接口

接口技术用于描述类有什么功能，但并不给出具体实现，类要遵从接口描述的统一规则进行定义，所以，接口是对外实现的一组规则、标准

比如：USB接口遵守一定规则，无论是电脑、充电宝、数据线，这些USB接口都是通用的，因为USB遵守了统一的标准和规则

接口的定义

使用关键字 ==interface== interface接口名{}

类和接口的关系

实现关系，用==implements== class 类名 implements 接口名{}

接口创建对象特点

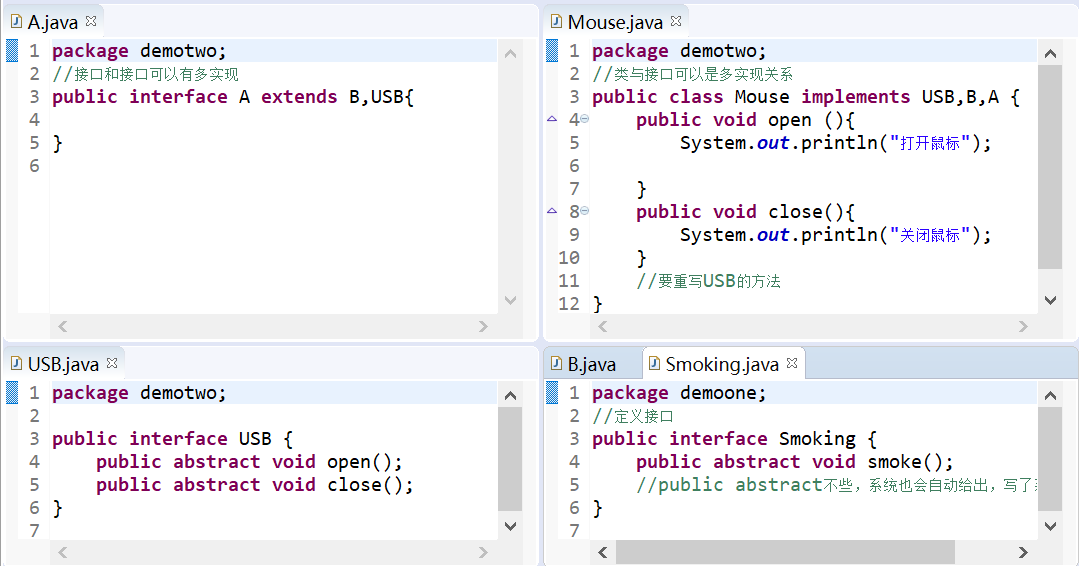
1.==接口不能实例化，通过多态的方式实例化子类对象==



2.接口的子类(实现类) ==可以是抽象类（不用重写接口方法），也可以是普通类（必须重写接口方法）

3.接口继承关系的特点：继承关系，也可以是多继承：

格式：接口 extends 接口1，接口2，接口3........



继承和实现的区别

==继承体现的是 “is a”的关系，父类中定义共性内容==

==实现体现的是“like a”的关系，接口中定义拓展功能==

比如：狗和猫都有吃饭睡觉，就把这些共性定义在父类。猫会独特的跳高，把这个属性定义在接口，再让猫调用

接口成员特点

成员变量特点：接口没有成员变量，只有共有的、静态的常量

public static final 常量 = 常量（系统会默认写）

成员方法：==JDK7及以前，共有的、抽象的方法

(JDK7以前只能有抽象方法)

==JDK8以后，有默认方法和静态方法==

public default 返回值类型 方法名（）{}

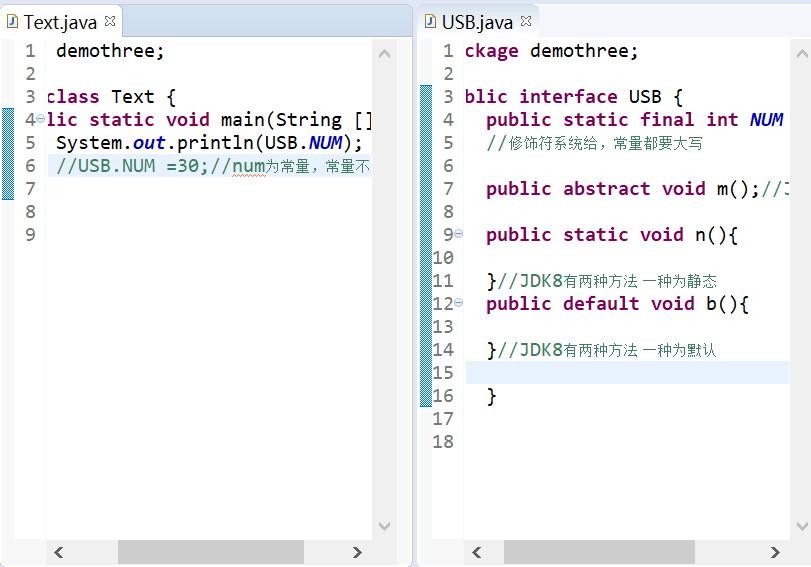
static 返回值类型 方法名（）{}

​ ==JDK9以后,可以有私有方法；==

private 返回值类型 方法名（）

接口构造方法的特点

接口不能实例化，也没有需要初始化的成员，所以==没有构造方法==



内部类

指一个类的内部还包含了另外一个操作类的定义，被包含的类称为内部类，后者的类称为外部类

内部类大脑

String name；

Int age；

有个变量比较复杂，则可以用内部类表示

Person类

分类 : 成员内部类（static和非static）、局部内部类（方法块内、代码块内、构造器内）、静态嵌套类、匿名内部类

成员内部类，就是作为外部类的成员，可以直接使用外部类的所有成员和方法，虽然成员内部类可以无条件地访问外部类的成员，而外部类中如果要访问成员内部类的成员，必须先创建一个成员内部类的对象，再通过指向这个对象的引用来访问

成员内部类：一方面，作为外部类的成员（调用外部类的结构，用stati修饰，也可以用四种权限符修饰, 外部类只能被public和包访问两种权限修饰 。）

另一方面，作为一个类属性（类内可以定义属性、方

法、构造器，可以继承，若被final修饰，不能被继

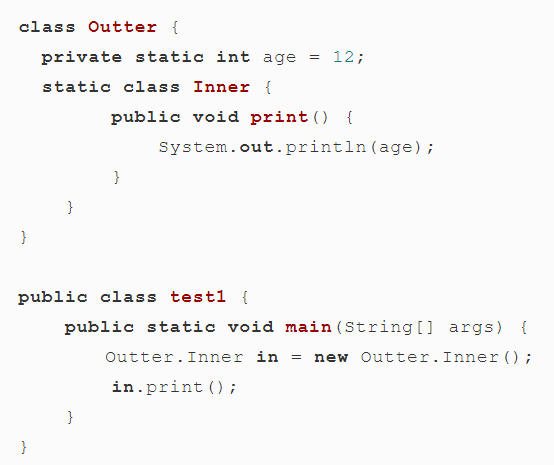
承，可以被abstract修饰）



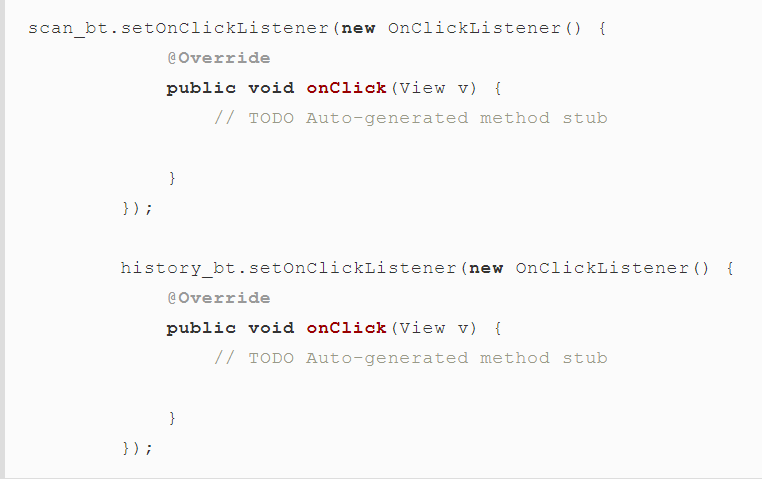
局部内部类是定义在一个方法或者一个作用域里面的类，它和成员内部类的区别在于局部内部类的访问仅限于方法内或者该作用域内。在方法中定义的内部类只能访问方法中final类型的局部变量



静态嵌套类又叫静态局部类、嵌套内部类，就是修饰为static的内部类。声明为static的内部类，不需要内部类对象和外部类对象之间的联系，就是说我们可以 直接引用outer.inner ，即不需要创建外部类，也不需要创建内部类。如果用static 将内部内静态化，那么内部类就只能访问外部类的静态成员变量，具有局限性。



匿名内部类,使用匿名内部类能够在实现父类或者接口中的方法情况下同时产生一个相应的对象，但是前提是这个父类或者接口必须先存在才能这样使用



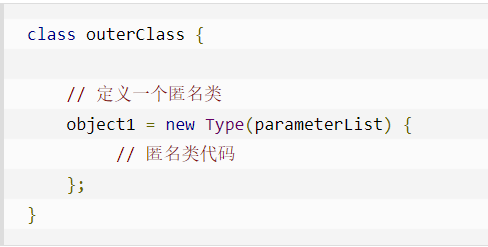
匿名类

Java 中可以实现一个类中包含另外一个类，且不需要提供任何的类名直接实例化。

主要是用于在我们需要的时候创建一个对象来执行特定的任务，可以使代码更加简洁。

匿名类是不能有名字的类，它们不能被引用，只能在创建时用 new 语句来声明它们。

匿名类语法格式：



创建了一个匿名类对象 object1，匿名类是表达式形式定义的，所以末尾以分号 ; 来结束。匿名类通常继承一个父类或实现一个接口。

异常类

一、什么是异常：

异常就是在运行时产生的问题。通常用Exception描述。

在java中，把异常封装成了一个类，当出现问题时，就会创建异常类对象并抛出异常相关的信息（如详细信息，名称以及异常所处的位置）。

二、异常的继承关系：

Throwable类是所有错误跟异常类的超类（祖宗类）。

Exception异常类及其子类都是继承自Throwable类，用来表示java中可能出现的异常，并且合理的处理这些异常。

RuntimeException类是运行异常类，继承自Exception类，它以及它的子类只能在运行过程中存在，当出现时，只能修改源代码，此异常无法处理。

Error类是与Exception的平级的类，用来表示Java中存在的严重错误，只能通过修改代码来解决问题。

继承体系的总结：

Throwable: 它是所有错误与异常的超类（祖宗类）

|- Error 错误

|- Exception 编译期异常,进行编译JAVA程序时出现的问题

|- RuntimeException 运行期异常, JAVA程序运行过程中出现的问题

三、异常与错误的区别：

异常是指程序在编译或者运行时出现的某种异常问题，我们可以对异常进行某种处理，如果不处理异常的话，程序将会停止运行。

错误是指程序在运行时出现的严重问题，无法处理，程序将会停止运行，Error通常都是系统级别的问题，都是虚拟机jvm所在系统发生的，只能通过修改源代码解决问题。

四、异常产生的过程：

1.运行或编译时产生异常

2.创建异常类的对象

3.声明异常类

4.将异常类对象传给调用者（main()方法）处理

5.调用者无法处理，再将异常类对象传给jvm虚拟机

6.jvm虚拟机将异常类的信息（名称、详细信息、异常所处的位置）打印在屏幕上，并且停止程序的运行



五、抛出异常throw

在java中，提供了一个throw关键字，用来抛出一个指定的异常。

使用方法：

1.创建一个异常对象。封装一些提示信息

2.将这个异常告知调用者

使用格式：throw new 异常类名（参数）；

六、声明异常throws

声明异常格式：

修饰符 返回值类型 方法名称 （参数）throws 异常1名称，异常2名称{

}

七、捕获异常try...catch...finally

捕获异常：Java中对异常有针对性的语句进行捕获，可以对出现的异常进行指定方式的处理

语句格式：复制代码

try {

//需要被检测的语句。

}

catch(异常类 变量) { //参数。

//异常的处理语句

}

finally {

//一定会被执行的语句。

}

try：该代码块中编写可能产生异常的代码。

catch：用来进行某种异常的捕获，实现对捕获到的异常进行处理。

finally：有一些特定的代码无论异常是否发生，都需要执行。另外，因为异常会引发程序跳转，导致有些语句执行不到。而finally就是解决这个问题的，在finally代码块中存放的代码都是一定会被执行的。

捕获异常的组合方式：

1. try catch finally组合：检测异常，并传递给catch处理，并在finally中进行资源释放。

2. try catch组合 : 对代码进行异常检测，并对检测的异常传递给catch处理。对异常进行捕获处理。

3. 一个try 多个catch组合 : 对代码进行异常检测，并对检测的异常传递给catch处理。对每种异常信息进行不同的捕获处理。

注意:这种异常处理方式，要求多个catch中的异常不能相同，并且若catch中的多个异常之间有子父类异常的关系，那么子类异常要求在上面的catch处理，父类异常在下面的catch处理。

4. try finally 组合: 对代码进行异常检测，检测到异常后因为没有catch，所以一样会被默认jvm抛出。异常是没有捕获处理的。但是功能所开启资源需要进行关闭，所有finally。只为关闭资源。

八、运行时期异常 RuntimeExecption

特点：方法中抛出运行时期异常,方法定义中无需throws声明,调用者也无需处理此异常

运行时期异常一旦发生,需要程序人员修改源代码

九、异常在方法重写中细节

1. 子类覆盖父类方法时，如果父类的方法声明异常，子类只能声明父类异常或者该异常的子类，或者不声明。

class Fu {

public void method () throws RuntimeException {

}

}

class Zi extends Fu {

public void method() throws RuntimeException { } //抛出父类一样的异常

//public void method() throws NullPointerException{ } //抛出父类子异常

}

2.当父类方法声明多个异常时，子类覆盖时只能声明多个异常的子集

class Fu {

public void method () throws NullPointerException, ClassCastException{

}

}

class Zi extends Fu {

public void method()throws NullPointerException, ClassCastException { }

public void method() throws NullPointerException{ } //抛出父类异常中的一部分

public void method() throws ClassCastException { } //抛出父类异常中的一部分

}

3.当被覆盖的方法没有异常声明时，子类覆盖时无法声明异常的。

public void method (){

}

}

class Zi extends Fu {

public void method() throws Exception { }//错误的方式

}

十、异常中常用方法

在Throwable类中为我们提供了很多操作异常对象的方法，常用的如下：

1.getMessage方法：返回该异常的详细信息字符串，即异常提示信息

2. toString方法：返回该异常的名称与详细信息字符串

3. printStackTrace：在控制台输出该异常的名称与详细信息字符串、异常出现的代码位置

十一、自定义类异常

格式：

Class 异常名 extends Exception{ //或继承RuntimeException

public 异常名(){

}

public 异常名(String s){

super(s);

}

}

自定义异常继承Exception演示：

class MyException extends Exception{

/\*为什么要定义构造函数，因为看到Java中的异常描述类中有提 供对异常对象的初始化方法。\*/

public MyException(){

super();

}

public MyException(String message) {

super(message);// 如果自定义异常需要异常信息，可以通过 调用父类的带有字符串参数的构造函数 即可。

}

}

自定义异常继承RuntimeException演示：

class MyException extends RuntimeException{

/\*为什么要定义构造函数，因为看到Java中的异常描述类中有提 供对异常对象的初始化方法。 \*/

MyException(){

super();

}

MyException(String message) {

super(message);// 如果自定义异常需要异常信息，可以通过 调用父类的带有字符串参数的构造函数 即可。

}

}

实例代码：throw,throws,try/catch的使用





String类 （不导包）

简介：字符里每一个字符对象都是常量

常量：自定义常量，由final值修饰；字面值常量，包含字符串常量Java.base（模块）------java.lang（包）不需要------String（不需要打包）

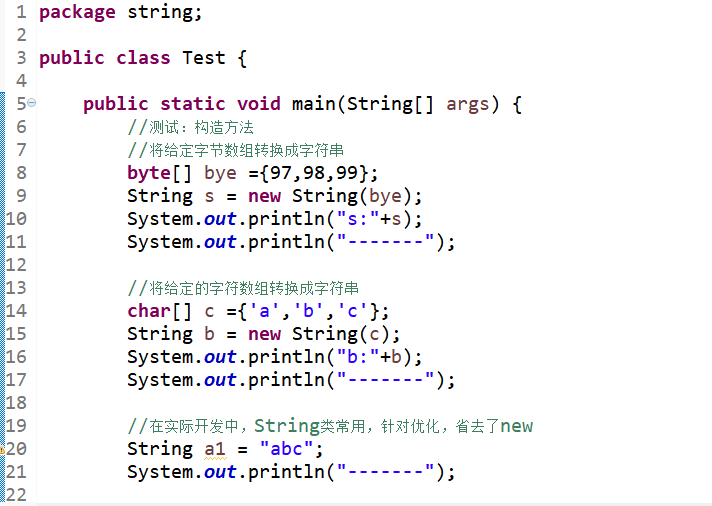
构造方法：String（byte[]）/String（char[]）：构造一个String对象，将指定的字节数组中的数据转化成字符串

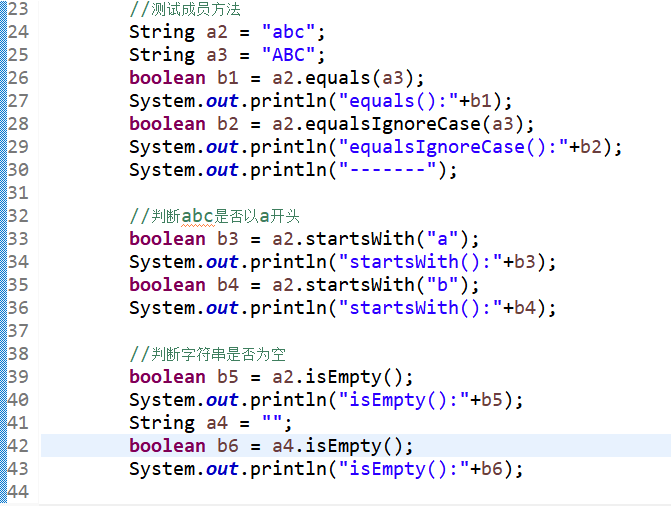
成员方法：boolean equals（String）：判断当前字符串与所给定字符串是否相同，区分大小写

boolean equalsIgnoreCase（String）：判断当前字符串与所给定字符串是否相同

boolean startsWith（String）：判断是否以给定字母开头

boolean isEmpty()（String）：判断字符串是否为空





成员方法：int length( ):获取当前字符串的长度

char charAt(int index):获取指定索引位置的字母

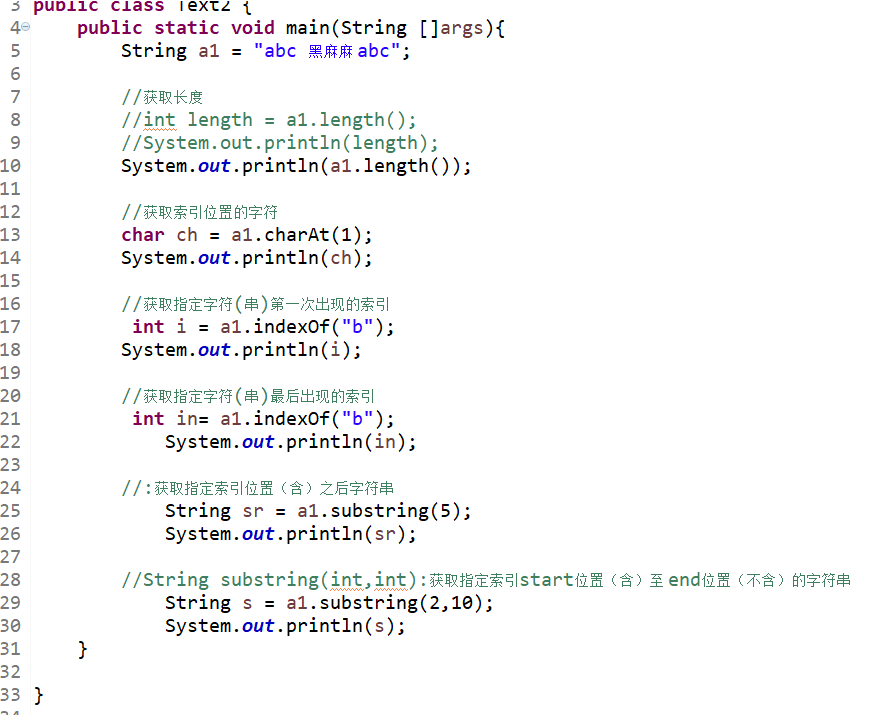
int indexOf(String):获取指定字符(串)第一次出现的索引

int lastIndexOf(String):获取指定字符(串)最后出现的索引

String substring(int):获取指定索引位置（含）之后字符串

String substring(int,int):获取指定索引start位置（含）至 end位置（不含）的字符串

String.valueOf()对象转为String

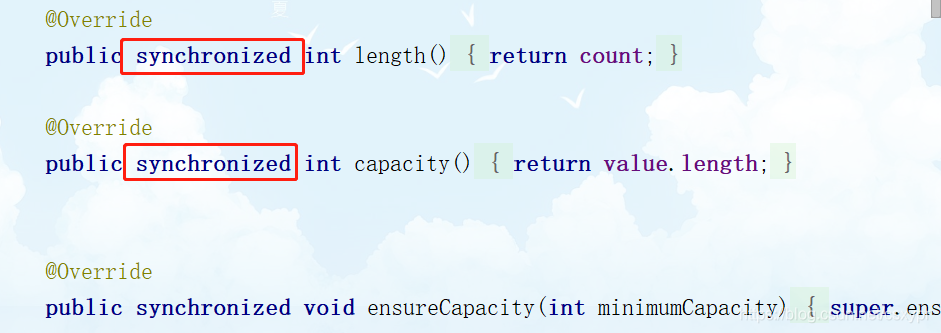


StringBuilder类和StringBuffer类

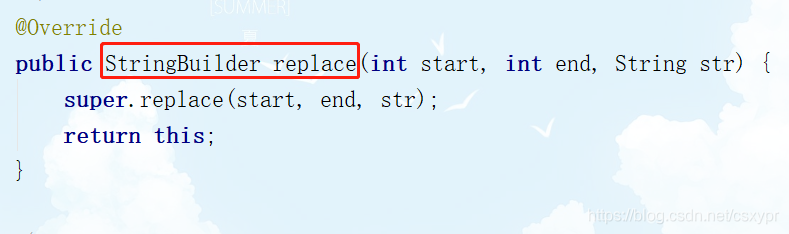
与String区别：String类是不可变类，且修改后的值仍存在，直到垃 圾回收。而上述两个类则是合并在一起

两者区别：StringBuffer是线程安全的，而StringBuilder则没有实现线程安全功能，所以性能略高。

StringBuffer类中实现的方法：



StringBuilder类中实现的方法：



  由此可见，StringBuffer类中的方法都添加了synchronized关键字，也就是给这个方法添加了一个锁，用来保证线程安全。

简洁：可变字符序列，用于构造字符对象。内部使用自动扩容的数组操作字符串数据，StringBuilder和StringBuffer类使用相同的API

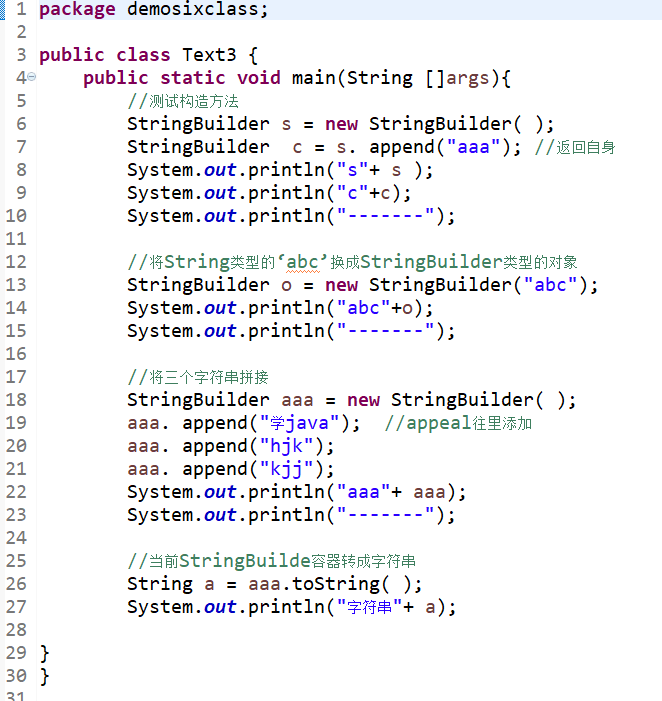
构造方法：StringBuilder( )：构造一个空的StringBuilder容器

StringBuilder(String )：构造一个空的StringBuilder容器，

并添加指定字符串

成员方法：StringBuilde append(...)：将任意数据添加到StringBuilde容器中，返回自身

String toString(...)：将当前StringBuilde容器转成字符串，返回本身



Scanner类 （导包）

简介：扫描器，能够解析字符串（String）和基本数据类型的转换

构造方法：Scanner（InputStream）：构造一个扫描器对象，从指定输入流中获取数据参数System.in，对应键盘录入

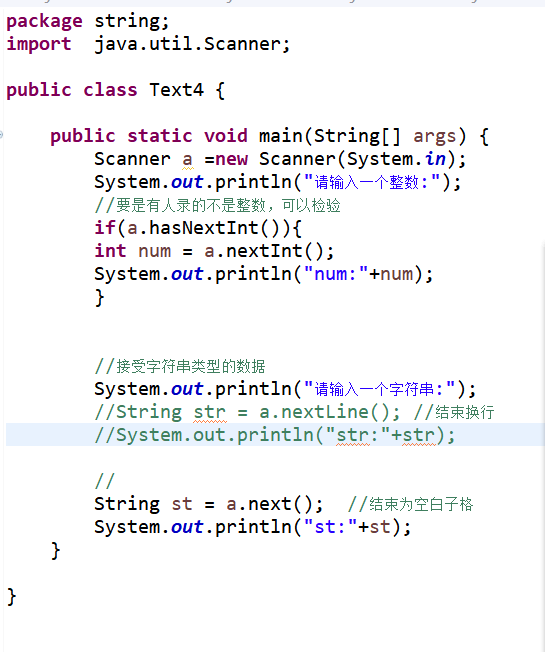
成员方法：hasNextXxx（）：判断是否还有下一个输入项，其中Xxx是任意基本数据类型，返回值类型为布尔类型

NextXxx（）：获取下一个输入项，其中Xxx可能是任意基本数据类型，返回对应类型的数据

两者合起来可以操作类型数据

String nextLine（）：获取下一行数据，以换行符作为分隔符

String next（）；获取下一行数据，以空白符作为分隔符



Object类（不导包）

类层次结构最顶层的基类，所有类都直接或间接的继承object类，所以所有的类都为object的子类

构造方法：Object（）：创造对象

成员方法：int hashCode()：返回对象的哈希码值（根据地址值计算

来的结果），该方法通过对对象地址值进

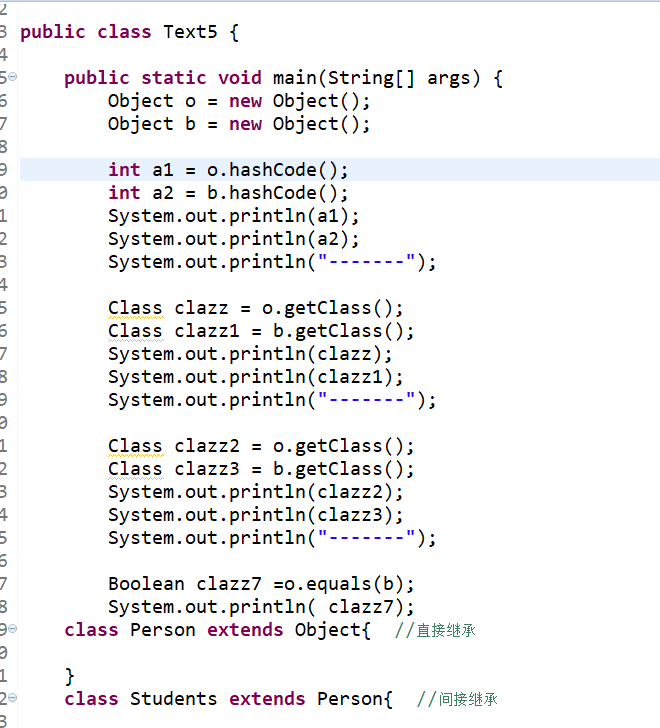
行计算，不同对象的返回值一般不同

Class<?>getClass():返回调用此方法对象的运行时类对（调用者的字节码对象)，且一个类只有一个字节码文件，所以值都一样

String toString():返回该对象的返回值类型，默认打印的是地址值，每个对象地址值不同。

Boolean equals()：返回其他某个对象是否与此对象的地址值“相等”，默认情况下两个对象的地址值相同，建议重写

地址值：全类名+该对象的哈希码的无符号十六进制形式



Date类和Calendar（抽象）类（导包）

简介：日期和日历类，用于操作日期相关信息

构造方法：Date（）：构造一个日期对象，当前系统时间，精确到毫秒

1秒 =1000毫秒

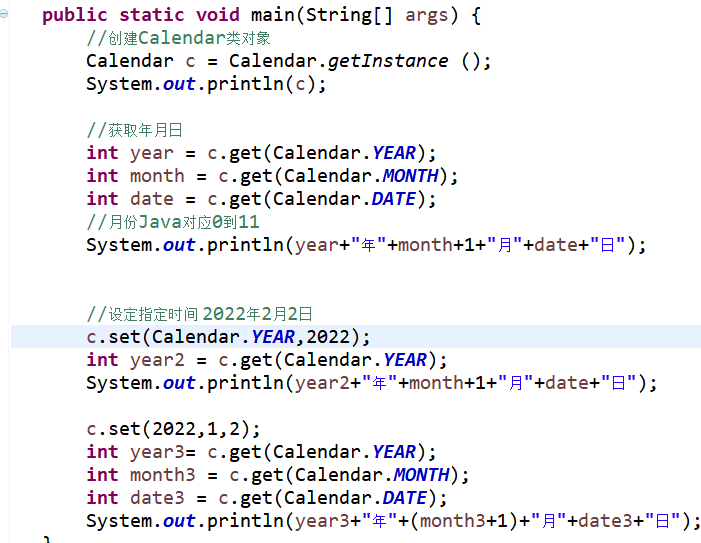
Date（long）：构造一个日期对象，时间自“1970年1月1日00：00：00GMI”算起，至指定参数的毫秒值

成员方法：long getTime（）：将日期对象转换成相应的毫秒值

Static getInstance():根据当前时区和语言环境获取日历对象

int get(int field):返回给定日历字段的值

void set（int field，int value）：将给定字段设置为时间



基本类型的包装类

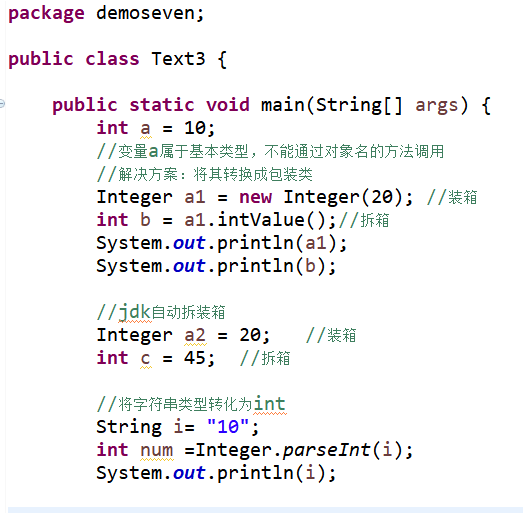
Byte Short Integer Long Character Float Double Boolean

简介：基本数据类型不是对象，所以Java针对基本类型提供了对应的包装类，以对象的形式使用

装箱：基本类型转包装类型（对象类型） Integer i 自动装箱

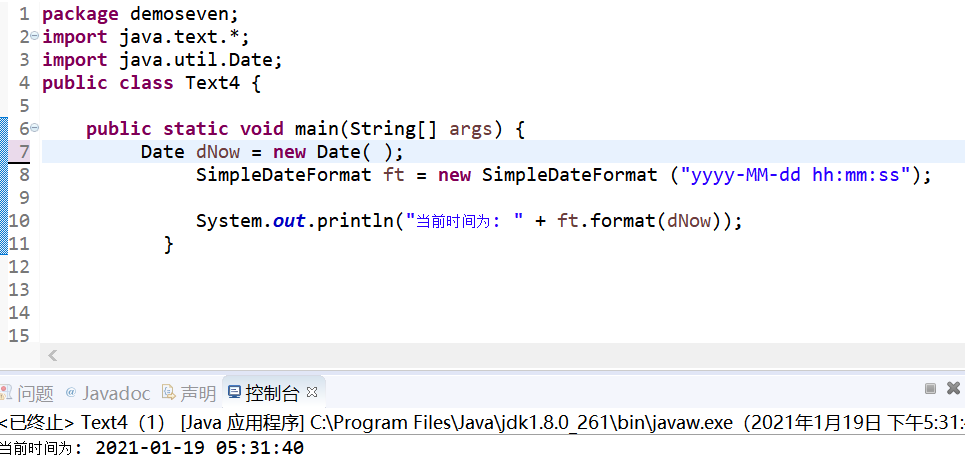
拆箱：包装类型（对象类型）转换为基本类型 int 自动拆箱

成员方法：static包装类型parseXxx（String）：将字符串类型转换为对应的基本类型，除了character类，其他包装类都有parseXxx方法，像转换可通过charAt转换



格式化日期

使用SimpleDateFormat 是一个以语言环境敏感的方式来格式化和分析日期的类。SimpleDateFormat 允许你选择任何用户自定义日期时间格式来运行。例如：



这一行代码确立了转换的格式，其中 yyyy 是完整的公元年，MM 是月份，dd 是日期，HH:mm:ss 是时、分、秒。

注意:有的格式大写，有的格式小写，例如 MM 是月份，mm 是分；HH 是 24 小时制，而 hh 是 12 小时制。

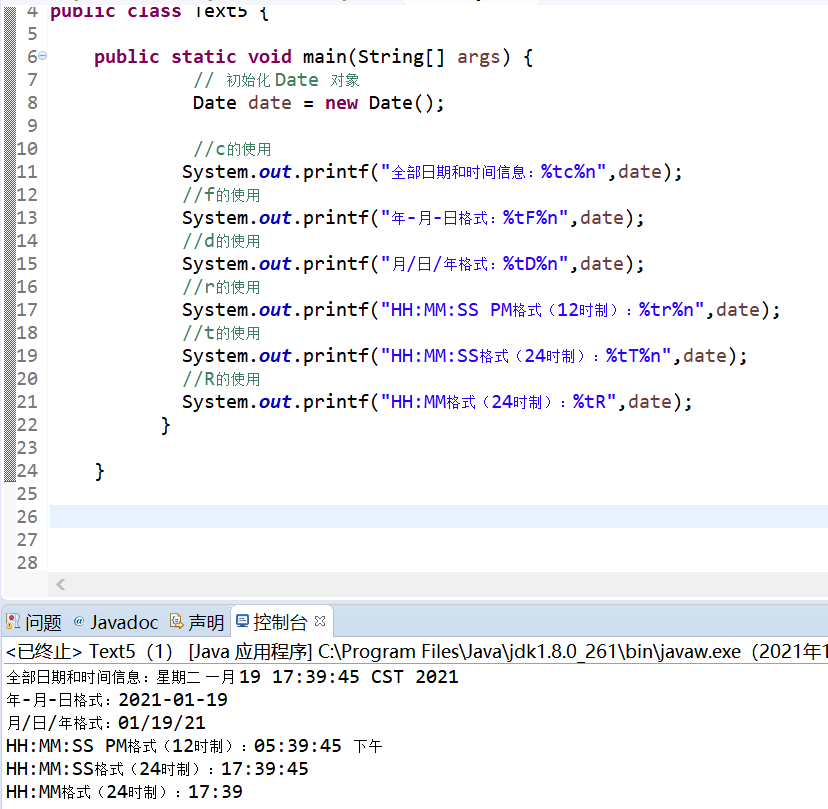
## 日期和时间的格式化编码

时间模式字符串用来指定时间格式。在此模式中，所有的 ASCII 字母被保留为模式字母，定义如下：

|  |  |  |
| --- | --- | --- |
| **字母** | **描述** | **示例** |
| G | 纪元标记 | AD |
| y | 四位年份 | 2001 |
| M | 月份 | July or 07 |
| d | 一个月的日期 | 10 |
| h | A.M./P.M. (1~12)格式小时 | 12 |
| H | 一天中的小时 (0~23) | 22 |
| m | 分钟数 | 30 |
| s | 秒数 | 55 |
| S | 毫秒数 | 234 |
| E | 星期几 | Tuesday |
| D | 一年中的日子 | 360 |
| F | 一个月中第几周的周几 | 2 (second Wed. in July) |
| w | 一年中第几周 | 40 |
| W | 一个月中第几周 | 1 |
| a | A.M./P.M. 标记 | PM |
| k | 一天中的小时(1~24) | 24 |
| K | A.M./P.M. (0~11)格式小时 | 10 |
| z | 时区 | Eastern Standard Time |
| ' | 文字定界符 | Delimiter |
| " | 单引号 |  |

使用printf格式化日期

printf 方法可以很轻松地格式化时间和日期。使用两个字母格式，它以 %t 开头并且以%n结尾。



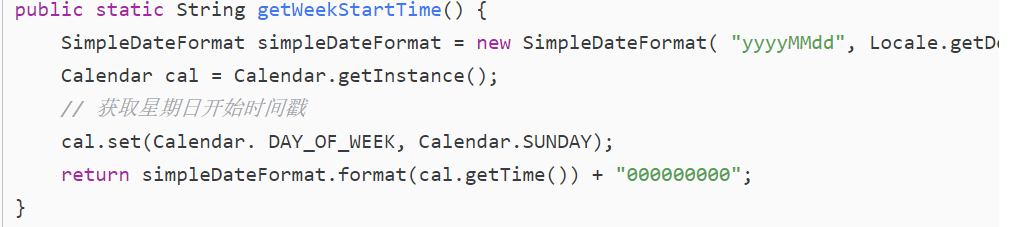
|  |  |  |
| --- | --- | --- |
| **转  换  符** | **说    明** | **示    例** |
| c | 包括全部日期和时间信息 | 星期六 十月 27 14:21:20 CST 2007 |
| F | "年-月-日"格式 | 2007-10-27 |
| D | "月/日/年"格式 | 10/27/07 |
| r | "HH:MM:SS PM"格式（12时制） | 02:25:51 下午 |
| T | "HH:MM:SS"格式（24时制） | 14:28:16 |
| R | "HH:MM"格式（24时制） | 14:28 |

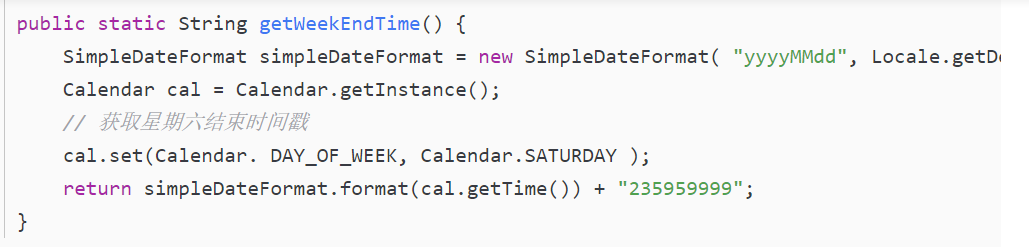
DAY\_OF\_WEEK

使用Calendar获取DAY\_OF\_WEEK本周开始和结束的时间戳；

和获取今日开始和结束的时间戳一样，本周开始和结束的时间戳，就是以本周第一天开始的00:00到本周最后一天的23:59分结束。

需要注意的是，国际上是以星期日为一周第一天的开始，Calendar中提供的DAY\_OF\_WEEK获取的一周也是以星期日作为一周的开始。





而中国的习惯是以星期一作为一周的开始，所以需要根据项目的实际需求来设置。

