Java笔记

流程控制

循环语句

foreach

```
for(元素类型x:遍历对象obj){
    // 引用了x的java语句;
}
```

```
public class Repetition{
    public static void main(String args[]){
        int arr[]={5,13,96};
        System.out.println("一维数组中的元素分别为：");
        //x的类型与arr元素的类型相同 。for循环依次取出arr中的值并赋给x
        for(int x:arr){
            System.out.println(x);
        }
    }
}

/*输出结果：
一维数组中的元素分别为：
5
13
96
*/
```

数组

基本操作

数组排序

- `Arrays.sort(arr);`
- `Arrays.parallelSort(arr);`多线程排序，数据量大于一百万

数组复制

- `int a[]=Arrays.copyOf(arr,int newlength);`

> newlength：复制后新数组的长度

> 注意不能直接a = b， 数组名是指针常量

类与对象

封装

- protected**同包其他类或子类(继承)**可见，其他包的类不可见；
- private都不可见，只有本类可见。

包访问权限

当声明类时不使用public等关键字修饰类的权限的时候，则这个类预设为包存取范围，即只有一个包中的类可以访问这个类的成员变量或成员方法。

- 包访问权限介于private与protected之间
- 包外派生类可以调用基类protected成员，包访问权限不行

类的构造方法

> Java类内的属性值不支持默认值，不能直接定义 `int count=0`
>
> 应使用默认构造函数初始化

```
public class eggCake{
    int eggCount;
    public EggCake(int eggCount){//有参构造
        this.eggCount=eggCount;
    }
    public EggCake(){
        //设置鸡蛋灌饼里蛋的个数为1
        this(1);
    }
}
```

this关键字

this关键字用于表示本类当前的对象，只能在本类中使用。

```
public void setName(String name){//定义一个setName()的方法
    this.name=name;//将参数值赋予类中的成员变量
}
```

继承

类的继承

extends 关键字

语法：`CHild extends Parents`

super 关键字

super可以看作是**父类的**，存在类与类的继承关系会存在。主要功能有如下：

- 用**super.属性、super.方法**的方式，来表明调用的是父类的属性、方法
- 在重写了父类的方法时，如果**子类需要调用父类被重写的方法**，需要加super关键字，显示了调用了父类的方法

- 可以在子类的构造器中使用super(形参列表)，来调用父类的构造器但是必须要在代码的首行。
- this（形参列表）和super（形参列表）只能用一个，因为这两个都要求放在首行。

instanceof关键字

`instanceof` 是 Java 的保留关键字，它的作用是测试它**左边的对象是否是它右边的类的实例**，返回 boolean 的数据类型。

final关键字

final关键字代表最终、不可改变的。

- 修饰类：fianl修饰符如果修饰类的话就代表该类不可以被继承,即**不允许被继承**
- 修饰变量：final修饰成员变量之后就意味着该成员**变量不能够被修改**，由于外界无法修改该成员变量的值，故而其创建时就需要被赋值或者在其构造方法处赋值
- 修饰方法：修饰方法之后该方法**无法被子类覆盖**
- 修饰引用：这是个重头戏！引用被final修饰之后，虽然不能再指向其他对象，**但是它指向的对象的内容是可变的**
- 修饰数组：final对于数组的修饰跟final对于引用的修饰是有点儿类似的，对于被final修饰的数组而言，不能让这个数组指向别的数组，**但是是可以修改这个数组中的元素的值的**
- final不能用来修饰构造方法

Object 类

Object类是一切类的基类，隐含的继承

- 如果没声明，toString()一定是调用自Object类，输出是字符编号。
- 输出字符串自动会调用toString()函数，应当重写以达到需要的输出目的。
- 重写只能**保持或扩大**访问权限，如原本是Public不能改成Private。

对象的类型转换

- 需要基类对象的任何地方，都可以用**派生类对象替代**
- 定义为基类对象，**只能使用基类的方法**，对于派生类新增的方法是无法使用的

抽象类与接口

抽象类

- 只要类中有一个抽象方法，此类就是抽象类
- 抽象类不能实例化，抽象类存在的目的就是**为了被继承**
- C++中全是抽象方法就叫纯虚类

接口

接口是抽象类的延伸，可以将它看作纯粹的抽象类，接口中的方法都没有方法体。

- 接口使用`interface`关键字定义
- 使用接口使用`implements`关键字

```
public interface Paintable{
    ...
}
```

```
public class className implements Paintable{
    ...
}
```

字符串

注意：字符串不能使用 == 进行对比，需要使用**equals**关键字。

String创建的字符串是不可变的，改变字符串对象会在内存中创建新的字符串对象。

常用库类

Random类

```
Random r=new Random();
```

BigInteger类

```
BigInteger twoLnstance=new BigInteger("2");
```

Runtime类

```
public class RuntimeExecDemo{
    public
}
```

集合类

概述：Java语言的java.util包中提供了一些集合类，这些集合类又被称为容器。提到容器不难会想到数组，集合类与数组不同之处是，数组的长度是固定的，集合的长度是可变的；Lab7

常用的集合：

- List集合
- Set集合
- Map集合

- 其中List与Set实现了Collection接口。各接口还提供了不同的实现类。

方·····法	功·能·描·述
add(E e)	将指定的对象添加到该集合中
remove(Object o)	将指定的对象从该集合中移除
isEmpty()	返回 boolean 值，用于判断当前集合是否为空
iterator()	返回在此 Collection 的元素上进行迭代的迭代器。用于遍历集合中的对象
size()	返回 int 型值，获取该集合中元素的个数

在使用集合类的时候，遍历集合一般使用**iterator迭代器**进行遍历。

```
Iterator<String> iterator = arrayList.iterator();
while(iterator.hasNext()) { // 使用hasNext判断
    String tempString = iterator.next();
    if(word.equals(tempString)) {
        System.out.println(word+iterator.next());
        arrayList.remove(iterator.next());
    }
}
```

List集合

List接口继承了Collection接口，并且定义了两个非常重要的方法：

- get(int index)：获取指定索引位置的元素
- set(int index,Object obj)：将集合指定索引位置的对象修改为指定的对象

ArrayList

ArrayList类实现了**可变的数组**，允许所有元素，包括null，并可以根据索引位置对集合进行快速的随机访问。缺点是向指定的索引位置插入对象或删除对象的速度较慢。

```
List<E> list = new ArrayList<E>()
```

使用父类对象来承接子类对象的主要原因是多态性。多态性允许我们在不知道具体的子类型的情况下，使用父类的引用来操作子类的对象。这样，我们就可以使用相同的代码来处理多种不同的子类型，并且可以在运行时再确定具体的子类型，这样可以提高代码的灵活性和可扩展性。

LinkedList

LinkedList类采用**链表结构**保存对象。这种结构的优点是便于向集合中插入和删除对象，需要向集合中插入、删除对象时，使用LinkedList类实现的List集合的效率较好；但对于随机访问集合中的对象，使用LinkedList类实现List集合的效率较慢。

```
List<E> list = new LnkedList<E>() // 后者E可以省略
```

Set集合

Set集合中的对象不按特定的方式排序，只是简单地把对象加入集合中，但Set集合中不能包含重复对象。重复对象add进入后不会报错，但在集合内只有一个重复对象。

HashSet

HashSet实现Set接口，由哈希表（HashMap）支持。它不保证Set集合的迭代顺序，特别是它不保证该序列恒久不变。此类允许使用null元素。

TreeSet

TreeSet还实现了SortedSet接口，因此TreeSet类实现的Set集合在遍历集合时按照自然顺序递增排序。TreeSet新增加的方法如下。

方法	功能描述
first()	返回此 Set 中当前第一个（最低）元素
last()	返回此 Set 中当前最后一个（最高）元素
comparator()	返回对此 Set 中的元素进行排序的比较器。如果此 Set 使用自然顺序，则返回 null
headSet(E toElement)	返回一个新的 Set 集合，新集合包含 toElement（不包含）之前的所有对象
subSet(E fromElement, E fromElement)	返回一个新的 Set 集合，包含 fromElement（包含）对象与 fromElement（不包含）对象之间的所有对象
tailSet(E fromElement)	返回一个新的 Set 集合，新集合包含对象 fromElement（包含）之后的所有对象

如果要指定对比的属性，需要重写compareTo方法

```
@Override
public int compareTo(UpdateStu o) {
    if (this.id<o.id)
        return -1; // 本对象小于下一个对象 返回负数
    else if (this.id==o.id)
        return 0;
    else
        return 1;
}
```

Map集合

概述：Map接口提供了将**key映射到value**的对象。一个映射不能包含重复的key，每个key最多只能映射到一个值。Map接口中同样提供了集合的常用方法，除此之外还包括如下表所示的常用方法。

方法	功能描述
put(K key, V value)	向集合中添加指定的 key 与 value 的映射关系
containsKey(Object key)	如果此映射包含指定 key 的映射关系，则返回 true
containsValue(Object value)	如果此映射将一个或多个 key 映射到指定值，则返回 true
get(Object key)	如果存在指定的 key 对象，则返回该对象对应的值，否则返回 null
keySet()	返回该集合中的所有 key 对象形成的 Set 集合
values()	返回该集合中所有值对象形成的 Collection 集合

HashMap

HashMap是基于哈希表的Map接口的实现，HashMap通过哈希码对其内部的映射关系进行快速查找。允许使用null值和null键。

TreeMap

TreeMap中的映射关系存在一定的顺序，如果希望Map集合中的对象也存在一定的顺序，应该使用TreeMap类实现Map集合。不允许对象为null。

```
HashMap<String,String> hashmap= new HashMap<>();
hashmap.put("bob", "Book");
hashmap.put("c", "look");
hashmap.put("a", "well");
System.out.println("HashMap: ");
Iterator<String> iterator1 = hashmap.iterator();
while(iterator1.hasNext()) { // iterator获取hashmap为键值
    System.out.println(iterator1.next()+" ");
}
// 获取value值需要使用values方法获取集合
Collection<String> collection = hashmap.values();
Iterator<String> iterator2 = collection.iterator();
while(iterator2.hasNext()) {
    System.out.println(iterator2.next()+" ");
}
```

枚举类型与泛型

枚举

在枚举出现之前，人们一般使用接口的方式放置常量。使用接口作为常量参数，可能会出现接口常量值以外的值“冒充”常量。Lab8

```
interface WeekInterface{
    int MON=1, TUE=2, WED=3, THU=4, FRI=5, SAT=6, SUN=7;
}
printDay(1) // 可填入-1
```

枚举类型出现后，其逐渐取代了上述的常量定义方式。使用枚举做参数，只能用枚举中有的值，无法“冒充”。

```
enum WeekEnum {
    MON, TUE, WED, THU, FRI, SAT, SUN;
}
printDay(WeekEnum.MON)
```

其中 **enum** 是定义枚举类型的关键字。

方法	具体含义	使用方法
values()	可以将枚举类型成员以数组的形式返回	WeekEnum.values()
valueOf()	可以实现将普通字符串转化为枚举实例	WeekEnum.valueOf("abc")
compareTo()	比较两个枚举对象在定义时的顺序	MON.compareTo(TUE)
ordinal	得到枚举成员的位置索引（0开始）	MON.ordinal

枚举类中还可以定义构造函数，但这个函数只能被**private**修饰。可提供注释的操作。

枚举特点：

- 类型安全
- 紧凑有效的数据定义
- 可以和程序其他部分完美交互
- 运行效率高

泛型

和C++模板类似。

```
class Book<T>{ // 使用类名<T>的型式
    private T bookInfo;
}
```

泛型的高级用法

- 定义泛型类时声明多个类型
    - ```
      class MyClass<T1, T2>{}
      ```
- 定义泛型类时声明数组类型
    - ```
      public class ArrayClass<T>{
          private T[] array;
          private T[] array = new T[10]; // 错误，无法声明数组的实例
          public static void main(Sting[] args){
              ArrayClass<String> demo = new ArrayClass<String>();
              String value[] = ...;
              demo.setArray(value);
          }
      }
      ```
- 限制泛型可用类型
    - 使用extends super限制

      ```
      class Class<T extends anyClass>
      class Class<T super anyClass>
      ```
- 使用类型通配符？

- 其主要作用是在创建一个泛型类对象时限制这个泛型类的类型实现或继承某个接口或类的子类

```
A<? extends List> a = null;
a = new A<ArrayList>(); // 正常
a = new A<HashMap>(); // 报错，HashMap没有实现List接口
```

- 继承泛型类与实现泛型接口
  - ```
    class ExtendClass<T1>{}
    class SubClass<T1, T2> extends ExtendClass<T1>{}
    ```

注意：泛型的类型只能是类类型，不可以是简单类型。如A<int>不行而A<Integer>可以

lambda表达式与流处理

lambda表达式

lambda表达式用来表示匿名函数，语法格式如下：

```
() -> 结果表达式
参数 -> 结果表达式
(参数1, 参数2) -> 结果表达式
(参数1, 参数2) -> {代码块}
```

实现**函数接口**：

- 实现函数式接口
- 实现无参抽象方法（当接口只有一个方法且无参数）

```
interface Say{
    String say();
}

Say pi = () -> "Hello";
sout(pi.say());
```

- 实现有参抽象方法

```
interface Add{
    int add(int a, int b);
}

Add np = (x, y)->x+y;
int result = np.add(11,23);
```

lambda表达式与外部变量：

- lambda表达式**无法更改局部变量**
  - 局部变量在lambda表达式中被定义成为final，只能调用不能修改。
```

- lambda表达式可以修改类成员变量
  - 类成员变量不被定义成final，可任意修改。

方法的引用

类似C++中的作用域，需要引用**静态方法**

```
类名::静态方法名
```

一般使用接口中的函数接住引用的方法。引用**有参**构造方法时，没有圆括号！

```
MethodInterface sm = MethodDemo::add
ConInterface a = ConDemo::new;
Con b = a.action(123);
```

Function接口

```
Function<T, R> demo = (t) ->{ return r ; };
R r = demo.apply(t);
```

- T：可以理解为方法参数类型
- R：可以理解为方法的返回类型

方法名	功能说明
apply(T t)	抽象方法。按照被子类实现的逻辑，执行函数。参数为被操作泛型对象
compose(Function<? super V, ? extends T> before)	先按照before函数逻辑操作接口的被操作对象t，再讲执行结果作为apply()方法的参数
andThen(Function<? super R, ? extends V> after)	先执行apply(t)方法，将执行结果作为本方法参数，再按照after函数逻辑继续执行
static identity()	此方法是静态方法。返回一个Function对象，此对象的apply()方法只会返回参数值。

流处理

类似数据库的SQL语句，可以执行非常复杂的过滤、映射、查找等功能。

接口与类	作用
Stream接口	流处理的核心接口
Optional类	对象容器，优化了空指针场景
Collectors类	收集器类，可以对Stream对象进行封装、归集、分组等处理

过滤方法

filter() 过滤

该方法可以将lambda表达式作为参数，按照其的逻辑过滤流中元素

```
ArrayList<Employee> employees = Employee.getEmpList();
Stream<Employee> stream = employees.stream();
List<Employee> salerList = stream.filter(e ->
e.isSaler()).collect(Collectors.toList()                    );
```

其他方法

- distinct() 去除流中的重复元素
- limit() 获取流中前N个元素
- skip() 忽略流中前N个元素
- peek() 执行参数中的代码

```
List<Employee> progarmmerList2 =
        employees.stream().filter(e->e.isProgrammer())
        .peek(e->e.setSalary(e.getSalary()+1000.0))
        .collect(Collectors.toList());
for (Employee employee2 : progarmmerList2) {
    System.out.println(employee2);
}
```

数据收集

可以对收集元素进行统计。使用Collectors收集器类实现的。

数据统计

- Collectors.couting() 统计个数
- Collectors.averagingDouble() 平均值
- Collectors.maxBy() 最大值
- Collectors.joining() 拼接
- Collectors.summingDouble() 总和

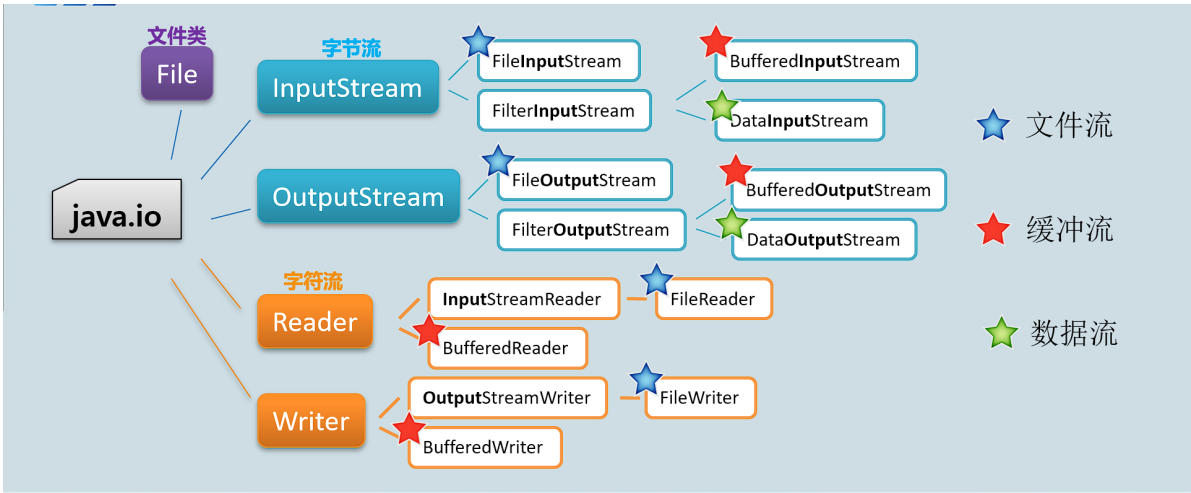
数据分组

Collectors类提供groupingBy()方法进行分组

```
Function<Employee, String> f = Employee::getDept;
Map<String,List<Employee>> map = stream.collect(Collectors.groupingBy(f));
```

I/O流

概述：流是一组有序的数据序列，根据操作的类型，可分为输入流和输出流两种。流提供了一条通道程序，就可以使用这条通道把流中的字节序列送到目的地。Lab10



Java中的所有输入流类都是InputStream（字节输入流）或Reader（字符输入流）的子类；所有输出流类都是OutputStream（字节输出流）或Writer（字符输出流）的子类。

- 输入流指的是将程序外面的数据输入至程序内部
- 输出流指的是将程序内部的数据输出至程序外面

File类

在Java中，File类是代表磁盘的**文件**或者**文件夹（目录）**

```
new File(String pathname)
new File(String parent, String child)
new File(File f, String child) // f父路径对象
```

File是代表文件对象，不是真正的文件，只是能对这个文件进行操作。

File类提供了很多方法获取文件本身的状态，如是否存在、隐藏；也可对文件进行创建和删除操作。

文件输入/输出流

FileInputStream & FileOutputStream

FileInputStream & FileOutputStream都用来操作磁盘文件，进行**字节流**读取。

```
File file = new File("./fileOperation/MyFile.txt");
if (file.exists()) {
    try {
        FileInputStream in = new FileInputStream(file);
        byte[] arr = new byte[1024]; // 缓存字节数组
        int len = in.read(arr); // 将文件信息读入缓存数组中 返回int长度
        System.out.println("文件中的信息是："+new String(arr,0,len)); // 将字节转化为
字符串输出
        System.out.println("文件长度是："+file.length());
        System.out.println("文件的绝对路径是："+file.getAbsolutePath());
        System.out.println("文件的最后修改时间是"+file.lastModified());
        in.close();
    } catch (Exception e) {
        // TODO: handle exception
        e.printStackTrace();
```

```
    }
} else {
    try {
        FileOutputStream out = new FileOutputStream(file);
        byte[] arr = "Java程序设计".getBytes(); // 将字符串写入字节数组
        out.write(arr); // 将字节写入文件中
        out.close();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

FileReader & FileWriter

FileInputStream & FileOutputStream只能提供字节或字节数组的读取方法，而汉字在文件中占用两个字符，如果使用字节流，读取不好可能会产生乱码现象。

使用FileReader & FileWriter来读取**字符流**可避免这种现象。

```
File file = new File("./fileOperation/MyFile.txt");
try {
    FileWriter fw = new FileWriter(file);
    String word = "Java程序设计";
    fw.write(word);
    fw.close();
} catch (Exception e) {
    e.printStackTrace();
}

try {
    FileReader fr = new FileReader(file);
    char[] ch = new char[1024];
    int len = fr.read(ch);
    System.out.println("文件中的信息是：" + new String(ch, 0, len));
    fr.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

`InputStreamReader`是 Java 语言中的字符流转换流类，它可以将字节流转换为字符流。

`InputStreamReader` 是从输入流读取字符的类，它可以将输入的字节流解码为字符。它的构造函数接受一个输入流对象作为参数，例如：

```
InputStreamReader reader = new InputStreamReader(new
FileInputStream("file.txt"));
```

`FileReader` 是从文件中读取字符的类，它可以打开文件并从中读取字符。它的构造函数接受文件路径作为参数，例如：

```
FileReader reader = new FileReader("file.txt");
```

总的来说，`InputStreamReader` 是一个通用的输入流读取类，而 `FileReader` 则是一个专门用于从文件中读取字符的类。

带缓存的输入/输出流

缓存是I/O的一种性能优化。缓存流为I/O流增加了内存缓存区。缓存流需要I/O流在前。

BufferedInputStream & BufferedOutputStream

BufferedInputStream类可以对所有InputStream类进行带缓存区的包装以达到性能的优化。

```
BufferedInputStream(InputStream in) // 创建一个32字节的缓存区
BufferedInputStream(InputStream in, int size) // 创建指定大小的缓存区
```

BufferedOutputStream与之类似，但是其有一个**flush()**方法用来将缓存区未存满的数据强制输出完。当调用close()方法时，系统也会将未存满的数据刷新到文件中。

BufferedReader & BufferedWriter

与上述类似，其也有内部缓存机制。

BufferedReader：

- read() 读取单个字符
- readLine() 读取一个文本行，返回字符串。无数据返回null

BufferedWriter：

- write(String s, int off, int len) 写入字符串的某一部分
- newLine() 写入一个换行符
- flush() 刷新该流的缓存
    - BufferedWriter调用write()方法时，数据不会立刻写入，而是先进入缓存区。如果想立刻写入，需要调用flush方法。

```
File file = new File("./fileOperation/蜀道难.txt");
File writeFile = new File("./fileOperation/newText.txt");
try {
    FileReader fReader = new FileReader(file);
    BufferedReader bReader = new BufferedReader(fReader);
    String tempString = null;
    int i = 1;
    while((tempString = bReader.readLine())!=null) {
        System.out.println("第"+i+"行：  "+tempString);
        try {
            BufferedWriter bWriter = new BufferedWriter(new
FileWriter(writeFile,true));
            bWriter.write("第"+i+"行：  "+tempString);
            bWriter.newLine();
            bWriter.close();
        } catch (Exception e) {
            e.printStackTrace();
```

```
            }
            i++;
        }
        bReader.close();
        fReader.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

BufferedReader也可读取字节流创建字符输入流

```
BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream));
```

数据输入/输出流

DataInputStream & DataOutputStream允许应用程序以与机器无关的方式从底层输入流读取基本Java类型数据。P272

DataOutputStream提供将字符串、double数据、int数据、boolean数据写入文件的方法。

反射和注解

反射

反射的本质：动态获取类结构、动态调用类成员

```
JTextField textField = new JTextField();
Class textFieldC = textField.getClass(); // 获取Class对象
```

```
ListAndSet listAndSet = new ListAndSet("a",10);
Class listAndSetClass = listAndSet.getClass();
listAndSetClass.getConstructor(String.class, int.class); // 获取指定的构造方法

Constructor[] constructors = listAndSetClass.getConstructors(); // 获取所有构造方法
Field[] fields = listAndSetClass.getDeclaredFields(); // 获取所有成员变量
Method[] declaredMethods = listAndSetClass.getDeclaredMethods(); // 获取所有成员方
法
```

Annotation注解

声明Annotation类型的关键字为**@interface**

```
public @interface MyAnnotation {
    String name() default "<默认值>";
    int age();
    Class type() default void.class;
}
```

注解	作用
@Decumented	指示某一类型的注释通过javadoc 和类似的默认工具进行文档化
@Inherited	指示注释类型被自动继承
@Retention	指示注释类型的注释要保留多久
@Target	指示注释类型所适用的程序元素的种类

RetentionPolicy枚举 @Retention	作用
SOURCE	不编译Annotation到类文件中，有效范围最小
CLASS	编译Annotation到类文件中，运行时不加载
RUNTIME	在运行时加载到JVM中，有效范围最大

ElementType枚举 @Target	作用
ANNOTATION_TYPE	用于Annotation类型
FIELD	用于成员变量和枚举常量
METHOD	用于方法
LOCAL_VARIABLE	用于局部变量

数据库操作

JDBC

JDBC的全称是Java DataBase Connectiviry，是一套面向对象的应用程序接口，指定了统一的访问各种关系型数据库的标准接口，是连接数据库和Java应用程序的纽带。

JDBC常用类与接口

DriverManager类

DriverManager类用来管理数据库中的所有驱动程序，是JDBC的管理层，作用于用户和驱动程序之间，跟踪可用的驱动程序，并在数据库的驱动程序之间建立连接。

```
Class.forName("com.mysql.jdbc.Driver");
```

Connection接口

Connection接口代表与特定的数据库的**连接**。要对数据表中数据进行操作，首先要获取数据库连接。Connection实例就像在应用程序与数据库之间开通了一条渠道。

Statement接口

Statement接口用于创建向数据库中传递SQL语句的对象，该接口提供了一些方法可以实现对数据库的常用操作。

PreparedStatement接口

PreparedStatement接口继承Statement，用于**执行动态的SQL语句**，通过PreparedStatement实例执行的SQL语句，将被预编译并保存到PreparedStatement实例中，从而可以反复地执行该SQL语句。

ResultSet接口

ResultSet接口类似于一个临时表，用来暂时存放数据库查询操作所获得的结果集。

数据库操作

```
String driver = "jdbc:mysql://127.0.0.1:3306/test“;
Connection con = DriverManager.getConnection(driver, "root", “123456");
Statement stmt = con.createStatement();
ResultSet res = stmt.executeQuery("selete * from tb_stu");
while (res.next()) {
    System.out.println(res.getString("id"));
}
```

模糊查询

使用“%”代替0个或多个字符，使用“_”代替一个字符。

Java绘图

Java绘图类

Graphics类

Graphics类是所有图形上下文的抽象类。Graphics2D类继承Graphics类，实现了更强大的绘图操作合集。

语法

- draw(Shape form) 画某个图形
- fill(Shape form) 填充某个图形
- Color col = new Color(int r, int g, int b) 创建Color对象
- g2.setColor(Color color) 设置颜色
- BasicStroke(float width, int cap, int join) 创建画笔
- setStroke(Stroke stroke) 设置画笔
- drawImage(Image img, int x, int y, ImageObserver observer) 绘制图像
- drawImage(Image img,

  int x, int y,

  int width, int height,

ImageObserver observer) 图像缩放 利用宽高

Swing程序设计

常用组件

`container` 是主容器，组件都在这个范围内

`JLabel` 内容可以用html标签

`setBounds(距离左边x,距离上边y,高度,宽度)` 设置窗体的坐标和大小，对于`JFrame`是设置距离外部窗口的位置，对于`container`里的就是相对灰色的`container`位置

Jdialog

```
public JDialog(Frame f, boolean mode)
```

设置`model`为`true`时，打开对话框时，阻塞主窗体不可操作。

`JButton`单击事件

```
JButton bl=new JButton("弹出对话框");
bl.setBounds(10,10,100,21);
bl.addActionListener(new ActionListener(){//为按钮添加单击事件
    public void actionPerformed(ActionEvent e){//单击事件触发的方法

    }
})
container.add(bl);
```

常用面板

JPanel 面板

必须在窗体容器中使用，无法脱离窗体显示

```
Jpanel p1=new JPanel(new GridLayout(1,4,10,10));//初始化面板，使用1行4列的网格布局，组件水平间隔10像素，垂直间隔10像素
```

JScrollPane 滚动面板

```
Container c=getContentPane();
JTextArea ta=new JTextArea(20,50);//创建文本区域组件，文本域默认大小为20行、50列
JScrollPane sp=new JScrollPane(ta);//创建滚动面板，并将其文本域放到滚动面板中
c.add(sp);
```

图片路径

```
Icon con=new ImageIcon("src/注意.png");//使用字符串作为路径，是以项目文件夹为根目录
URL url=MyImageIcon.class.getResourse("注意.png");//getResourse是以类所在文件夹为根目录
Icon icon=new ImageIcon(url);//创建Icon对象
```

当用new File()时相对路径是相对于项目的路径，例如JavaSE下面有src，src下面有包，包里有类，当在类中用new File()相对路径访问src下的文件时应该是

```
File file=new File("src/[文件名]");
```

常用布局设计

null布局

绝对布局也叫null布局，其特点是硬性指定组件在容器中的位置和大小，组件的位置通过绝对坐标的方式来指定。

```
Container.setLayout(null);
Component.setBounds(int x, int y, int width, int height);
```

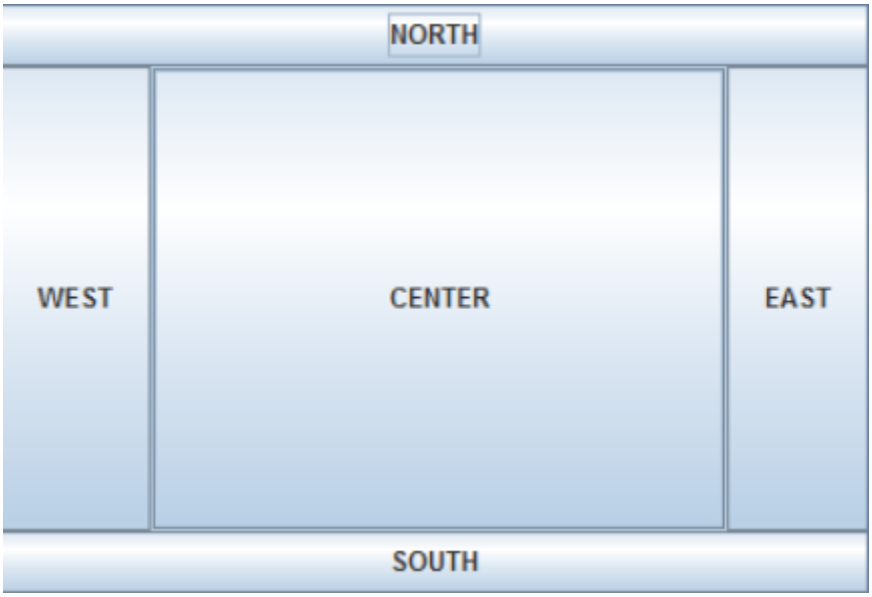
FlowLayout流布局

在使用流布局时，组件从左到右进行摆放，当组件占据了当前行的所有空间时，溢出的组件会被移动到下一行。默认情况下，行组件的**排列方式为居中对齐**（可更改）。

```
// 使用流布局，组件右对齐，组件之间水平间隔为10像素，垂直间隔为10像素
Contrainer.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 10));
```

BorderLayout边界布局

使用Swing创建窗口后，容器的**默认布局管理器**是**边界布局管理器**，它将整个容器划分为5个如图所示的区域。



向容器内添加组件时，如果不指定要把组件添加到哪个区域，则**默认**会添加到**CENTER区域**；如果同一个区域添加多个组件，后放入的组件会**覆盖**先放入的组件。

```
Container c = getContentPane();
setLayout(new BorderLayout());
JButton ...
c.add(centerBtn, BorderLayout.CENTER); // 向CENTER区域添加按钮
c.add(northBtn, BorderLayout.NORTH);
```

GridLayout网格布局

网格布局管理器能把容器划分为网络，组件可以按行列进行排序。设置完布局后，添加的**每个组件都会填满整个网格**；改变窗体大小时，组件的大小也会随之改变。

```
// 设置7行3列的网络，水平间距5像素，垂直间距5像素
setLayout(new GridLayout(7, 3, 5, 5));
```

事件监听器

在Swing中，组件本事是没有任何功能的，需要添加监听器以实现功能。常用的有3种事件监听器：动作事件监听器、键盘事件监听器、鼠标事件监听器。

ActionEvent动作事件

相关定义	实现方法
事件名	ActionEvent
事件源	JButton、JList、JTextField等组件
监听接口	ActionListener
添加监听方法	addActionListenser()
删除监听方法	removeActionListenser()

```
JButton jb = new JButton();
jb.addActionListener(new jbAction());

class jbAction implements ActionListener{
    public void actionPerformed(ActionEvent arg0){
        new JOptionPane("我被点击了");
    }
}
```

在本实例中，为按钮设置了动作动作监听器。由于获取事件监听时需要获取Action Listener接口对象，所以定义了一个jbAction类实现**ActionListener接口**（也可使用匿名类），同时在该类中实现了**actionPerformed()方法**。

KeyEvent键盘事件

当向文本框输入内容时，将发生键盘事件。KeyEvent类负责捕获键盘事件，可以通过为组件添加**KeyListener接口**的监听器类来处理相应的键盘事件。

事件名	作用
keyTyped(KeyEvent e);	发生击键时被触发
keyPressed(KeyEvent e);	按键被按下(手指按下键盘并未松开)时被触发
keyReleased(KeyEvent e);	按键被释放时被触发

```
JTextField textField = new JTextField();
KeyBoard(){
    textField.addKeyListener(new KeyAdapter() {
        char word;
        @Override
        public void keyPressed(KeyEvent e) { // 按键按下
            word = e.getKeyChar();
            // 遍历按钮数组，当word与btns值相同时变成绿色
        }

        @Override
        public void keyReleased(KeyEvent e) { // 按键释放
            word = e.getKeyChar();
            // 遍历按钮数组，当word与btns值相同时变成白色
        }
    });
```

```
}
```

MouseEvent鼠标事件

所有组件都能发生鼠标事件，MouseEvent类负责捕获鼠标事件，可以为组件添加实现了MouseListener接口的监听器来处理对应的鼠标事件。

事件名	作用
mouseEntered(MouseEvent e);	鼠标移入组件时被触发
mousePressed(MouseEvent e);	鼠标按键被按下时被触发
mouseReleased(MouseEvent e);	鼠标按键被释放时被触发
mouseClicked(MouseEvent e);	发生单击事件时被触发
mouseExited(MouseEvent e);	鼠标移出组件时被触发

```
JLabel jLabel = new JLabel();
public MouseEventExample(){
    jLabel.addMouseListener(new MouseListener() {
        @Override
        public void mouseClicked(MouseEvent e) {
            //..
        }
    });
}
```

可以使用getButton()方法获取返回的int值。BUTTON1->1->鼠标左键 BUTTON2->2->鼠标滚轮 ...

多线程

创建线程

主要有继承Thread类和实现Runnable接口。工作代码写在`run()`方法中。

线程的生命周期

线程具有生命周期，共有7种状态：出手状态、就绪状态、运行状态、等待状态、休眠状态、阻塞状态、死亡状态。

- **出生状态**就是**线程被创建时处于的状态**，在用户使用该线程实例调用`start()`方法之前线程都是出生状态。
- 当**调用**`start()`**方法**后，线程处于**就绪状态**(可执行状态)
- 当线程**得到系统资源**进入**运行状态。**
- 处于运行状态调用`Thread`类中的`wait()`**方法**时，进入**等待状态。**进入等待状态必须由**其他的线程调用**`notify()`唤醒，自己无法唤醒自己，`notifyAll()`方法将所有处于等待状态下的线程唤醒
- `sleep()`**休眠状态**，**时间到了会自动唤醒**，区别于等待状态。

- 线程中运行状态下发出**输入/输出请求**时，该线程将进入**阻塞状态。**等待输入输出结束时线程进入**就绪状态。**即使系统资源空闲，线程依然不能回到运行状态。
- 当线程的`run()`方法执行完毕时，线程进入**死亡状态。**

线程的操作方法

线程的加入

假如存在一个线程A，现在需要插入线程B，并要求线程B先执行完毕，然后再继续执行线程A，此时需要使用Thread类中的`join()`方法。

插入到主线程：

```
public static void main(Stirng[] args){
    类 A=new 类();
}
public 类{
    线程实例化;
    线程.start;
    try{
        线程.join();//这样就插入到主线程
    }catch(..){
        e.printStackTrace();
    }
}
```

线程的中断

如果线程是因为使用了`sleep()`或者`wait()`方法进入了就绪状态，可以使用`interrupt()`方法使线程离开`run()`方法，同时结束进程，程序会抛出`InterruptedException`异常。

线程的礼让

Java提供一种礼让方式，用`yield()`方法表示，它只是给当前正在处于运行状态的线程一个提醒，告诉它可以将资源礼让给其他线程，但这仅是一种暗示，没有任何一种机制保证当前线程会将资源礼让。

`yield()`方法使具有同样优先级的线程有进入可执行状态的机会。对于支持多任务的操作系统来说，无需调用。

线程的优先级

每个线程都有自己的优先级，范围1~10，默认为5。每个新产生的线程都会继承父线程的优先级。使用`setPriority()`方法调用。

线程同步

线程同步机制

解决多线程资源冲突问题的方法基本上都是才用给定时间只允许一个线程访问共享资源的方法，这就需要给共享资源上一道锁。

同步块与同步方法

Java的同步机制使用`synchronized`关键字，使用该关键字包含的代码块称为同步块，也称临界区。在方法前加`synchronized`为同步方法。

```
synchronized(Object){
    ...
}
```

通常**将共享资源的操作放置**在`synchronized`定义的区域内，当其他线程获取这个锁时，就必须**等待锁被释放后才可以进入该区域。**其中`Object`为任意一个对象，有标志位，0和1。**若为0，表示此同步块内存在其他线程**，这是当前线程处于就绪状态，直到同步块中的线程执行完同步块代码后，**该对象标志位设置为1，当前线程开始执行同步块。**

线程中执行n++结果不一定正确，因为n++不是原子操作，其中包含三步包括取值，加一，赋值，中间可能会被打断，导致结果不一定准确。可以加入锁解决。

```
synchronized(Object){
    try{
        n++;
        aomicinteger.incrementAndGet();
        longAdder.increment();
    }
}
```

网络通信

网络协议

- TCP是面向**连接**的可靠协议，效率低，保证确实送达。一对一。
- UDP是面向**无连接**的不可靠协议，效率高，不保证数据可靠的传输。一对多，类似广播。

端口

端口被规定在0~65535之间。类似营业厅的窗口，提供某些服务。

- HTTP 80
- FTP 21
- Tomcat 8080
- MySQL 3306

套接字

套接字(Socket)用于将应用程序与端口连接起来。

客户端(应用程序<–>Socket<–>Port<-)-(->Port<–>Socket<–>应用程序)服务器

SeverSocket可以理解为售后部门的电话

服务器端运行的socket可以理解为客服人员

客户端运行的socket可以理解为顾客

TCP程序

TCP网络程序是指利用Socket类编写通讯程序。P391图片很形象

InetAddress类

主要作用是获取IP地址等

```
InetAddress host = InetAddress.getLocalHost(); // 获取本地地址
InetAddress host = InerAddress.getByName("192.168.0.1"); // 获取指定地址
```

ServerSocket类

ServerSocket类用来表示服务器套接字，其主要功能是等待网络上的“请求”。

```
ServerSocket server = new ServerSocket(8998); // 服务器启动8998端口
```

调用ServerSocket的`accept()`方法，会返回一个和客户端Socket对象相连接的Socket对象。服务器端的Socket对象使用`getOutputStream()`方法获得的输出流将指向客户端Socket对象使用的`getInputStream()`方法获得的那个输入流。

Socket类

Socket类主要用于客户端，与服务器进行连接。

```
Socket socket = new Socket("127.0.0.1", 8998); // 连接本地计算机8998端口
```

对话逻辑：

```
Socket socket = new Socket("127.0.0.1", 8998);
PrintWriter wirter = new PrintWriter(socket.getOutputStream(),true);
wirter.printIn(text.getText().trim()); // 将文本框信息写入流
```

UDP程序

UDP主要的通讯模式如下：

- 将数据打包，然后将数据发往目的地
- 接受别人发来的数据包，并查看

发送数据包步骤：

- 使用`DatagramSocket()`创建一个数据包套接字
- 使用`DatagramPacket( byte[] buf , int length , InetAddress address , int port )`创建要发送的数据包
- 使用`DatagramSocket()`类的`send()`方法发送数据包

接受数据包步骤：

- 使用`DatagramSocket(int port)`创建数据包套接字，并指定端口
- 使用`DatagramPacket( byte[] buf , int length)`创建字节数组来接受数据包
- 使用`DatagramSocket()`类的`receive()`方法接受数据包

具体程序设计思路：

```
MulticastSocket socket = new MulticastSocket(port);
DatagarmPacket packet = null;
byte data[] = message.getBytes();

packet = new DatagramPacket(data, data.lenth, iaddress, port);
socket.send(packet);
```

```
byte data[] = new byte[1024];
DatagarmPacket packet = null;
packet = new DatagramPacket(data, data.lenth, group, port);
String message = new String(packet.getData(), 0, packet.getLength());
```

—— Jerry Kim

错题

【单选题】执行下面程序段后，输出结果是__C__。

int a=4,b=6,c=8;

System.out.println(++a*b---c);

A.11

B.16

C.22

D.23

是b-- - c

【单选题】执行下面程序段后，其控制台输出结果是A_。

for (int a=1,b=0;a<2;++a,b+=a)

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

A.b=0

B.b=1

C.b=2

D.b=3

第一次进入循环时为初值

【单选题】在Java语言中，下面关于组件定义错误的是 D

A.TextField tf=new TextField(3);

B.Timer tr=new Timer();

C.JFileChooser jf=new JFileChooser();

D.TextArea ta=new TextArea(3);

```
TextArea ta=new TextArea(3,4); // 需要指定行数和列数
TextField  tf=new TextField(3); // 可以只指定行数
```

【单选题】在Java语言中，下面不属于ComponentEvent的子类是D_。

A.InputEvent

B.FocusEvent

C.WindowEvent

D.ItemEvent

ComponentEvent是AWT包中的事件类，它是用于处理组件事件的基类。ComponentEvent的子类包括：

A.InputEvent：输入事件类，包含键盘和鼠标事件。

B.FocusEvent：焦点事件类，处理组件焦点相关的事件。

C.WindowEvent：窗口事件类，处理窗口相关的事件。

D.ItemEvent不属于ComponentEvent的子类。ItemEvent是AWT包中的事件类，它是用于处理列表、复选框、单选按钮等项目事件的基类。

【单选题】在Java语言中，下面关于Scanner类描述错误的是C。

A.Scanner类可以方便的完成输入流的输入操作

B.Scanner sc=new Scanner(System.in);//从标准输入中扫描

C.Scanner类位于javax.util包中，使用时需要import导入

D.Scanner可以扫描指定的文件

Scanner类位于**java.util包**中

【单选题】下面赋值语句不合法的是D_。

A.Long a=(Long)(long)3;

B.Long b=3L;

C. Long c=Long.parseLong("3");

D.Long d=(Long)3;

long 是基本类型，Long是 long的包装类。3为int型，不能直接从基本类型转换为包装类。

【单选题】java语言中，在定义类时不能使用的修饰符是_B_。

A.public

B.private

C.abstract

D.final

public表示该类对所有类可见；abstract表示该类是抽象类；final表示该类是不可继承的。

private 修饰符**只能用于修饰类的成员变量和成员方法**，表示只能在本类内访问

【单选题】在Java语言中，下面符合数组定义格式的是_B_。

A.int a[3]={1,2,3}; // 长度要在中括号内定义

B.int b[][]={{1,2},{2,3}};

C.int c[3]=new int [3]; // 不能同时前后都有数字

D.int d[]={1.5,2}; // 有小数

Java 中数组定义的常见格式有以下三种：

1. int a[]=new int[3];
2. int b[]={1,2,3};
3. int c[][]={{1,2},{2,3}};

【单选题】下面关于java.sql包中接口和类的描述不正确的是_B_。

A.Connection 接口：表示数据库连接

B.DriverManager类：表示驱动器

C.ResultSet接口：表示SQL查询语句返回的结果集

D.Statement接口：负责执行SQL语句

DriverManager 类是用于**管理数据库驱动程序**的类，不是驱动器本身

【单选题】下面不属于Java语言中常见事件类型的是C___。

A.KeyEvent B.MouseEvent **C.TouchEvent** D.ItemEvent

Java 语言中常见事件类型有以下四种：

1. KeyEvent：键盘事件，例如键盘按下、抬起等。
2. MouseEvent：鼠标事件，例如鼠标点击、拖动等。
3. WindowEvent：窗口事件，例如窗口关闭、最大化等。
4. ActionEvent：操作事件，例如按钮点击、菜单选择等。

【单选题】在Java语言中，下面关于颜色定义不合法的是B___。

A.Color c1=new Color(0xffffff)

B.Color c2=new Color(Color.BLUE)

C.Color c3=new Color(0,0,255)

D.Color c4=new Color(0.2f,0.6f,1.0f)

Color的形参**需要数字**，而不是直接的Color.BLUE

【单选题】现有 “int result=0,jian=3;float x=30.5f,y=0.95f;”两个语句，下面对“result=jian*x*y;”语句修改后依然存在错误的是__**C**_。

A.result=(int)((int)jian*x*y);

B.result=(jian*(int)(x*y));

C.result=(int)jian*x*y;

D.result=jian*(int)x*(int)y;

C中从左到右计算，最后的y是float型，最后的结果也是float型，而result是int，故发生错误。

【单选题】在Java语言中，下面关于接口错误描述的是__A。

A.接口不仅包括方法的特征，还有方法的实现。

B.接口只允许public 和abstract修饰。

C.接口中的**属性**只能被public 、final、static修饰。

D.一个类可以实现多个接口。

接口没有实现。注意B、C两个选项的正确性。

【单选题】在Java语言中，下面关于List不正确的描述是__**B**_。
A.List是在java.util包中
B.List是一个类
C.List具有get(int index)方法
D.List是一个接口

List是接口，不是类！只有实现其中方法的`ArrayList`才可使用具体对象。List 接口继承了 Collection 接口。

【单选题】在Java语言中，下面相关描述错误的是___B___。
A.File类对象对应于系统中的一个目录或文件
B.CharArrayReader 是一个把字符数组作为源的输出流的实现
C.FileInputStream：以字节流方式读取
D.FileReader：把文件转换为字符流读入

CharArrayReader 是一个把字符数组作为源的**输入流**的实现。FileInputStream：以字节流方式读取，与FileReader不同。

【单选题】在Java语言中，下面关于RandomAccessFile描述错误的是D__

A.实现DataInput和DataOutput接口
B.getFilePointer()方法：返回此文件中的当前偏移量 C.readFloat()方法：从此文件读取一个 float
D.writeChar(int v)：按双字节值将char写入该文件，先写低字节

writeChar(int v) 方法是将 char 类型的 **v 值以单字节的形式**写入该文件，不是双字节。

【单选题】下面哪项不属于Statement接口提供的3个执行SQL语句的方法_**A**__。

A.executeDelete(String sql)
B.executeUpdate(String sql)
C.executeQuery(String sql)
D.execute(String sql)

executeDelete(String sql) 是**不存在的**
Statement 接口提供的 3 个执行 SQL 语句的方法是：

- executeUpdate(String sql)：执行 INSERT、UPDATE 或 DELETE 语句以及 SQL DDL（数据定义语言）语句，如 CREATE TABLE 和 DROP TABLE。
- executeQuery(String sql)：执行 SELECT 语句，并返回结果集。
- execute(String sql)：执行任何类型的 SQL 语句，如 SELECT、INSERT、UPDATE 或 DELETE，以及 SQL DDL 语句。

定义int A=5，执行"System.out.println("a="+(A<5)5.1:4))"语句的结果是B
(A) a=5.1
(B) a=4.0
(0) a5
(D) a=4

由于5.1的关系，这里的4会转变成4.0

Applet 是 Java 语言中的类，它继承自 java.awt.Applet 类，是一种小型的 Java 应用程序，可以在浏览器或其他容器中运行。

Applet 可以和 HTML 页面结合使用，使用 <applet> 标签在 HTML 页面中嵌入 Applet。

Applet 具有以下特点：

- 与 HTML 页面结合使用，可以在 HTML 页面中嵌入。
- 可以在浏览器或其他容器中运行。
- 由于运行在客户端，可以使用较多的系统资源，但需要考虑安全性问题。

[单选题]在Java语言中，下面关于TexiField组件相关方法描述不正确的是 C
(A)setTexd(Strings) 方法是设置文本框中的文本为s
(B)addActionListener () 方法是向文本框增加动作监视器
(c) char getText()方法是获取文本框中的文本
(D) setChar(charc) 方法是设置文本框中的回显字符，只显示字符c

getText()方法返回类型为**String**

[单选题]在Java语言中，下面关于异常的错误描述是？
(A) java.lang. Exception类是所有异常的父类
(B)java.lang.NullPointerException是空指针异常类
(C)当异常产生时，程序会自动跳转到异常处理程序
(D)异常是java提供的用于处理程序中错误的一种机制

[单选题]在Java语言中，关于final修饰符的说法不正确的是
(A) final成员变量表示常量，只能被赋值一次，赋值后值不再改变
(B) final类不能被继承，没有子类，final类中的方法默认是final的
(C)final能用于修饰构造方法
(D) final方法不能被子类的方法覆盖，但可以被继承

[单选题]在Java语言中，下面关于File类描述错误的是
(A)执行File f=new Filer("e:\\txx.txt")语句的结果是在e盘上创建了一个txx.txt文件
(B) File类对象对应于系统中的一个目录和文件
(C) File类对象描述文伴名、可否读写等属性，但不读写文件(D)一旦创建，File对象表示的抽象路径名将不会改变

执行File f=new File("e:\txx.txt")语句的结果是**创建一个File对象**，表示e盘上的txx.txt文件，并不会在e盘上创建一个txx.txt文件。注意其他选项。

知识点整理

- 接口没有构造方法
- PreparedStatement继承自Statement
  CallableStatement继承自PreparedSatement
- 在编译Java应用程序源程序文件时，会生成相应的字节码文件。这些字节码文件的扩展名为`.class`
- 字节Byte 比特bit 1Byte = 8 bit
  int变量占用4字节 char占用1字节
- float变量在赋值时要在数字后面加f
- 将小数强转为int型时是直接把小数后面的数字抹去