是同一引用，地址相同；
>
> StringBuffer线程安全，效率低；
>
> StringBuilder线程不安全，但效率高；

3. StringBuilder和StringBuffer常用方法

> append()：末尾追加字符；
>
> reverse()：翻转；
>
> setCharAt(Index,value)：设置指定位置处的字符；
>
> insert(index，char)：指定位置插入字符 该方法可以链式调用，核心是它调用了return this，将自己返回；
>
> delete（start,end）：删除指定区间的字符

4. 可变字符序列与不可变字符序列使用陷阱

> 使用String进行字符串的拼接，当拼接的次数变多，产生很多的无用对象，浪费空间与时间；
>
> 应该使用StringBuilder或StringBuffer进行字符串的拼接

时间处理相关

详情查看网页：https://www.sxt.cn/Java_jQuery_in_action/eight-timeprocessing.html

Date类

1. 获取当前时间的毫秒数（距1970-1-1 0:0:0）System.currentTimeMillis()，是一个long类型的值
2. 初始化

```
Date date = new Date();

//源码

public Date() {
    this(System.currentTimeMillis());//可见仍然是上述方法
}
```

DateFormate接口

1. 将时间按照指定格式转化为字符串

```
DateFormat df = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");//传入参数为日期的格式化，具体可见api
        String str = df.format(new Date());//传入一个日期进行格式化，结果为String
        System.out.println(str);
//输出结果2019-07-21 01:53:47
```

2. 将指定字符串转化为时间对象

```
DateFormat df2 = new SimpleDateFormat("yyyy年MM月dd日 hh时mm分ss秒");
Date date = df2.parse("1983年05月10日 10时35分49秒");
System.out.println(date);
//结果Tue May 10 10:35:49 CST 1983
```

3. 测试其他格式

```java
//查看今天2019-7-21是今年的多少天
DateFormat df3 = new SimpleDateFormat("D");
String str2 = df3.format(new Date());
System.out.println(str2);
//结果202

//查看今天2019-7-21是今年的多少周
DateFormat df4 = new SimpleDateFormat("w");
String str3 = df4.format(new Date());
System.out.println(str3);
//结果 30
```

Calendar接口

1. 相关使用方法

```java
 Calendar calendar = new GregorianCalendar(2999,10,9,22,10,50);//构造方法有很多，此处为年月日时分秒
        int year = calendar.get(Calendar.YEAR);
        int month = calendar.get(Calendar.MONTH);
        int day = calendar.get(Calendar.DATE);//获得几号,也可用DAY_OF_MONTH
        int weekday = calendar.get(Calendar.DAY_OF_WEEK);//星期几，1-7对应
        System.out.println(year+" "+month+" "+" "+day+" "+weekday);//0-11表示对应的月份

        //设置日期相关元素
        Calendar calendar1 = new GregorianCalendar();//不添加参数，结果为当前时间的日期
        calendar1.set(Calendar.YEAR,8012);//设置当前年份


        //日期的计算
        Calendar c3 = new GregorianCalendar();
        c3.add(Calendar.DATE,100);//在当前日期上加100天,参数一是待修改参数，参数二是实际修改的数量
        System.out.println(c3);


        //日期对象与时间对象的转化
        Date d4 = c3.getTime();//calendar转date
        System.out.println(d4);
        Calendar c4 = new GregorianCalendar();
        c4.setTime(new Date());//date转calendar
        System.out.println(c4);
```

2. calendar接口的格式化输出日期

```java
public static void printCalendar(Calendar c){

        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH)+1;//1-12
        int day = c.get(Calendar.DATE);
```

```
        int weekDay = c.get(Calendar.DAY_OF_WEEK)-1;
        String weekDay2 = weekDay==0?"日":weekDay+"";

        int hour = c.get(Calendar.HOUR);
        int min = c.get(Calendar.MINUTE);
        int sec = c.get(Calendar.SECOND);


 System.out.println(year+"年"+month+"月"+day+"日"+hour+"时"+min+"分"+sec+"秒 星期"+weekDay2);


    }

Calendar c4 = new GregorianCalendar();
c4.setTime(new Date());//date转calendar
calendarTest.printCalendar(c4);
//结果 2019年7月21日2时29分13秒 星期日
```

3. calendar小应用，生成输入日期那一个月的日历

```
 System.out.println("请输入日期: (格式为: 2020-09-15) ");
        Scanner reader = new Scanner(System.in);
        String str = reader.nextLine();
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        Date date = dateFormat.parse(str);//字符串转化为格式化日期对象,两者格式需相同
        Calendar calendar = new GregorianCalendar();
        calendar.setTime(date);
        int currentDay = calendar.get(Calendar.DAY_OF_MONTH);
        System.out.println("日\t一\t二\t三\t四\t五\t六\t");
        calendar.set(Calendar.DAY_OF_MONTH,1);

        for (int i=0;i<calendar.get(Calendar.DAY_OF_WEEK)-1;i++){//先计算出当前月的第一天是周几，将其放置于正确的位置上
            System.out.print("\t");
        }
        int maxDay = calendar.getActualMaximum(Calendar.DATE);//获取当前月份一共多少天
        for (int i=1;i<=maxDay;i++){
            if (calendar.get(Calendar.DAY_OF_MONTH)==currentDay){
                System.out.print(calendar.get(Calendar.DAY_OF_MONTH)+"*\t");
            }else{
                System.out.print(calendar.get(Calendar.DAY_OF_MONTH)+"\t");
            }
            if (calendar.get(Calendar.DAY_OF_WEEK)== Calendar.SATURDAY){//每逢周六换行
                System.out.println();
            }
            calendar.add(Calendar.DAY_OF_MONTH,1);
        }
```

一次输入的结果

> 请输入日期：（格式为：2020-09-15） 2019-7-21 日 一 二 三 四 五 六 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21* 22 23 24 25 26 27 28 29 30 31

4. math包下的方法

```
System.out.println(Math.ceil(3.2));//上取整 4
System.out.println(Math.floor(3.2));//下取整 3
System.out.println(Math.round(3.2));//四舍五入 3
System.out.println(Math.round(3.8));//四舍五入 4
//绝对值、开方、a的b次幂等操作
System.out.println(Math.abs(-45));//绝对值 45
System.out.println(Math.sqrt(64));//开方 8
System.out.println(Math.pow(5, 2));//5的平方 25
System.out.println(Math.pow(2, 5));//2的5次方 32
//Math类中常用的常量
System.out.println(Math.PI);//常数π 3.141592653589793
System.out.println(Math.E);//常数E 2.718281828459045
//随机数
System.out.println(Math.random());// [0,1)之间的随机数 0.7504092646240315

```

5. random类下的随机数功能

```
Random rand = new Random();
//随机生成[0,1)之间的double类型的数据
System.out.println(rand.nextDouble());
//随机生成int类型允许范围之内的整型数据
System.out.println(rand.nextInt());
//随机生成[0,1)之间的float类型的数据
System.out.println(rand.nextFloat());
//随机生成false或者true
System.out.println(rand.nextBoolean());
//随机生成[0,10)之间的int类型的数据
System.out.println(rand.nextInt(10));
//随机生成[20,30)之间的int类型的数据
System.out.println(20 + rand.nextInt(10));
//随机生成[20,30)之间的int类型的数据（此种方法计算较为复杂）
System.out.println(20 + (int) (rand.nextDouble() * 10));
```

> 一次运行结果
>
> 0.7810988585807036 -1021351191 0.4829309 false 6 28 26

File类

1. File类的相关用法

```
File f = new File("d:/magic.txt");
System.out.println(f);//打印的是文件路径
f.renameTo(new File("d:/magic2.txt"));//对文件改名

```

```
System.out.println(System.getProperty("user.dir"));//当前用户的工作控空间

File f2 = new File("gg.txt");
f2.createNewFile(); //创建新文件，不加路径，将会添加到当前的用户空间

System.out.println("File是否存在："+f2.exists());
System.out.println("File是否是目录："+f2.isDirectory());
System.out.println("File是否是文件："+f2.isFile());
System.out.println("File最后修改时间："+new Date(f2.lastModified()));
System.out.println("File的大小："+f2.length());
System.out.println("File的文件名："+f2.getName());
System.out.println("File的目录路径："+f2.getPath());//获取绝对路径可以用
getabsolutePath()
```

> 运行结果：
>
> d:\magic.txt F:\JetBrains project\javaBaseTest File是否存在：true File是否是目录：false File是否是文件：true File最后修改时间：Sun Jul 21 15:33:25 CST 2019 File的大小：0 File的文件名：gg.txt File的目录路径：gg.txt
>
> 此外还有其他方法：删除delete()，

2. File类的mkdir与mkdirs的区别

```
File f2 = new File("d:/电影/华语/大陆");
boolean flag = f2.mkdir(); //目录结构中有一个不存在，则不会创建整个目录树
System.out.println(flag);//创建失败
//运行结果：false，因为d盘下不存在任何要创建的目录树中的任何一个

File f2 = new File("d:/电影/华语/大陆");
boolean flag = f2.mkdirs();//目录结构中有一个不存在也没关系；创建整个目录树
System.out.println(flag);//创建成功
```

3. 递归调用输出文件树

```
static void printFileTree(File file,int level){//level为层数
        for (int i=0;i<level;i++){
            System.out.print("-");
        }
        System.out.println(file.getName());
        if (file.isDirectory()){
            File[] files = file.listFiles();//如果当前文件是目录，列出其下所有文件
            for(File temp:files) {
                printFileTree(temp,level+1);
            }
        }
    }

File file = new File("d:/电影");
showfileTree.printFileTree(file,0);
```

> 输出结果为：
>
> 电影 -华语 --见龙卸甲.txt --黄金甲.txt -好莱坞 --123.txt --456.txt

枚举

有必要定义一组常量时建议定义一组枚举

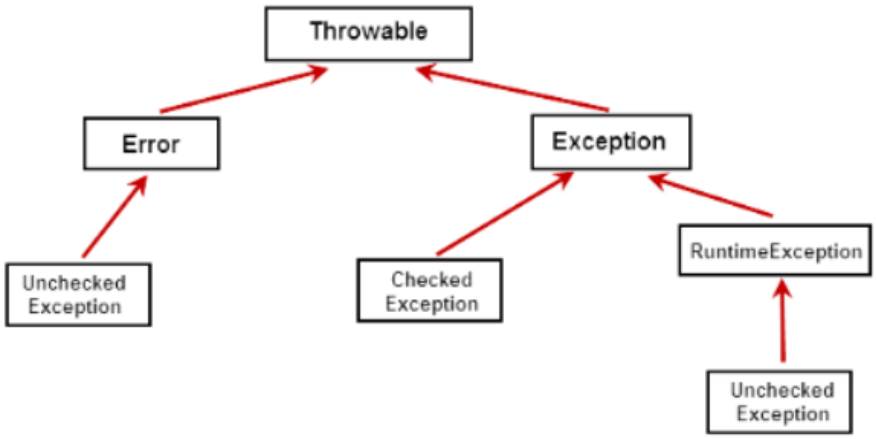
```
enum Season {
    SPRING, SUMMER, AUTUMN, WINDER
}

int a = new Random().nextInt(4); // 生成0, 1, 2, 3的随机数
       switch (Season.values()[a]) {  //枚举类型取值
       case SPRING:
            System.out.println("春天");
            break;
       case SUMMER:
            System.out.println("夏天");
            break;
       case AUTUMN:
            System.out.println("秋天");
            break;
       case WINDTER:
            System.out.println("冬天");
            break;
  }

//枚举类型的的对象
Season season = Season.SPRING
```

异常EXCEPTION

1. 分类：



2. Exception

> Exception分为RuntimeException和CheckedException

3. 运行时异常RuntimeException

> 派生于RuntimeException的异常，如被 0 除、数组下标越界、空指针等，其产生比较频繁，处理麻烦，如果显式的声明或捕获将会对程序可读性和运行效率影响很大。 因此由系统自动检测并将它们交给缺省的异常处理程序(用户可不必对其处理)。
>
> 这类异常通常是由编程错误（逻辑处理）导致的，所以在编写程序时，并不要求必须使用异常处理机制来处理这类异常,经常需要通过增加"逻辑处理来避免这些异常

分类：

> ArithmeticException异常：试图除以0 解决：增加预先处理
>
> NullPointerEXception异常：空指针异常，对象为空时调用其属性或方法 解决：增加非空判断
>
> ClassCastException异常：强制转型异常
>
> ArrayIndexOutofBoundsException异常：数组越界
>
> NumberFormatException异常：数字格式化异常
>
> 这些异常需要我们进行判断与处理

4. CheckedException:

> 所有不是RuntimeException的异常，统称为Checked Exception，又被称为"已检查异常"，如IOException、SQLException等以及用户自定义的Exception异常。 这类异常在编译时就必须做出处理，否则无法通过编译。
>
> 处理时需要通过try-catch捕获或者通过throws申明；

5. 异常的捕获

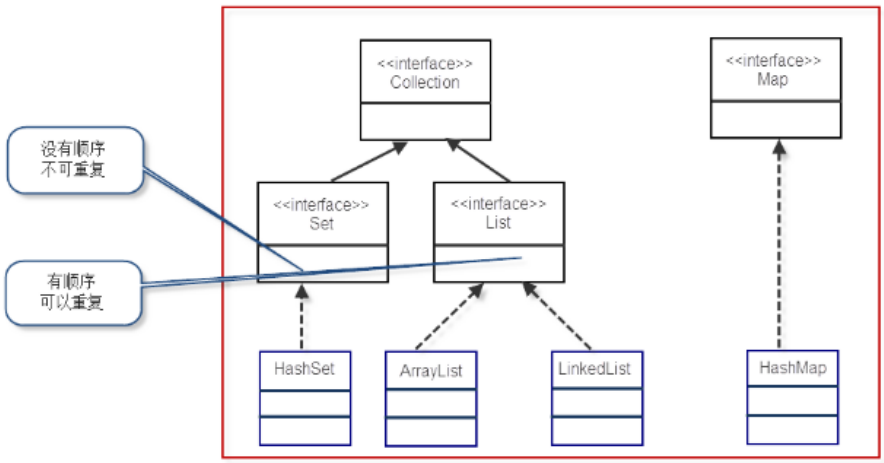
> 通过try-catch捕获的异常存在父子关系时，需要子类在前，父类在后进行catch，否则子类异常将会提前被父类捕获，无法被准确捕捉；
>
> 通过throws抛出异常，则谁调用它，谁需要对它抛出的异常进行处理

容器

1. 定义：

> 用来容纳和管理数据即存放其他对象的对象,数组就是一种容器，更强大的容器也叫做Collection

2. 分类



3. 泛型Generic

> 它可以帮助我们建立类型安全的集合。在使用了泛型的集合中，遍历时不必进行强制类型转换。JDK提供了支持泛型的编译器，将运行时的类型检查提前到了编译时执行，提高了代码可读性和安全性;
>
> 泛型的本质就是"数据类型的参数化"。 我们可以把"泛型"理解为数据类型的一个占位符(形式参数)，即告诉编译器，在调用泛型时必须传入实际类型。
>
> 如果初始化时未指定泛型，则默认为object，即可容纳任何类型
>
> 申明:

```
class MyCollection<E> {// E:表示泛型;
Object[] objs = new Object[5];

public E get(int index) {// E:表示泛型;
  return (E) objs[index];
}
public void set(E e, int index) {// E:表示泛型;
  objs[index] = e;
}
}
```

4. Collection的相关方法使用，collection下有list，set两个子接口

```
List<String> list01 = new ArrayList<>();
        list01.add("aa");
        list01.add("bb");
        list01.add("cc");

        List<String> list02 = new ArrayList<>();
        list02.add("aa");
        list02.add("dd");
```

```
        list02.add("ee");
        System.out.println("list01:"+list01);

        list01.addAll(list02);//将list02的所有元素加入到list01
        System.out.println("list01:"+list01);

        list01.removeAll(list02);//移除list01与list02的交集元素
        System.out.println("list01:"+list01);

        list01.addAll(1,list02);//从list01的指定位置开始插入list02的全部值
        System.out.println("list01:"+list01);

        list01.retainAll(list02);//取两个list共同的元素，移除非交集元素
        System.out.println("list01:"+list01);

/*运行结果
list01:[aa, bb, cc]
list01:[aa, bb, cc, aa, dd, ee]
list01:[bb, cc]
list01:[bb, aa, dd, ee, cc]
list01:[aa, dd, ee]
此外还有简单方法，add，remove，contains，size，isempty等，见名知意
*/
```

5. 关于List

> list有序，且可重复（允许e1.equals(e2)的元素重复加入容器），可根据索引访问元素；
>
> 三个实现类：Arraylist底层数组实现；LinkedList底层双向链表实现；vector也是数组实现，但是加了synchronized标记线程安全

带索引重载方法的使用

```
System.out.println(list);
list.add(2, "我");//指定位置添加
System.out.println(list);
list.remove(2);//指定位置移除
System.out.println(list);
list.set(2, "你");//指定位置替换
System.out.println(list);
list.add("你");
System.out.println(list.get(2));//获取指定位置的值
System.out.println(list);//获取指定位置的值
System.out.println(list.indexOf("你"));//查询第一次出现的索引
System.out.println(list.lastIndexOf("你"));//从末尾向前查询首次出现的位置
/*输出结果
[A, B, C, D]
[A, B, 我, C, D]
[A, B, C, D]
[A, B, 你, D]
你
[A, B, 你, D, 你]
2
4
```

```
*/
```

6. ArrayList的动态扩容源码

```
private void grow(int minCapacity) {
        // overflow-conscious code
        int oldCapacity = elementData.length;
        int newCapacity = oldCapacity + (oldCapacity >> 1);
        //容量扩充为原来的1.5倍
        if (newCapacity - minCapacity < 0)
            newCapacity = minCapacity;
        if (newCapacity - MAX_ARRAY_SIZE > 0)
            newCapacity = hugeCapacity(minCapacity);
        // minCapacity is usually close to size, so this is a win:
        elementData = Arrays.copyOf(elementData, newCapacity);//elementData是旧数组，newCapacity是新数组的长度
    }
```

7. map

> 通过键值对来存储，键对象不能重复，如果重复（通过equals方法判断），新的value将会覆盖旧的value
>
> 实现类有HashMap,TreeMap,HashTable,currentHashMap，Properties等

常用方法测试

```
Map<Integer,String> map = new HashMap<>();
map.put(1,"张三");
map.put(2,"张四");
map.put(3,"王五");
System.out.println(map);
System.out.println(map.size());
System.out.println(map.isEmpty());
System.out.println(map.containsKey(2));
System.out.println(map.containsValue("王二麻子"));
System.out.println(map.get(1));

Map<Integer,String> map2 = new HashMap<>();
map2.put(4,"老李");
map2.put(5,"老刘");
map.putAll(map2);
System.out.println(map);
map.put(3,"4564");
System.out.println(map);
/*结果如下
{1=张三, 2=张四, 3=王五}
3
false
true
false
张三
{1=张三, 2=张四, 3=王五, 4=老李, 5=老刘}
```

```
{1=张三, 2=张四, 3=4564, 4=老李, 5=老刘}
*/
```

8. 关于HashMap

> 底层由数组加链表的数据结构构成;
>
> 扩容：如果位桶数组中的元素达到(0.75*数组 length)， 就重新调整数组大小变为原来2倍大小。扩容的本质是定义新的更大的数组，并将旧数组内容挨个拷贝到新数组中;
>
> JDK8中，HashMap在存储一个元素时，当对应链表长度大于8时，链表就转换为红黑树;
>
> HashTable类和HashMap用法几乎一样，底层实现几乎一样，只不过HashTable的方法添加了synchronized关键字确保线程同步检查，效率较低,主要区别如下:
>
> HashMap: 线程不安全，效率高。允许key或value为null;
>
> HashTable: 线程安全，效率低。不允许key或value为null

9. TreeMap

> TreeMap是红黑二叉树的典型实现;
>
> TreeMap和HashMap实现了同样的接口Map，因此，用法对于调用者来说没有区别。HashMap效率高于TreeMap;在需要排序的Map时才选用TreeMap

使用

```
Map<Integer,String> map = new TreeMap<>();
map.put(20,"aa");
map.put(3,"bb");
map.put(6,"cc");
//按照Key自增的方式排序
for (Integer key:map.keySet()){
    System.out.println(key+": "+map.get(key));
}
/*输出结果为
3: bb
6: cc
20: aa
*/
```

```
//已知TreeMap是按照key来实现排序的，如果key是自定义类该如何排序呢?
//答：自定义类需要实现Comparable接口并实现compareTo方法，实例如下:
static class Employ implements Comparable<Employ>{
        int id;
        String name;
        double salary;

        public Employ(int id, String name, double salary) {
            this.id = id;
            this.name = name;
            this.salary = salary;
        }

        @Override
```

```
        public String toString() {
            return "Employ{" +
                    "id=" + id +
                    ", name='" + name + '\'' +
                    ", salary=" + salary +
                    '}';
        }

        @Override
        public int compareTo(Employ o) {//复数小于，正数大于，0等于
            if (this.salary>o.salary){
                return 1;
            }else if (this.salary<o.salary){
                return -1;
            }else{
                if (this.id>o.id){
                    return 1;
                } else if (this.id<o.id){
                    return -1;
                } else if (this.id>o.id){
                    return 0;
                }
            }
            return 0;
        }
    }

public static void main(String[] args) {
 Map<Employ,String> emp = new TreeMap<>();
        emp.put(new Employ(100,"ads",50000),"ads是个好小伙");
        emp.put(new Employ(200,"asadf",6000),"asadf不是个好小伙");
        emp.put(new Employ(150,"asadf2",6000),"asadf2不是个好小伙2");
        emp.put(new Employ(300,"adfe",1500),"adfe是个开心果");

        for (Employ ppp:emp.keySet()){
            System.out.println(ppp+":"+emp.get(ppp));
        }

}

/*输出结果为：按照自定义的排序规则，首先按照salary排序，salary相同则按照id排序，已知id不可能重复
Employ{id=300, name='adfe', salary=1500.0}:adfe是个开心果
Employ{id=150, name='asadf2', salary=6000.0}:asadf2不是个好小伙2
Employ{id=200, name='asadf', salary=6000.0}:asadf不是个好小伙
Employ{id=100, name='ads', salary=50000.0}:ads是个好小伙
*/

//compareTo使用时的返回值需要注意，返回复数表示小于，返回正数表示大于，返回0表示等于
```

10. set

> Set容器特点：无序、不可重复。无序指Set中的元素没有索引，我们只能遍历查找;不可重复指不允许加入重复的元素。更确切地讲，新元素如果和Set中某个元素通过equals()方法对比为true，则不能加入;甚至，Set中也只能放入一个null元素，不能多个；
>
> HashSet底层实际是用HashMap实现的，HashSet的add()方法，发现增加一个元素说白了就是在map中增加一个键值对，键对象就是这个元素，值对象是名为PRESENT的Object对象。说白了，就是“往set中加入元素，本质就是把这个元素作为key加入到了内部的map中”，由于map中key都是不可重复的，因此，Set天然具有“不可重复”的特性

11. TreeSet

> TreeSet底层实际是用TreeMap实现的，内部维持了一个简化版的TreeMap，通过key来存储Set的元素。 TreeSet内部需要对存储的元素进行排序，因此，我们对应的类需要实现Comparable接口。这样，才能根据compareTo()方法比较对象之间的大小，才能进行内部排序,详细使用可参考上方TreeMap的使用

12. 迭代器Iterator

```java
/*迭代器遍历list*/
List<String> list = new ArrayList<>();
list.add("aaa");
list.add("bbb");
list.add("ccc");

Iterator<String> iterator = list.iterator();
for (;iterator.hasNext();){//是否存在下一个元素
    String str = iterator.next();   //取出元素
    System.out.println(str);
}
/*结果:
aaa bbb ccc
*/

/*迭代器遍历set*/
Set<String> set = new HashSet<>();
set.add("123");
set.add("456");
set.add("789");
Iterator<String> iterator = set.iterator();
for (;iterator.hasNext();){//是否存在下一个元素
      tring str = iterator.next();   //取出元素
      System.out.println(str);
}
/*结果
123 456 789
*/
```

需要注意的是，迭代器遍历map有所不同，是通过现货区set，再通过迭代器遍历set来达到遍历的目的，共有两种方法

```java
/*法1*/
        Map<Integer,String> map = new HashMap<>();
        map.put(1,"123");
```

```
        map.put(2,"456");
        map.put(3,"789");
        Set<Map.Entry<Integer,String>> ss = map.entrySet();//获取到键值对set

        for (Iterator< Map.Entry<Integer,String> > Item =
ss.iterator();Item.hasNext();){
            Map.Entry<Integer,String > temp = Item.next();//迭代器遍历set，获取到
键值对Entry对象

            System.out.println(temp.getKey()+":"+temp.getValue());
/*结果
1:123
2:456
3:789
*/
```

```
/*法2*/
        Map<Integer,String> map = new HashMap<>();
        map.put(1,"123");
        map.put(2,"456");
        map.put(3,"789");
        Set<Integer> ss = map.keySet();//获取到key的set

        for (Iterator<Integer> Item = ss.iterator();Item.hasNext();){//遍历key
的set，再通过key获取map的value值
            Integer temp = Item.next();

            System.out.println(map.get(temp));

        }
/*结果
123
456
789
*/
```

13. 遍历方法总结

    https://www.sxt.cn/Java_jQuery_in_action/nine-ergodicset.html

14. Collections工具类的方法介绍

```
        List<String> list = new ArrayList<>();
        for (int i =0;i<10;i++){
            list.add("这是: "+i);
        }
        System.out.println(list);

        Collections.shuffle(list);//随机排列list中的元素
        System.out.println(list);

        Collections.reverse(list);//逆序排列
        System.out.println(list);
```

```
12
13          Collections.sort(list);//按照递增方式排序，自定义类需实现comparable接口
14          System.out.println(list);
15
16          System.out.println(Collections.binarySearch(list,"这是: 5"));//二分查找值的具体位置
17
18  /*结果
19  [这是: 0, 这是: 1, 这是: 2, 这是: 3, 这是: 4, 这是: 5, 这是: 6, 这是: 7, 这是: 8, 这是: 9]
20  [这是: 4, 这是: 6, 这是: 5, 这是: 9, 这是: 1, 这是: 8, 这是: 3, 这是: 2, 这是: 0, 这是: 7]
21  [这是: 7, 这是: 0, 这是: 2, 这是: 3, 这是: 8, 这是: 1, 这是: 9, 这是: 5, 这是: 6, 这是: 4]
22  [这是: 0, 这是: 1, 这是: 2, 这是: 3, 这是: 4, 这是: 5, 这是: 6, 这是: 7, 这是: 8, 这是: 9]
23  5
24
25  */
```

15. 通过Collection来存放表格数据(ORM思想，即对象关系映射)

ID	姓名	薪水
1001	张三	5000
1002	李四	6000
1003	王五	7000

方法一：行数据用map存放，整张表用list存放（也可以反过来存放）

```
1  Map<String,Object> row1 = new HashMap<>();
2  row1.put("id",1001);
3  row1.put("姓名","张三");
4  row1.put("薪水",5000);
5  //row2,row3同理，多行数据多个map
6  List<Map<String,Object>> table = new ArrayList<>();
7  table.add(row1);
8  table.add(row2);
9  table.add(row3);
```

方法二：通过构造javabean，再将对象存储起来

```
1  public class employee{
2      private int ID;
3      private String Name;
4      private double salary;
5      //构造函数等省略
6  }
7
8  Map<Integer,employee> map = new HashMap<>();
```

```
employee e1 = new employee(1001,"张三",3000);
//e2,e3同理
map.put(e1.getId,e1);
map.put(e2.getId,e2);
map.put(e3.getId,e3);
//这样，一张表就被存储起来了，如果不需要key标记，直接使用list存储也可以
list.add(e1);

```