Java 习题集

学　　院　计算机科学与技术学院

班　　级

学　　生　banban

学　　号

授课老师

二〇二二 年 十一 月 十一 日

一、选择题

1 如下代码的结果是什么 ？

```
class Base {
   Base() {
      System.out.print("Base");
   }
}

public class Alpha extends Base {
   public static void main( String[] args ) {
      new Alpha(); //调用父类无参的构造方法
      new Base();
   }
}
```

A. Base
B. BaseBase
C. 编译失败
D. 代码运行但没有输出
E. 运行时抛出异常

2 下面的方法，当输入为 2 的时候返回值是多少?

```
public static int getValue(int i) {
      int result = 0;
      switch (i) {
            case 1:
                  result = result + i;
            case 2:
                  result = result + i * 2;
            case 3:
                  result = result + i * 3;
      }
      return result;
}
```

0
2
4
10

3 下列说法正确的有

class 中的 constructor 不可省略
constructor 必须与 class 同名，但方法不能与 class 同名　　//可以同名,public void 类名() {}
constructor 在一个对象被 new 时执行
一个 class 只能定义一个 constructor　　//重载

4 下面的程序执行结果是：

```
public static void main(String[] args) {
        int i = 0;
        for (i++; i++ < 10; i++);      // 有;号
        System.out.println(++i);
}
```

输出 0

输出 13

输出 1

输出 12

5 下列哪些代码符合 java 规范：

public static void main(){}//可以这样写,但这样写就不是 main 方法

public method(){ private Integer num; }//局部变量必须赋初值，需要 void

public method(){ Integer num; }//局部变量必须赋初值，需要 void

public method(){ int num; System.out.println(num); }//局部变量必须赋初值，需要 void

private person=5;//没有写数据类型,

6 请看下面的程序段

```
public class P1_06 {
        long[] a = new long[10];
        public static void main(String[] args) {
                System.out.println(a[6]);
        }
}
```

哪一个选项是正确的（　　）。

不输出任何内容

输出 0

当编译时有错误出现//类加载的时候,先加载静态成员和静态代码块,非静态的成员不能放进静态代码块中,类加载:1.静态成员和静态初始化块 2.实例成员和实例初始化块 3.构造方法

当运行时有错误出现

7 下列方法的声明中不合法的是

float area( ){…}

void area( ){…}

area{…}//方法需要加()

int area(int r){…}

8 有一个类 A，对于其构造函数的声明正确的是（　　）。

void A(int x){…}

public A(int x){…}//构造函数没有返回值

A A(int x){…}

int A(int x){…}

9 表达式(short)10/10.2*2 运算后结果是什么类型？
short
int
double//默认 double(强制类型转换有就近原则)
float

10 子类构造方法的哪个地方可以调用其父类的构造方法
任何地方
构造方法的第一条语句
构造方法的最后一条语句
无法在子类构造方法中调用父类的构造方法

11 当 n=5 时，下列函数的返回值是：

```
int foo(int n) {
    if(n<2) return n;
    return foo(n-1)+foo(n-2);
}
```

5
7
8
10

12 下面是 findSum（int m,int n）方法的定义，方法调用 findSum（1，5）的返回结果是( )

```
int findSum(int m, int n) {
    int sum = 0;
    for(int i = m; i <= n; i++){
        sum += i;
    }
    return sum;
}
```

1
5
10
15

13 以下类型为 Final 类型的为
HashMap
StringBuffer//public final class
String//final 修饰变量就变为常量,final 修饰方法这个方法就不能重写,final 修饰类这个类就不能被继承。
Hashtable

14 关于下面的一段代码，以下哪些说法是正确的：

```
public static void main(String[] args) {
        String a = new String("myString");//这里先 new 了一个"myString"字符串放到了常量池里,又 new 了一个新的对象"myString"。
        String b = "myString";              //b 使用常量池里的
        String c = "my" + "String";     //c 也一样
        String d = c;          //d 也一样.所以 b == c == d
        System.out.print(a == b);
        System.out.print(a == c);
        System.out.print(b == c);
        System.out.print(b == d);
}
```

System.out.print(a == b)打印出来的是 false
System.out.print(a == c)打印出来的是 true
System.out.print(b == c)打印出来的是 false
System.out.print(b == d)打印出来的是 true

15 关于 Java 中的数组，下面的一些描述，哪些描述是准确的：（　　）
数组是一个对象，不同类型的数组具有不同的类
数组长度是可以动态调整的//数组长度不可以动态调整,但是 ArrayList 类型的数组可以
数组是一个连续的存储结构
一个固定长度的数组可类似这样定义: int array[100]　//Java 中不可以，必须 new
两个数组用 equals 方法比较时，会逐个便利其中的元素，对每个元素进行比较//比较对象是非相等,比较的是地址值
可以二维数组，且可以有多维数组，都是在 Java 中合法的

16、What is Static Method in Java（）
It is a method which belongs to the class and not to the object(instance)
//静态方法属于类,不属于对象(实例化)
A static method can access only static data. It can not access non-static data (instance variables)
//静态方法只能访问静态数据,不能访问非静态数据(实例化的变量)
A static method can call only other static methods and can not call a non-static method from it.
//静态方法只能访问静态方法,不能访问非静态的方法。
A static method can not be accessed directly by the class name and doesn’t need any object
//静态方法不能通过类名直接访问,也不需要任何对象//静态方法可以通过类名直接访问

17 下列属于关系型数据库的是
1.Oracle2. MySql3. IMS4. MongoDB
//Oracle，SqlServer，Informix，MySql，SyBase 等都是关系数据库

18 下列说法正确的是
JAVA 程序的 main 方法必须写在类里面
JAVA 程序中可以有多个名字为 main 方法
JAVA 程序中类名必须与文件名一样//应该是要和 public 类名一致
JAVA 程序的 main 方法中，如果只有一条语句，可以不用{}（大括号）括起来

1、abstract 和 final 可以同时作为一个类的修饰符。（ ）
A 正确　　B 错误
题解：
abstract 修饰一个类，这个类肯定可以被继承，但是 final 类是不能继承的，所以有矛盾，肯定不能同时用

2、下列描述中，错误的是
A SQL 语言又称为结构化查询语言
B java 中”static”关键字表明一个成员变量或者是成员方法可以在没有所属的类的实例变量的情况下被访问
C 面向对象开发中，引用传递意味着传递的并不是实际的对象，而是对象的引用，因此，外部对引用对象所做的改变不会反映到所引用的对象上 // 会改变，值传递不会改变
D java 是强类型语言，javascript 是弱类型语言
E 面向对象的三大特性包括：封装，继承，多态
题解：
值传递，传递的是原来值的副本。
引用传递，除了一些特殊的（String，包装类属于不可变类），一般的引用类型在进行传递的时候，一开始形参和实参都是指向同一个地址的，这个时候形参对对象的改变会影响到所引用的对象上。

3、关于 protected 修饰的成员变量，以下说法正确的是
A 可以被该类自身、与它在同一个包中的其它类、在其它包中的该类的子类所访问
B 只能被该类本身和该类的所有的子类访问
C 只能被该类自身所访问
D 只能被同一个包中的类访问
题解：

4、以下哪个类包含方法 flush()？（）
A InputStream
B OutputStream
C A 和 B 选项都包含
D A 和 B 选项都不包含
题解：
flush（）函数强制将缓冲区中的字符流、字节流等输出，目的是如果输出流输出到缓冲区完成后，缓冲区并没有填满，那么缓冲区将会一直等待被填满。所以在关闭输出流之前要调用 flush（）。

5、设 m 和都是 int 类型,那么以下 for 循环语句的执行情况是( )
for (m = 0, n = -1; n = 0; m++, n++)
n++;
A 循环体一次也不执行 循环体执行一次 是无限循环 有限次循环 循环结束判断条件不合法 运行出错
B 循环体执行一次 是无限循环

C 有限次循环
D 循环结束判断条件不合法 // 即使==也不循环
题解：
判断条件写的是：n = 0，错误，判断条件应该返回 Boolean 值，应该写：n == 0.

6、下列关于 final、finally、finalize 说法正确的是（ ）多选
A final 可以用来修饰类、方法、变量
B finally 是 java 保证重点代码一定要被执行的一种机制
C 变量被 final 修饰后不能再指向其他对象，但可以重写
D finalize 设计的目的是保证对象在被垃圾收集前完成特定资源的回收
题解：
被 final 修饰后的变量不可变
被 final 修饰后的方法不可被重写
被 final 修饰后的类不可被继承

7、Java 的集合框架中重要的接口 java.util.Collection 定义了许多方法。选项中哪个方法是 Collection 接口所定义的（ ）多选
A int size()
B boolean containsAll(Collection c) // java.util.ArrayList
C compareTo(Object obj)
D boolean remove(Object obj)

题解：
compareTo 是接口 Comparable 中的方法

8、关于 ThreadLocal 类 以下说法正确的是
A ThreadLocal 继承自 Thread // ThreadLocal 继承 Object
B ThreadLocal 实现了 Runnable 接口 // 没有实现任何接口
C ThreadLocal 重要作用在于多线程间的数据共享 // 限制共享数据
D ThreadLocal 是采用哈希表的方式来为每个线程都提供一个变量的副本
E ThreadLocal 保证各个线程间数据安全，每个线程的数据不会被另外线程访问和破坏
题解：
ThreadLocal 继承 Object，相当于没继承任何特殊的。
ThreadLocal 没有实现任何接口。
ThreadLocal 并不是一个 Thread，而是 Thread 的局部变量。

9、以下程序执行后，错误的结果是（）

```
public class Test {
   private String name = “abc”;
   public static void main(String[] args) {
      Test test = new Test();
      Test testB = new Test();
      String result = test.equals(testB) + “,”;
      result += test.name.equals(testB.name) + “,”;
```

```
        result += test.name == testB.name;
        System.out.println(result);
    }
}
```

A true,true,true

B true,false,false

C false,true,false

D false,true,true

题解：

注意是选错误的，所以选三个

10、下面的 Java 赋值语句哪些是有错误的 （） MULTI CHOICE

A int i =1000;

B float f = 45.0; // float f = 45.0f;

C char s = ‘\u0639’;

D Object o = ‘f’;

E String s = “hello,world\0”;

F Double d = 100; // Double d = Double.valueOf(100);

题解：

B：小数如果不加 f 后缀，默认是 double 类型。double 转成 float 向下转换，意味着精度丢失，所以要进行强制类型转换。

C：是使用 unicode 表示的字符。

D：‘f’ 字符会自动装箱成包装类，就可以向上转型成 Object 了。

F：整数默认是 int 类型，int 类型不能转型为 Double，最多通过自动装箱变为 Integer 但是 Integer 与 Double 没有继承关系，也没法进行转型

二、简答题

1. JDK 和 JRE 有什么区别？

JDK：Java Development Kit
它是 Java 开发运行环境，在程序员的电脑上当然要安装 JDK，JDK 包含 JRE。
JRE：Java Runtime Environment
它是 Java 运行环境，如果不需要开发只需要运行 Java 程序，那么可以只安装 JRE。JRE 中包含虚拟机 JVM。

2. == 和 equals 的区别是什么？

equals 是方法，== 是操作符。equals 方法可被重写。
在基本类型的变量中，== 比较值是否相等，没有 equals 方法。
在引用类型的变量中：== 和 equals 方法都是比较内存地址值是否相等

3. 两个对象的 hashCode()相同，则 equals()也一定为 true，对吗？

不对，两个对象的 hashCode()相同，equals()不一定 true。
代码解读：很显然“通话”和“重地”的 hashCode() 相同，然而 equals() 则为 false，因为在散列表中，hashCode()相等即两个键值对的哈希值相等，然而哈希值相等，并不一定能得出键值对相等。

4. final 在 java 中有什么作用？

final 修饰表示常量、一旦创建不可改变；
final 标记的成员变量必须在声明的同时赋值，或在该类的构造方法中赋值，不可以重新赋值；
final 方法不能被子类重写；
final 类不能被继承，没有子类
final 类中的方法默认是 final 的
final 不能用于修饰构造方法

5. java 中的 Math.round(-1.5) 等于多少？

等于-1。

6. String 属于基础的数据类型吗？

String 不属于基础数据类型，属于引用类型，但不需要 new 来创建对象，因为有常量池机制。

7. java 中操作字符串都有哪些类？它们之间有什么区别？

操作字符串的类有：String、StringBuffer、StringBuilder。

8. String str="i"与 String str=new String("i")一样吗？

不一样

(1) String : final 修饰，String 类的方法都是返回 new String。即对 String 对象的任何改变都不影响到原对象，对字符串的修改操作都会生成新的对象。

(2) StringBuffer : 对字符串的操作的方法都加了 synchronized，保证线程安全。

(3) StringBuilder : 不保证线程安全，在方法体内需要进行字符串的修改操作，可以 new StringBuilder 对象，调用 StringBuilder 对象的 append、replace、delete 等方法修改字符串。

9. 如何将字符串反转？

使用 StringBuilder 或者 stringBuffer 的方法。

```
public static String reverse1(String str) {
    return new StringBuilder(str).reverse().toString();
}
```

10. String 类的常用方法都有那些？

indexOf()：返回指定字符的索引。
charAt()：返回指定索引处的字符。
replace()：字符串替换。
trim()：去除字符串两端空白。
split()：分割字符串，返回一个分割后的字符串数组。
getBytes()：返回字符串的 byte 类型数组。
length()：返回字符串长度。
toLowerCase()：将字符串转成小写字母。
toUpperCase()：将字符串转成大写字符。
substring()：截取字符串。
equals()：字符串比较。

11. 抽象类必须要有抽象方法吗？

不需要，抽象类不一定非要有抽象方法。

13. 抽象类能使用 final 修饰吗？

不能，抽象类必须被继承，方法必须被重写实现，被 final 修饰的类不能被继承和方法重写。

14. 接口和抽象类有什么区别？

(1) 抽象类可以有构造方法，接口中不能有构造方法。
(2) 抽象类中可以有普通成员变量，接口中没有普通成员变量。
(3) 抽象类中可以包含静态方法，接口中不能包含静态方法。
(4) 一个类可以实现多个接口，但只能继承一个抽象类。
(5)接口可以被多重实现，抽象类只能被单一继承。
(6)如果抽象类实现接口，则可以把接口中方法映射到抽象类中作为抽象方法而不必实现，而在抽象类的子类中实现接口中方法

15. java 中 IO 流分为几种？

按功能来分：输入流和输出流
按类型来分：字节流和字符流
字节流：inputStream、outputStream
字符流：reader、writer

16. BIO、NIO、AIO 有什么区别？

(1) BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。
(2) NIO：New IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。
(3) AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

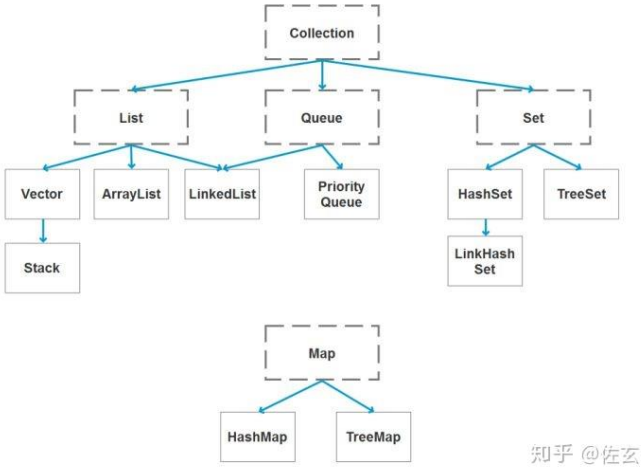
17. Files 的常用方法都有哪些？

Files.exists()：检测文件路径是否存在。
Files.createFile()：创建文件。
Files.createDirectory()：创建文件夹。
Files.delete()：删除一个文件或目录。
Files.copy()：复制文件。
Files.move()：移动文件。
Files.size()：查看文件个数。
Files.read()：读取文件。
Files.write()：写入文件。

二、容器

18. java 容器都有哪些？

常用容器的图录：



19. Collection 和 Collections 有什么区别？

(1) java.util.Collection 是一个集合接口（集合类的一个顶级接口）。它提供了对集合对象进行基本操作的通用接口方法。Collection 接口在 Java 类库中有很多具体的实现。Collection 接口的意义是为各种具体的集合提供了最大化的统一操作方式，其直接继承接口有 List 与 Set。

(2) Collections 则是集合类的一个工具类/帮助类，其中提供了一系列静态方法，用于对集合中元素进行排序、搜索以及线程安全等各种操作。

20. List、Set、Map 之间的区别是什么？

比较	List	Set	Map
继承接口	Collection	Collection	
常见实现类	AbstractList(其常用子类有 ArrayList、LinkedList、Vector)	AbstractSet(其常用子类有 HashSet、LinkedHashSet、TreeSet)	HashMap、HashTable
常见方法	add( )、remove( )、clear( )、get( )、contains( )、size( )	add( )、remove( )、clear( )、contains( )、size( )	put( )、get( )、remove( )、clear( )、containsKey( )、containsValue( )、keySet( )、values( )、size( )
元素	可重复	不可重复(用 equals() 判断)	不可重复
顺序	有序	无序(实际上由HashCode决定)	
线程安全	Vector线程安全		Hashtable线程安全

21. HashMap 和 Hashtable 有什么区别？

(1) hashMap 去掉了 HashTable 的 contains 方法，加上 containsValue() 和 containsKey()方法
(2) hashTable 同步的，而 HashMap 是非同步的，效率上逼 hashTable 要高。
(3) hashMap 允许空键值，而 hashTable 不允许。

22. 如何决定使用 HashMap 还是 TreeMap？

对于在 Map 中插入、删除和定位元素这类操作，HashMap 是最好的选择。然而，假如你需要对一个有序的 key 集合进行遍历，TreeMap 是更好的选择。基于你的 collection 的大小，也许向 HashMap 中添加元素会更快，将 map 换为 TreeMap 进行有序 key 的遍历。

23. 说一下 HashMap 的实现原理？

(1) HashMap 概述：是一个散列表，它存储的内容是键值对(key-value)映射。
(2) HashMap 的数据结构：数组+链表/红黑树

25. ArrayList 和 LinkedList 的区别是什么？

(1) ArrayList 其优点在于随机访问元素
(2) LinkedList 对顺序访问进行了优化，List 中间插入与移除元素的速度很快

26. 如何实现 数组 和 List 之间的转换？

(1) List 转 数组：调用 ArrayList 的 toArray 方法。
(2) 数组 转 List：调用 Arrays 的 asList 方法。

27. ArrayList 和 Vector 的区别是什么？

(1) Vector 是同步的，而 ArrayList 不是。然而，如果你寻求在迭代的时候对列表进行改变，你应该使用 CopyOnWriteArrayList。
(2) ArrayList 比 Vector 快，它因为不同步，不会过载。
(3) ArrayList 更加通用，因为可以使用 Collections 工具类轻易地获取同步列表和只读列表。

28. Array 和 ArrayList 有何区别？

(1) Array 可以容纳基本类型和对象，而 ArrayList 只能容纳对象。
(2) Array 是指定大小的，而 ArrayList 大小是固定的。
(3) Array 没有提供 ArrayList 那么多功能，比如 addAll、removeAll 和 iterator 等。

29. 在 Queue 中 poll()和 remove()有什么区别？

poll() 和 remove() 都是从队列中取出队头元素，但是 poll() 在获取元素失败的时候会返回 null，但是 remove() 失败的时候会抛出 NoSuchElementException 异常。

30. 哪些集合类是线程安全的？

(1) vector：就比 arraylist 多了个同步化机制（线程安全），因为效率较低，现在已经不太建议使用。在 web 应用中，特别是前台页面，往往效率（页面响应速度）是优先考虑的。
(2) statck：堆栈类，先进后出。
(3) hashtable：就比 hashmap 多了个线程安全。

(4) enumeration：枚举，相当于迭代器。

31. 迭代器 Iterator 是什么？

迭代器是一种设计模式，它是一个对象，它可以遍历并选择序列中的对象，而开发人员不需要了解该序列的底层结构。迭代器通常被称为“轻量级”对象，因为创建它的代价小。

32. Iterator 怎么使用？有什么特点？

(1) 使用方法 iterator() 要求容器返回一个 Iterator。第一次调用 Iterator 的 next()方法时，它返回序列的第一个元素。注意：iterator()方法是 java.lang.Iterable 接口，被 Collection 继承。
(2) 使用 next() 获得序列中的下一个元素。
(3) 使用 hasNext() 检查序列中是否还有元素。
(4) 使用 remove() 将迭代器新返回的元素删除。
Iterator 是 Java 迭代器最简单的实现，为 List 设计的 ListIterator 具有更多的功能，它可以从两个方向遍历 List，也可以从 List 中插入和删除元素。

33. Iterator 和 ListIterator 有什么区别？

(1) Iterator 可用来遍历 Set、List、Map 集合，但是 ListIterator 只能用来遍历 List。
(2) Iterator 对集合只能是前向遍历，ListIterator 既可以前向也可以后向。
(3) ListIterator 实现了 Iterator 接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等等。

三、多线程

35. 并行和并发有什么区别？

真正的并行是多核 CPU 下的概念
并发针对单核 CPU 而言，它指的是 CPU 交替执行不同任务的能力
并行针对多核 CPU 而言，它指的是多个核心同时执行多个任务的能力

36. 线程和进程的区别？

进程是操作系统资源分配的基本单位
线程是处理器任务调度和执行的基本单位

37. 守护线程是什么？

通常来说，守护线程经常被用来执行一些后台任务。当希望在程序退出时，或者 JVM 退出时，线程能够自动关闭，此时可以选择守护线程。

38. 创建线程有哪几种方式？

(1) 继承 Thread 类创建线程类

(2) 通过 Runnable 接口创建线程类
(3) 通过 Callable 和 Future 创建线程
(4) 用 JDK 自带的 Executors 来创建线程池对象

39. 说一下 runnable 和 callable 有什么区别？

Runnable 接口中的 run()方法的无返回值，它做的事情只是纯粹地去执行 run()方法中的代码而已；
Callable 接口中的 call()方法是有返回值的，是一个泛型，和 Future、FutureTask 配合可以用来获取异步执行的结果。

40. 线程有哪些状态？

线程通常都有五种状态，创建、就绪、运行、阻塞和死亡。

41. sleep() 和 wait() 有什么区别？

sleep()：方法是线程类（Thread）的静态方法，让调用线程进入睡眠状态，让出执行机会给其他线程，等到休眠时间结束后，线程进入就绪状态和其他线程一起竞争 cpu 的执行时间。因为 sleep() 是 static 静态的方法，他不能改变对象的机锁，当一个 synchronized 块中调用了 sleep() 方法，线程虽然进入休眠，但是对象的机锁没有被释放，其他线程依然无法访问这个对象。
wait()：wait()是 Object 类的方法，当一个线程执行到 wait 方法时，它就进入到一个和该对象相关的等待池，同时释放对象的机锁，使得其他线程能够访问，可以配合 notify，notifyAll 方法来唤醒等待的线程。

42. notify()和 notifyAll()有什么区别？

如果线程调用了对象的 wait() 方法，那么线程便会处于该对象的等待池中，等待池中的线程不会去竞争该对象的锁。

当有线程调用了对象的 notifyAll() 方法（唤醒所有 wait 线程）或 notify()方法（只随机唤醒一个 wait 线程），被唤醒的的线程便会进入该对象的锁池中，锁池中的线程会去竞争该对象锁。

优先级高的线程竞争到对象锁的概率大，假若某线程没有竞争到该对象锁，它还会留在锁池中，唯有线程再次调用 wait() 方法，它才会重新回到等待池中。而竞争到对象锁的线程则继续往下执行，直到执行完了 synchronized 代码块，它会释放掉该对象锁，这时锁池中的线程会继续竞争该对象锁。

43. 线程的 run()和 start()有什么区别？

(1) start()方法来启动一个线程，真正实现了多线程运行。这时无需等待 run 方法体代码执行完毕，可以直接继续执行下面的代码； 这时此线程是处于就绪状态， 并没有运行。 然后通过此 Thread 类调用方法 run()来完成其运行状态， 这里方法 run()称为线程体，它包含了要执行的这个线程的内容，Run 方法运行结束，此线程终止。然后 CPU 再调度其它线程。
(2) run()方法是在本线程里的，只是线程里的一个函数，而不是多线程的。 如果直接调用

run()，其实就相当于是调用了一个普通函数而已，直接待用 run()方法必须等待 run()方法执行完毕才能执行下面的代码，所以执行路径还是只有一条，根本就没有线程的特征，所以在多线程执行时要使用 start()方法而不是 run()方法。

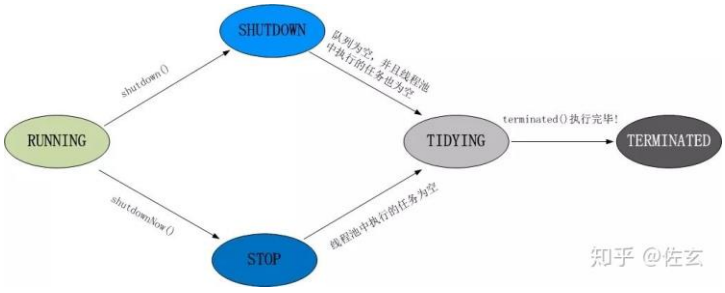
44. 创建线程池有哪几种方式？

1. ThreadPoolExecutor 创建的线程池
ThreadPoolExecutor：最原始的创建线程池的方式。它包含了 7 个参数可供设置

2. Executors 创建的线程池
(1) Executors.newFixedThreadPool：创建一个固定大小的线程池。可控制并发的线程数，超出的线程会在队列中等待
(2) Executors.newCachedThreadPool：创建一个可缓存的线程池。若线程数超过处理所需，缓存一段时间后会回收；若线程数不够，则新建线程
(3) Executors.newSingleThreadExecutor：创建单个线程数的线程池。它可以保证先进先出的执行顺序。它创建单个线程执行任务，如果这个线程异常结束，会创建一个新的替代它
(4) Executors.newScheduledThreadPool：创建了一个固定长度且可以延迟或定时的方式执行任务的线程池。类似于 Timer。
(5) Executors.newSingleThreadScheduledExecutor：创建一个单线程的可执行延迟任务的线程池
(6) Executors.newWorkStealingPool：创建一个抢占式执行的线程池（任务执行顺序不确定）

45. 线程池都有哪些状态？

线程池有 5 种状态：Running、ShutDown、Stop、Tidying、Terminated。
线程池各个状态切换框架图：



46. 线程池中 submit()和 execute()方法有什么区别？

接收的参数不一样
submit 有返回值，而 execute 没有
submit 方便 Exception 处理

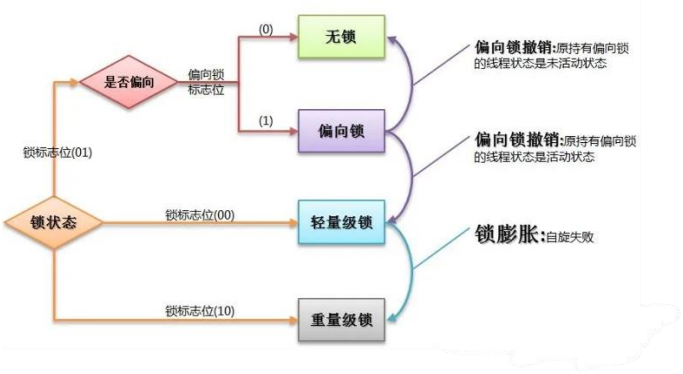
47. 在 java 程序中怎么保证多线程的运行安全？

线程安全在三个方面体现：
(1) 原子性：提供互斥访问，同一时刻只能有一个线程对数据进行操作，（atomic,synchronized）

(2) 可见性：一个线程对主内存的修改可以及时地被其他线程看到，（synchronized,volatile）
(3) 有序性：一个线程观察其他线程中的指令执行顺序，由于指令重排序，该观察结果一般杂乱无序，（happens-before 原则）。

48. 多线程锁的升级原理是什么？

在 Java 中，锁共有 4 种状态，级别从低到高依次为：无状态锁，偏向锁，轻量级锁和重量级锁状态，这几个状态会随着竞争情况逐渐升级。锁可以升级但不能降级。
锁升级的图示过程：



49. 什么是死锁？

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。

50. 怎么防止死锁？

(1) 加锁顺序：线程按照一定的顺序加锁，采用加锁队列
(2) 加锁时限：一个线程尝试去获取锁，如果在指定的时间内获取不到，就放弃等待锁，并释放自己现在所持有的锁，然后随机等待一定时间，再去获取锁
(3) 死锁检测：利用数据结构

51. ThreadLocal 是什么？有哪些使用场景？

线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。
Java 提供 ThreadLocal 类来支持线程局部变量，是一种实现线程安全的方式。但是在管理环境下（如 web 服务器）使用线程局部变量的时候要特别小心，在这种情况下，工作线程的生命周期比任何应用变量的生命周期都要长。任何线程局部变量一旦在工作完成后没有释放，Java 应用就存在内存泄露的风险。

52. 说一下 synchronized 底层实现原理？

synchronized 可以保证方法或者代码块在运行时，同一时刻只有一个方法可以进入到临界区，同时它还可以保证共享变量的内存可见性。
Java 中每一个对象都可以作为锁，这是 synchronized 实现同步的基础：
(1) 普通同步方法，锁是当前实例对象

(2) 静态同步方法，锁是当前类的 class 对象
(3) 同步方法块，锁是括号里面的对象

53. synchronized 和 volatile 的区别是什么？

(1) volatile 本质是在告诉 jvm 当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取； synchronized 则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。
(2) volatile 仅能使用在变量级别；synchronized 则可以使用在变量、方法、和类级别的。
(3) volatile 仅能实现变量的修改可见性，不能保证原子性；而 synchronized 则可以保证变量的修改可见性和原子性。
(4) volatile 不会造成线程的阻塞；synchronized 可能会造成线程的阻塞。
(5) volatile 标记的变量不会被编译器优化；synchronized 标记的变量可以被编译器优化。

54. synchronized 和 Lock 有什么区别？

(1) synchronized 无法判断是否获取锁的状态，Lock 可以判断是否获取到锁；
(2) synchronized 会自动释放锁 a 线程执行完同步代码会释放锁 ； b 线程执行过程中发生异常会释放锁)，Lock 需在 finally 中手工释放锁（unlock()方法释放锁），否则容易造成线程死锁；
(3) 用 synchronized 关键字的两个线程 1 和线程 2，如果当前线程 1 获得锁，线程 2 线程等待。如果线程 1 阻塞，线程 2 则会一直等待下去，而 Lock 锁就不一定会等待下去，如果尝试获取不到锁，线程可以不用一直等待就结束了
(4) synchronized 的锁可重入、不可中断、非公平，而 Lock 锁可重入、可判断、可公平（两者皆可）
(5) Lock 锁适合大量同步的代码的同步问题，synchronized 锁适合代码少量的同步问题
(6) 首先 synchronized 是 java 内置关键字，在 jvm 层面，Lock 是个 java 类

55. synchronized 和 ReentrantLock 区别是什么？

(1) synchronized 是和 if、else、for、while 一样的关键字，ReentrantLock 是类，这是二者的本质区别。
(2) ReentrantLock 的 tryLock()方法可以对获取锁的等待时间进行设置，避免了死锁
(3) ReentrantLock 可以获取各种锁的信息
(4) ReentrantLock 可以灵活地实现多路通知
(5) 另外，二者的锁机制其实也是不一样的:ReentrantLock 底层调用的是 Unsafe 的 park 方法加锁，synchronized 操作的应该是对象头中 mark word。

56. 说一下 atomic 的原理？

(1) Atomic 包中的类基本的特性就是在多线程环境下，当有多个线程同时对单个（包括基本类型及引用类型）变量进行操作时，具有排他性，即当多个线程同时对该变量的值进行更新时，仅有一个线程能成功，而未成功的线程可以向自旋锁一样，继续尝试，一直等到执行成功。

(2) 通过 CAS 乐观锁保证原子性，通过自旋保证当次修改的最终修改成功，通过降低锁粒度（多段锁）增加并发性能。

四、反射

57. 什么是反射？

反射主要是指程序可以访问、检测和修改它本身状态或行为的一种能力
Java 反射：在 Java 运行时环境中，对于任意一个类，能否知道这个类有哪些属性和方法？对于任意一个对象，能否调用它的任意一个方法
Java 反射机制主要提供了以下功能：
在运行时判断任意一个对象所属的类。
在运行时构造任意一个类的对象。
在运行时判断任意一个类所具有的成员变量和方法。
在运行时调用任意一个对象的方法。

58. 什么是 java 序列化？什么情况下需要序列化？

简单说就是为了保存在内存中的各种对象的状态（也就是实例变量，不是方法），并且可以把保存的对象状态再读出来。虽然你可以用你自己的各种各样的方法来保存 object states，但是 Java 给你提供一种应该比你自己好的保存对象状态的机制，那就是序列化。

什么情况下需要序列化：
a）当把内存中的对象状态保存到一个文件中或者数据库中时候；
b）当用套接字在网络上传送对象的时候；
c）当通过 RMI 传输对象的时候；

二、编程题（输入补充程序，运行， 抓图）

1、练习题：面向对象 封装百分制分数

需求：封装百分制分数，和它对应的五档分制分数

```
class Score {
    int score;
    char level;
    public Score(int score) {
        this.score = score;
        //计算五档分数，保存到成员变量 level
        this.level = setLevel(score);
    }
    private char setLevel(int s) {
        char r = 0;
        switch (s / 10) {
            case 10 :
            case 9 : r = 'A';break;
            case 8 :
            case 7 : r = 'B';break;
            case 6 : r = 'C';break;
            case 2 :
            case 1 :
            case 0 : r = 'E';break;
            default: r = 'D';break;
        }
        return r;
    }

    public String toString() {
        return score + ", " + level;
    }
}
public class P4_1 {
    public static void main(String[] args) {
        /*
        A [90,100]
        B [70,90)
        C [60,70)
        D [20,60)
        E [0,20)
        */
        Score s = new Score(54);
        System.out.println(s.toString());
    }
```

```
}
```

2、练习题:面向对象 打印图形
需求：设计一个可以随机打印形状的代码

```
import java.util.Random;
import java.util.Scanner;
// 设计一个可以随机打印形状的代码
// 形状父类
class Shape {
    public void draw() {
        System.out.println("---图形形状---");
    }

    public void clear() {
        System.out.println("---清空---");
    }
}

// 圆形父类
class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("打印一个圆形 o");
    }
}

// 方形父类
class Square extends Shape {
    @Override
    public void draw() {
        System.out.println("打印一个方形 口");
    }
}

// 直线父类
class Line extends Shape {
    @Override
    public void draw() {
        System.out.println("打印一条直线 ———");
    }

    public void length() {
        System.out.println("——— 一米长 ———");
```

```
    }
}

public class P4_2 {
    public static void main(String[] args) {
        System.out.println("按回车继续");
        while(true) {
            int r = new Random().nextInt(4);
            System.out.println(r);
            switch(r) {
                case 0: f(new Shape()); break;
                case 1: f(new Line()); break;
                case 2: f(new Square()); break;
                case 3: f(new Circle()); break;
            }
        }
    }
    /*
     * Shape
     *   |- Line
     *   |- Square
     *   |- Circle
     */
    static void f(Shape s) {
        new Scanner(System.in).nextLine();  // 读回车
        System.out.println("+++++++++++++++");
        s.draw();
        // 向上转型后，只能调用父类定义的通用成员
        // 子类特有成员不能调用
        // s.length();
        // s 对象的真实类型是 Line 类型
        if (s instanceof Line) {
            // 向下转型成 Line 类型，才能调用它特有的方法
            Line line = (Line) s;
            line.length();
        }
        new Scanner(System.in).nextLine(); // 读回车
        s.clear();
    }
}
```

3、练习题：面向对象 设计士兵类

需求：设计士兵与武器 AK47 类，并完成前进、进攻、发射子弹、装载子弹的功能

```java
// 设计士兵与武器 AK47 类，并完成前进、进攻、发射子弹、装载子弹的功能
import java.util.Scanner;
import java.util.Random;

// 士兵类
class Soldier {
    // 成员变量，属性变量
    int id;  // 默认值 0
    int blood = 100;  // 血量
    AK47 a;  // 默认 null 值
    int count = 0;

    public void go() {
        System.out.println(this.id + "号士兵前进");
    }

    public void attack() {
        System.out.println("----------------------");
        if (blood == 0) {
            System.out.println("这是" + id + "号士兵的尸体");
            return;
        }
        System.out.println(id + "号士兵进攻" + ++count);
        if (a != null) {
            a.fire();  // 调用枪发射子弹
        }
        // 随机的减血量
        int d = new Random().nextInt(10) + 1;
        blood -= d;
        if (blood < 0) {  // 不允许负数血量
            blood = 0;
        }
        System.out.println("伤害：" + d + "   剩余血量：" + blood);
        // 血量是 0
        if (blood == 0) {
            System.out.println(id + "号士兵阵亡");
        }
    }
}

// 武器类：
class AK47 {
    int bullets = 100;   // 子弹数量
```

```
    public void fire() {
        if (bullets == 0) {
            System.out.println("没有子弹，请输入'load'装载子弹");
            return;
        }
        // 随机产生发射子弹数量
        int r = new Random().nextInt(10) + 1;
        // 要发射的数量，比现有子弹多
        if (r > bullets) {
            r = bullets;  // 有多少发多少
        }
        bullets -= r;
        System.out.println("-------------------------");
        System.out.println("发射 " + r + " 个子弹" + "   还剩：" + bullets + "
发子弹");
        for (int i = 0; i < r; i++) {
            System.out.print("突");
        }
        System.out.println("~");
        if (bullets == 0) {
            System.out.println("弹夹空了，请输入'load'装载子弹");
        }
    }

    public void load() {
        bullets = 100;
        System.out.println("弹夹已装满");
    }
}

public class P4_3 {
    public static void main(String[] args) {
        //test1();
        test2();
    }

    static void test1() {
        Soldier s1 = new Soldier();
        Soldier s2 = new Soldier();
        // 赋值
        s1.id = 9527;
        s2.id = 9528;
        // 前进
        s1.go();
```

```
        s2.go();
        // 新建 AK47 对象
        s1.a = new AK47();
        s2.a = new AK47();
        for (int i = 0; i < 20; i++) {
            s2.attack();
        }
    }

    static void test2() {
        // 新建 AK47 对象，地址存到变量 a
        AK47 a = new AK47();
        System.out.println("按回车射击，请输入'load'装载子弹");
        do {
            String s = new Scanner(System.in).nextLine();
            if (s.equals("load")) {
                a.load();
                continue;
            }
            a.fire();
        } while (true);
    }
}
```

4、练习题：面向对象 设计宠物类
需求：设计宠物类，用户可以自由选择养猫还是养狗，可以给宠物起名字，还可以实现喂食互动的功能，宠物需要有饱食度和快乐度

```
import java.util.Date;
import java.util.Random;
import java.util.Scanner;

// 宠物类：
class Pet {
    String name;
    int full;
    int happy;

    public Pet(String name) {
        this(name, 50, 50);  // 会调用 public Pet(String name, int full, int happy)
    }
```

```
    public Pet(String name, int full, int happy) {
        this.name = name;
        this.full = full;
        this.happy = happy;
        System.out.println("起始  饱食度：" + full + "  快乐度：" + happy);
    }

    public void feed() {  // 宠物的喂食方法
        if (full == 100) {
            System.out.println(name + "已经吃饱了");
            return;
        }
        System.out.println("给" + name + "喂食");
        full += 10;
        System.out.println("full+10  饱食度：" + full);
    }

    public void play() {  // 宠物的互动玩耍方法
        if (full == 0) {
            System.out.println(name + "已经饿得玩不动了");
            return;
        }
        System.out.println("陪" + name + "玩耍");
        happy += 10;
        full -= 10;
        System.out.println("happy+10  快乐度：" + happy);
        System.out.println("full-10  饱食度：" + full);
    }

    public void punish() {  // 宠物的惩罚方法
        System.out.println("打" + name + "的pp，" + name + "哭叫：" + cry());
        happy -= 10;
        System.out.println("happy-10  快乐度：" + happy);
    }

    public String cry() {  // 小动物被打哭了
        return "此处有哭叫声";
    }
}

// 小猫类：
class Cat extends Pet {
    public Cat(String name, int full, int happy) {
        super(name, full, happy);
```

```
    }

    public Cat(String name) {
        super(name);
    }

    @Override
    public String cry() {
        return "喵~";
    }
}

// 小狗类:
class Dog extends Pet {
    public Dog(String name, int full, int happy) {
        super(name, full, happy);
    }

    public Dog(String name) {
        super(name);
    }

    @Override
    public String cry() {
        return "汪~";
    }
}

public class P4_4 {
    public static void main(String[] args) {
        System.out.println("1. 狗   2. 猫");
        System.out.print("你选择领养：  ");
        int c = new Scanner(System.in).nextInt();

        System.out.print("给宠物起个名字：");
        String name = new Scanner(System.in).nextLine();
//        Dog dog = null;
//        Cat cat = null;
        if (c == 1) {
            Dog dog = new Dog(name);
            play(dog);
        } else {
            Cat cat = new Cat(name);
            play(cat);
```

```
        }
    }

    private static void play(Pet pet) {
        System.out.println("按回车执行");
        while (true) {
            new Scanner(System.in).nextLine();
            int r = new Random().nextInt(3);   // [0, 3)
            switch (r) {
                case 0: pet.feed();break;
                case 1: pet.play();break;
                default: pet.punish();break;
            }
        }
    }
}
```

5. 设计一个形状类 Shape，方法：求周长和求面积
形状类的子类：Rect(矩形)，Circle(圆形)
Rect 类的子类：Square(正方形)
不同的子类会有不同的计算周长和面积的方法

结构：

```
// 形状父类
class Shape1 {
    static double PI = 3.1415;
    public double sum = 0;   // 周长
    public double area = 0;  // 面积
    public double getSum() {}
    public double getArea() {}
}

// 矩形
class Rect extends Shape1 {
    private double length;
    private double wide;
    public Rect(double length, double wide) { }
    public double getSum() {}
    public double getArea() {}
}

// 圆
class Circle1 extends Shape1 {
    private final static double p = 3.1415926;
```

```
    private double radius;
    public Circle1(double radius) {}
    public double getSum() {}
    public double getArea() {}
}
// 正方形
class Square1 extends Shape1 {
    private double sidelength;
    public Square1(double sidelength) {}
    public double getSum() {}
    public double getArea() {}
}
```

解题思路：
计算三个形状类的周长和面积，周长与面积是他们共有的结果，可以单独写一个类进行结果接收，其他类继承父类；形状类定义各自的属性并封装，对属性提供 get() 和 set() 方法；有参构造方法(根据需求而定，这个场景使用有参构造，可以初始化实例时，直接给属性赋值)；重写父类方法(计算周长与面积)

```
// 形状父类
class Shape1 {
    static double PI = 3.1415;
    public double sum = 0;   // 周长
    public double area = 0;  // 面积

    public double getSum() {
        return sum;
    }

    public double getArea() {
        return area;
    }
}

// 矩形
class Rect extends Shape1 {
    private double length;
    private double wide;

    public void setLength(double length) {
        this.length = length;
    }

    public void setWide(double wide) {
```

```
        this.wide = wide;
    }

    public double getLength() {
        return this.length;
    }

    public double getWide() {
        return this.wide;
    }

    public Rect(double length, double wide) {
        this.length = length;
        this.wide = wide;
    }

    @Override
    public double getSum() {
        return 2 * (this.length + this.wide);
    }

    @Override
    public double getArea() {
        return this.length * this.wide;
    }
}

// 圆
class Circle1 extends Shape1 {
    private final static double p = 3.1415926;
    private double radius;

    public Circle1(double radius) {
        this.radius = radius;
    }

    public double getRadius() {
        return this.radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }
```

```
    @Override
    public double getSum() {
        return 2 * PI * this.radius;
    }

    @Override
    public double getArea() {
        return PI * this.radius * this.radius;
    }

}

// 正方形
class Square1 extends Shape1 {
    private double sidelength;

    public Square1(double sidelength) {
        this.sidelength = sidelength;
    }

    public void setSidelength(double sidelength) {
        this.sidelength = sidelength;
    }

    public double getSidelength() {
        return this.sidelength;
    }

    @Override
    public double getSum() {
        return 4 * this.sidelength;
    }

    @Override
    public double getArea() {
        return this.sidelength * this.sidelength;
    }
}

public class P4_5 {
    public static void main(String[] args) {
        Rect r = new Rect(4, 5);        // 矩形
        Circle1 c = new Circle1(4.56); // 圆
        Square1 s = new Square1(5.67); // 正方形
```

```
        r.getSum();
        r.getArea();
        System.out.println("矩形的周长：" + r.getSum() + ";矩形的面积：" +
r.getArea());

        c.getSum();
        c.getArea();
        System.out.println("圆的周长:" + c.getSum() + ";圆的面积:" + c.getArea());

        s.getSum();
        s.getArea();
        System.out.println("正方形的周长：" + s.getSum() + ";正方形的面积：" +
s.getArea());
    }
}
```

6. 设计一个台灯类(Lamp)，其中台灯有灯泡类(Buble)这个属性，还有开灯(on)这个方法。设计一个灯泡类(Buble)，灯泡类有发亮(lampOn)方法，其中有红灯泡类(RedBuble)和绿灯泡类(GreenBuble)，他们都继承灯泡类(Buble)一个发亮的方法。

结构：

```
// 灯泡类
abstract class Buble {
    public abstract String lampOn();
}
class RedBuble extends Buble {
    public String lampOn(){}
}
class GreenBuble extends Buble {
    public String lampOn() {}
}
// 台灯类
class Lamp {
    private Buble buble;
    public void on(Buble buble) {}
}
```

解题思路：

灯泡发光是一个事物的共同行为，可以作为父类，让具体的物品进行继承(使用抽象类而不是普通类，是因为发光的方法是需求必须要求实现的部分)；绿灯与红灯继承发光的方法并重写(每种灯发出光的方式或条件是不一样的，所以需要重写父类方法)；台灯打开开关，然后发光，除了需要电以外，他还需要一个灯泡，所以将灯泡的类实例化作为属性；灯泡作为

父类，可以引用子类对象，最终 Test 类调用台灯开关 ON()，并上传作为参数的灯泡(红灯与绿灯类)，达到需求不同灯泡发出不同光的需求。

```
// 灯泡
abstract class Buble {
    public abstract String lampOn();  //开灯，灯泡发亮啦！！
}

class RedBuble extends Buble {
    @Override
    public String lampOn() {
        return "红灯亮了！";
    }
}

class GreenBuble extends Buble {
    @Override
    public String lampOn() {
        return "绿灯亮了！";
    }
}

class Lamp {
    private Buble buble; // 属性封装(一个类的属性可以是一个实例对象)

    public Buble getBuble() {  //提供读与写的方法
        return this.buble;
    }

    public void setBuble(Buble buble) {
        this.buble = buble;
    }

    public void on(Buble buble) {            // 传入 Buble 类型的对象
        if (buble instanceof GreenBuble) {    // instanceof 严格来说是 Java 中的一个双目运算符
            String g = buble.lampOn();        // instanceof 用来测试一个对象是否为一个类的实例
            System.out.println("打开台灯：" + g);
        } else if (buble instanceof RedBuble) {
            String r = buble.lampOn();        // 父类调用子类重写方法
            System.out.println("打开台灯：" + r);
        }
    }
```

```
}

public class P4_6 {
    public static void main(String[] args) {
        Lamp l = new Lamp();
        Buble r = new RedBuble();   // 父类引用子类对象
        Buble g = new GreenBuble();
        l.on(r);
        l.on(g);
    }
}
```

7. 写一个程序，把若干各种类型的员工放在一个 Employee 数组里，写一个方法，打印出某月每个员工的工资数额
Employee：这是所有员工总的父类
属性：员工的姓名，员工的生日月份

方法：
double getSalary(int month)：根据参数月份来确定工资，如果该月员工过生日，则公司会额外奖励 100 元
SalariedEmployee：Employee 的子类，拿固定工资的员工。属性：月薪
HourlyEmployee：Employee 的子类，按小时拿工资的员工，每月工作超出 160 小时的部分按照 1.5 倍工资发放。属性：每小时的工资、每月工作的小时数
SalesEmployee：Employee 的子类，销售人员，工资由月销售额和提成率决定。属性：月销售额、提成率
BasePlusSalesEmployee：SalesEmployee 的子类，有固定底薪的销售人员，工资由底薪加上销售提成部分。属性：底薪。

要求：
写一个程序，把若干各种类型的员工放在一个 Employee 数组里，
写一个方法，打印出某月每个员工的工资数额。

注意：
要求把每个类都做成完全封装，不允许非私有化属性。

结构：

```
import java.time.LocalDate;
class Employee {  //  员工信息
    private String name;
    private LocalDate birthday;
    public Employee() {}  // 如果没有无参构，直接子类不可有无参构造
    public Employee(String name, LocalDate birthday) {}
```

```
    public void setName(String name) {}
    public void setBirthday(LocalDate birthday) {}
    public String getName() {}
    public LocalDate getBirthday() { }
    public double getSalary(int month) {}
}

class HourlyEmployee extends Employee { // 按小时领取工资
    private double hourLyEmp;
    private int hours;
    public HourlyEmployee() {}
    public HourlyEmployee(String name, LocalDate birthday, int hourLyEmp, int
hours) {}
    public double getSalary(int month) {}
}

class SalariedEmployee extends Employee {  // 拿固定工资
    private double monthLysalary;   //月薪
    public SalariedEmployee(String name, LocalDate birthday, double
monthLysalary) {}
    public double getSalary(int month) {}
}

class SalesEmployee extends Employee { // 销售提成
    private double monthlySales; // 月销售额
    private double commissionRate;  // 提成率
    public SalesEmployee(String name, LocalDate birthday, double monthlySales,
double commissionRate) {}
    public double getSalary(int month) {}
}

class BasePlusSalesEmployee extends SalesEmployee { //底薪 + 提成
    private int baseSalary;
    public BasePlusSalesEmployee(String name, LocalDate birthday, double
monthlySales, double commissionRate, int baseSalary) {}
    public double getSalary(int month) {}
}
```

解题思路：
Employee 类作为员工的基本信息，包含姓名、生日、薪资的属性
其他类(月薪、小时工、销售)分别继承父类信息，有销售底薪+提成的类继承销售提成的类，并实现各自的薪资的计算方式
通过 getSalary 方法进行调用和读取每月各自的薪资信息
这里用到了一个新实例，JDK8 提供获的 LocalDate 类，取时间和写入时间更方便

要实现某个功能或需求，首先是要想清楚实现的方式和逻辑，脑子里要有个大概的框架，条理清晰，才不会被自己绕晕和想复杂(个人小感悟)

```
import java.time.LocalDate;

class Employee {  //员工信息
    private String name;
    private LocalDate birthday;

    public Employee() {}    // 如果父类没有无参构造，直接子类不可有无参构造

    public Employee(String name, LocalDate birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setBirthday(LocalDate birthday) {
        this.birthday = birthday;
    }

    public String getName() {
        return name;
    }

    public LocalDate getBirthday() {
        return birthday;
    }

    public double getSalary(int month) {
        return 0;
    }
}

class HourlyEmployee extends Employee { // 按小时领取工资
    private double hourLyEmp;
    private int hours;

    public HourlyEmployee() {}

    public HourlyEmployee(String name, LocalDate birthday, int hourLyEmp, int
```

```
hours) {
        super(name, birthday);
        this.hourLyEmp = hourLyEmp;
        this.hours = hours;
    }

    public void setHourLyEmp(double hourLyEmp) {
        this.hourLyEmp = hourLyEmp;
    }

    public double getHourLyEmp() {
        return hourLyEmp;
    }

    public int getHours() {
        return hours;
    }

    public void setHour(int hour) {
        this.hours = hour;
    }

    @Override
    public double getSalary(int month) {
        double sum = 0.0;
        if (this.getBirthday().getMonthValue() == month) {
            sum += 100;
        }
        if (hours >= 160) {
            sum += (hours - 160) * 1.5 * hourLyEmp + 160 * hourLyEmp;
        } else {
            sum += hours * hourLyEmp;
        }
        return sum;
    }
}

class SalariedEmployee extends Employee {  // 拿固定工资
    private double monthLysalary;   //月薪

    public double getMonthLysalary() {
        return monthLysalary;
    }
```

```
    public void setMonthLysalary(double monthLysalary) {
        this.monthLysalary = monthLysalary;
    }

    public SalariedEmployee() {}

    public SalariedEmployee(String name, LocalDate birthday, double
monthLysalary) {
        super(name, birthday);
        this.monthLysalary = monthLysalary;
    }

    @Override
    public double getSalary(int month) {
        double num = this.getMonthLysalary();
        if (this.getBirthday().getMonthValue() == month) {
            num += 100;
        }
        return num;
    }
}

class SalesEmployee extends Employee { // 销售提成
    private double monthlySales; // 月销售额
    private double commissionRate;  // 提成率

    public SalesEmployee() {}

    public SalesEmployee(String name, LocalDate birthday, double monthlySales,
double commissionRate) {
        super(name, birthday);
        this.monthlySales = monthlySales;
        this.commissionRate = commissionRate;
    }

    public double getMonthlySales() {
        return monthlySales;
    }

    public double getCommissionRate() {
        return commissionRate;
    }

    public void setMonthlySales(double monthlySales) {
```

```
        this.monthlySales = monthlySales;
    }

    public void setCommissionRate(double commissionRate) {
        this.commissionRate = commissionRate;
    }

    @Override
    public double getSalary(int month) {
        double sum = 0.0;
        if (this.getBirthday().getMonthValue() == month) {
            sum += 100;
        }
        sum += this.getMonthlySales() * this.getCommissionRate();
        return sum;
    }
}

class BasePlusSalesEmployee extends SalesEmployee { //底薪 + 提成
    private int baseSalary;

    public BasePlusSalesEmployee(String name, LocalDate birthday, double
monthlySales, double commissionRate, int baseSalary) {
        super(name, birthday, monthlySales, commissionRate);
        this.baseSalary = baseSalary;
    }

    public int getBaseSalary() {
        return baseSalary;
    }

    public void setBaseSalary(int baseSalary) {
        this.baseSalary = baseSalary;
    }

    @Override
    public double getSalary(int month) {
        double sum = 0.0;
        if (this.getBirthday().getMonthValue() == month) {
            sum += 100;
        }
        sum += baseSalary;  // + 底薪
        sum += this.getMonthlySales() * this.getCommissionRate();//+销售提成
        return sum;
```

```
    }
}

public class P4_7 {
    public static void main(String[] args) {
//      LocalDate now = LocalDate.now();  jdk 8  新提供的方法，获取当前日期
//      LocalDate.of(2018, 9, 24);        jdk 8  写入日期
        Employee[] employees = new Employee[4];  //除了new 一个对象，我们还可以new 一个数组
        HourlyEmployee h = new HourlyEmployee("小时工", LocalDate.of(1995, 12, 17), 50, 160);
        SalariedEmployee s1 = new SalariedEmployee("固定工资", LocalDate.of(1994, 11, 17), 5000);
        SalesEmployee s2 = new SalesEmployee("销售提成", LocalDate.of(1995, 12, 17), 10000, 0.1);
        BasePlusSalesEmployee b = new BasePlusSalesEmployee("底薪+提成", LocalDate.of(1995, 12, 17), 3000, 0.1, 10000);
        employees[0] = h;
        employees[1] = s1;
        employees[2] = s2;
        employees[3] = b;
        for (Employee e : employees) {
            System.out.println("姓名： " + e.getName() + "   " + "薪资： " + e.getSalary(12) + "   " + "生日： " + e.getBirthday());
        }
    }
}
```

8. 完成以下要求，最终创建 Test 类，完成 Test 的操作要求

要求如下：

创建一个 Animal 动物类，要求有方法 eat()方法，方法输出一条语句“吃东西”

创建一个接口 A，接口里有一个抽象方法 fly()

创建一个 Bird 类继承 Animal 类并实现接口 A 里的方法输出一条有语句“鸟儿飞翔”，重写 eat()方法输出一条语句“鸟儿 吃虫”

在 Test 类中向上转型创建 b 对象，调用 eat 方法。然后向下转型调用 eat()方法、fly()方法

结构：

```
interface A {
    public abstract void fly();
}
class Animal {
    public void eat() {}
}
class Bird extends Animal implements A {
    public void eat() {}
```

```
    public void fly() {}
}
public static void main(String[] args) {
    Animal a = new Bird();  // 向上转型
    Bird c = (Bird) a;      // 向下转型
}
```

新知识点：
把子类对象直接赋给父类引用叫 upcasting 向上转型，向上转型不用强制转型，如：Father father = new Son()
把指向子类对象的父类引用赋给子类引用叫向下转型（downcasting），要强制转型，要向下转型，必须先向上转型为了安全可以用 instanceof 判断
如 father 就是一个指向子类对象的父类引用，把 father 赋给子类引用 son 即 Son son =（Son）father，其中 father 前面的（Son）必须添加，进行强制转换
upcasting 会丢失子类特有的方法,但是子类 overriding 父类的方法，子类方法有效，向上转型只能引用父类对象的属性，要引用子类对象属性，则要写 getter 函数
向上转型的作用，减少重复代码，父类为参数，调有时用子类作为参数，就是利用了向上转型。这样使代码变得简洁，体现了 JAVA 的抽象编程思想

```
interface A {
    public abstract void fly();
}

class Animal {
    public void eat() {
        System.out.println("吃东西");
    }
}

class Bird extends Animal implements A {
    @Override
    public void eat() {
        System.out.println("鸟儿 吃虫");
    }

    @Override
    public void fly() {
        System.out.println("鸟儿在飞翔");
    }
}

class AnimalTest {
    public static void main(String[] args) {
```

```
        Bird b = new Bird();
        b.eat();

        Animal a = new Bird(); //父类的引用指向了子类对象，子类对象转为父类。将子类的对象赋值给父类(向上转型)
        a.eat();
        // 多态，执行子类中的方法，体现了多态的特性，同一种事物在不同情形下所表现出来的不同状态，只针对方法，不针对成员变量
        // eat() 方法是子类重写父类方法(必须是父类的重写方法)，所以可以调用。。调用子类中独有的方法就会报错，如a.fly();

        Bird c = (Bird) a;  //子类的引用指向了父类，父类对象转为子类。并将父类强制转换为子类对象(向下转型)
        c.eat();
        c.fly();  // 可以调用子类独有方法，如 fly()
    }
}
```

9. 按照以下场景要求，完成编码设计

结构：

```
abstract class Person {
    private String name;
    private int age;
    public Person(String name, int age){}
    public int getAge(){}
    public String getName(){}
    abstract void eat();
    abstract void sleep();
}
class Student extends Person{
    private int snum;
    public Student(String name, int age) {}
    public int getSnum(){}
    public void study(){}
    public void eat(){}
    public void sleep(){}
}
class Worker extends Person{
    private int wnum;
    public Worker(String name, int age) {}
    public int getWnum(){}
    public void work(){}
    public void eat(){}
```

```
    public void sleep(){}
}
```

创建一个 Person 抽象类
要求含有姓名（name）年龄（age）两个私有属性以及吃饭（eat） 和睡觉（sleep）两个抽象方法，并写出带参构造方法
创建学生（student）和工人（worker） 两个类，继承 Person 类
学生类 多出了私有属性学号和学习方法（输出我爱学习）
工人类 多出了私有属性工号和工作方法（输出我爱工作）
在主函数中创建学生和工人类 的实例对象，使用构造方法赋值，并输出所有属性和方法

```
abstract class Person {
    private String name;
    private int age;

    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }

    public int getAge(){
        return age;
    }
    public String getName(){
        return name;
    }

    abstract void eat();
    abstract void sleep();
}

class Student extends Person{
    private int snum;
    public Student(String name, int age) {
        super(name, age);
    }
    public int getSnum(){
        return snum;
    }
    public void study(){
        System.out.println("我爱学习");
    }
    public void eat(){
        System.out.println("学生吃顿饭");
```

```
    }
    public void sleep(){
        System.out.println("学生睡觉");
    }
}
class Worker extends Person{
    private int wnum;
    public int getWnum(){
        return wnum;
    }
    public void work(){
        System.out.println("我爱工作");
    }
    public Worker(String name, int age) {
        super(name, age);
    }
    public void eat(){
        System.out.println("工人吃顿饭");
    }
    public void sleep(){
        System.out.println("工人睡觉");
    }
}
public class P4_9 {
    public static void main(String [] args){
        Student stu = new Student("ss", 20);
        Worker worker = new Worker("ww", 41);

        System.out.print("学号："+ stu.getSnum() +"  姓名为："+ stu.getAge()+"
年龄为：" + stu.getName()+" 的学生说：  ");
        stu.study();
        stu.eat();
        stu.sleep();

        System.out.print("工号："+ worker.getWnum() +"  姓名为："+
worker.getAge()+"  年龄为：" + worker.getName()+" 的工人说：  ");
        worker.work();
        worker.eat();
        worker.sleep();
    }
}
```

10. 编写测试类 TestDemo，通过多态创建猫，通过多态创建狗，并调用猫对象的 play 方法，狗对象的 play 方法

要求如下：
首先定义一个抽象类 Animal，抽象类中定义一个抽象方法 play()抽象方法
创建 一个猫 Cat 类，继承 Animal 并重写 play()方法输出“我是猫，我玩老鼠”
创建一个狗 Dog 类， 继承 Animal 并重写 play()方法输出“我是狗，我玩球”
最后编写测试类 TestDemo，通过多态创建猫，通过多态创建狗，并调用猫对象的 play 方，狗对象的 play 方法

```
abstract class Animal2{
    abstract void play();
}
class Cat2 extends Animal2{
    @Override
    public void play(){
        System.out.println("我是猫，我玩老鼠");
    }
}
class Dog2 extends Animal2{
    @Override
    public void play(){
        System.out.println("我是狗，我玩球");
    }
}
public class P4_10 {
    public static void main(String[] args) {
        Animal2 dog = new Dog2();
        dog.play();
        Animal2 cat = new Cat2();
        cat.play();
    }
}
```

11. 创建跑车 car2，用 set 方法赋值属性，输出属性，调用 run()、price()方法

要求如下：
创建一个 Car 抽象类
要求有 brand（品牌）属性，并且要求封装私有化，写出属性的 set、get 方法，抽象类 Car 构造方法中也可以对 brand 属性赋值，写出一个抽象方法 run()。
创建一个跑车类 SportsCar 继承抽象类 Car
实现 Car 抽象方法输出一条语句“超级跑车”，在本类中写出一个自己的方法 price()，输出一条语句“售价 100w”
在测试类 Test 类中创建跑车对象 car1，用构造器赋值品牌属性，输出属性，调用 run()、price()方法；创建跑车 car2，用 set 方法赋值属性，输出属性，调用 run()、price()方法

```java
abstract class Car {
    private String brand;
    public Car() {}
    public Car(String brand) {
        this.brand = brand;
    }
    abstract void run();
    public String getBrand() {
        return brand;
    }
    public void setBrand(String brand) {
        this.brand = brand;
    }

}
// 跑车类
class SportsCar extends Car {
    void price(){
        System.out.println("售价 100w");
    }
    public SportsCar(String brand) {
        super(brand);
    }
    public SportsCar() {
        super();
    }
    @Override
    void run() {
        System.out.println("超级跑车");
    }
}
public class P4_11 {
    public static void main(String[] args) {
        SportsCar car1 = new SportsCar("法拉利");
        System.out.println(car1.getBrand());
        car1.run();
        car1.price();

        SportsCar car2 = new SportsCar();
        car2.setBrand("兰博基尼");
        System.out.println(car2.getBrand());
        car1.run();
        car1.price();
```

```
    }
}
```

12.在主方法中实例化 Computer 类，调用 plugin 方法，分别传递 Flash 和 Print 的对象，通过向上转型，完成功能
完成那个 USB 接口的例子，分别定义 USB 接口，两个方法 start，stop
两个子类：Flash 和 Print，重写两个方法，方法中随便输出两句话
定义计算机类 Computer，有一个 plugin 方法，有一个 USB 类型的参数，用来调用 start 和 stop
在主方法中实例化 Computer 类，调用 plugin 方法，分别传递 Flash 和 Print 的对象，通过向上转型，完成功能

```
interface USB {
    public abstract void start();
    public abstract void stop();
}
class Flash implements USB{
    @Override
    public void start() {
        System.out.println("Flash--strat");
    }
    @Override
    public void stop() {
        System.out.println("Flash--stop");
    }
}
class Print implements USB {
    @Override
    public void start() {
        System.out.println("Print--start");
    }
    @Override
    public void stop() {
        System.out.println("Print--stop");
    }
}
class Computer {
    public void plugin(USB b){
        b.start();
        b.stop();
    }
    public  Computer(){}
}
public class P4_12 {
```

```
    public static void main(String[] args) {
        Computer c = new Computer();
        Flash f = new Flash();
        Print p = new Print();
        c.plugin(f);
        c.plugin(p);
    }
}
```

13. 根据以下要求，完成编程设计
(1) 定义一个 Flower 花作为父类
属性：颜色 价格
属性要进行封装（即为每条私有属性写 set，get 方法）
定义无参构造方法，和传两个参数的有参构造方法一个。
方法：显示详细信息的方法 showInfo

(2) 定义一个 Rose 玫瑰花类继承 Flower 类
玫瑰花自己的属性：产地（要求对产地属性进行封装 私有化属性）
重写父类中的显示详细信息的方法 showInfo，在重写的 showInfo 方法中通过 super 调用父类的 showInfo 方法；并显示产地信息。
再定义一个方法 warn 警告显示：不要摘玫瑰花，小心扎手！

(3) 定义测试类 Test 完成：
①实例化 Flower 类，调用构造方法赋值，并调用方法，输出：
花的颜色是白色，价格是 10 元
②实例化玫瑰花类，调用方法，输出：
花的颜色是紫色，价格是 30 元
产地是大理
不要摘玫瑰花，小心扎手！

```
class Flower {
    private String color;
    private String price;
    public Flower() {}
    public Flower(String color, String price) {
        this.color = color;
        this.price = price;
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
```

```
    }
    public String getPrice() {
        return price;
    }
    public void setPrice(String price) {
        this.price = price;
    }
    public void showInfo() {
        System.out.println("花的颜色是" + this.color + ",价格是" + this.price +
"元");
    }
}
class Rose extends Flower {
    private String address;
    public Rose() {}
    public Rose(String color, String price, String address) {
        super(color, price);
        this.address = address;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public void showInfo() {
        super.showInfo();
        System.out.println("产地是" + this.address);
    }

    public void warn() {
        System.out.println("不要摘玫瑰花，小心扎手！");
    }
}
public class P4_13 {
    public static void main(String[] args) {
        Flower f = new Flower("白色", "10");
        f.showInfo();
        Rose r = new Rose("紫色", "30", "大理");
        r.showInfo();
        r.warn();
    }
}
```

14. 第十题

(1) 首先定义一个程序员的接口 Programmer,接口中定义一个抽象方法 ACode（）

(2) 创建一个 Student 类，实现接口 Programmer 并实现 ACode（）方法，ACode（）方法可输出输出“程序员在敲代码”， Student 类有年龄，姓名，班级，性别私有属性（封装），各个属性的 set，get 方法，写空参构造方法，和一个有参构造方法包含这四个属性

(3) 定义测试类，在 main 方法中创建一个 Student 对象，将自己的年龄，姓名，班级，性别，赋值给这个对象，并在控制台打印出对象的各个属性。调用所有的方法

```
interface Programmer {
    public abstract void ACode();
}
class Student1 implements Programmer {
    private int age;
    private String name;
    private String clas;
    private String sex;
    public void ACode() {
        System.out.println("程序员在敲代码");
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getClas() {
        return clas;
    }
    public void setClas(String clas) {
        this.clas = clas;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public Student1() {
    }
```

```
    public Student1(int age, String name, String clas, String sex) {
        this.age = age;
        this.name = name;
        this.clas = clas;
        this.sex = sex;
    }
}
public class P4_14 {
    public static void main(String[] args) {
        Student1 s = new Student1(19, "菅田将晖", "三年级 A 班", "男");
        s.ACode();
        System.out.println("姓名：" + s.getName() + "，性别" + s.getSex() + "，
年龄：" + s.getAge() + "，班级：" + s.getClas());
    }
}
```