46.java运行时内存分为“线程共享”和“线程私有”两部分，以下哪些属于“线程共享”部分

A.程序计算器

B.方法区

C.java虚拟机栈

D.java堆

答案：B D

解析：

> 共享的资源有：
>
> - 堆 由于堆是在进程空间中开辟出来的，所以它是理所当然地被共享的；因此new出来的都是共享的（16位平台上分全局堆和局部堆，局部堆是独享的）
> - 全局变量 它是与具体某一函数无关的，所以也与特定线程无关；因此也是共享的
> - 静态变量 虽然对于局部变量来说，它在代码中是“放”在某一函数中的，但是其存放位置和全局变量一样，存于堆中开辟的.bss和.data段，是共享的
> - 文件等公用资源 这个是共享的，使用这些公共资源的线程必须同步。Win32 提供了几种同步资源的方式，包括信号、临界区、事件和互斥体。 独享的资源有
> - 栈 栈是独享的
> - 寄存器 这个可能会误解，因为电脑的寄存器是物理的，每个线程去取值难道不一样吗？其实线程里存放的是副本，包括程序计数器PC

47.下面哪些描述是正确的：（）

```
public class Test {
public static class A {
private B ref;
public void setB(B b) {
ref = b;
}
}
public static Class B {
private A ref;
public void setA(A a) {
ref = a;
}
}
public static void main(String args[]) {
…
start();
```

```
….
}
public static void start() {
    A a = new A();
    B b = new B();
    a.setB(b);
    b = null; //
    a = null;
…
}
}
```

A.b = null执行后b可以被垃圾回收

B.a = null执行后b可以被垃圾回收

C.a = null执行后a可以被垃圾回收

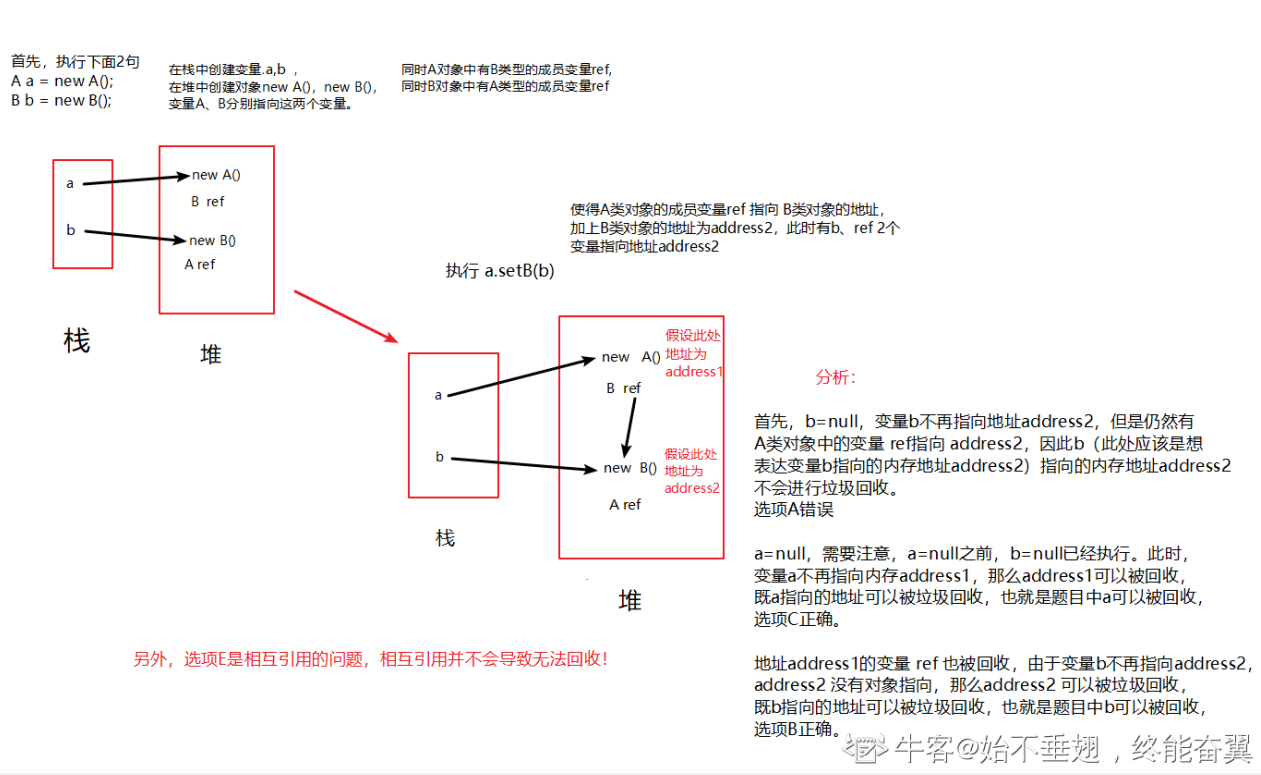
D.a,b必须在整个程序结束后才能被垃圾回收

E.类A和类B在设计上有循环引用，会导致内存泄露

F.a, b 必须在start方法执行完毕才能被垃圾回收

答案：B C

解析：



- 首先在栈中 创建A,B两个对象，假设A对象所在地址为address1,B对象地址为address2
- 执行new A操作后会有一个指向address1的地址

- 执行new B之后也会有一个指向address2的地址
- 接下来执行a.setB(b) 此时会有b，以及ref两个变量指向address2
- 执行b=null操作那么b就不在会指向address2，但是ref仍然指向address2，所以此时还无法回收b
- 执行a=null操作，那么所有的变量都不在指向address2，且所有的变量也都不在指向address1，所以a,b所指向的地址均可被回收

48.局部内部类可以用哪些修饰符修饰?

A.public

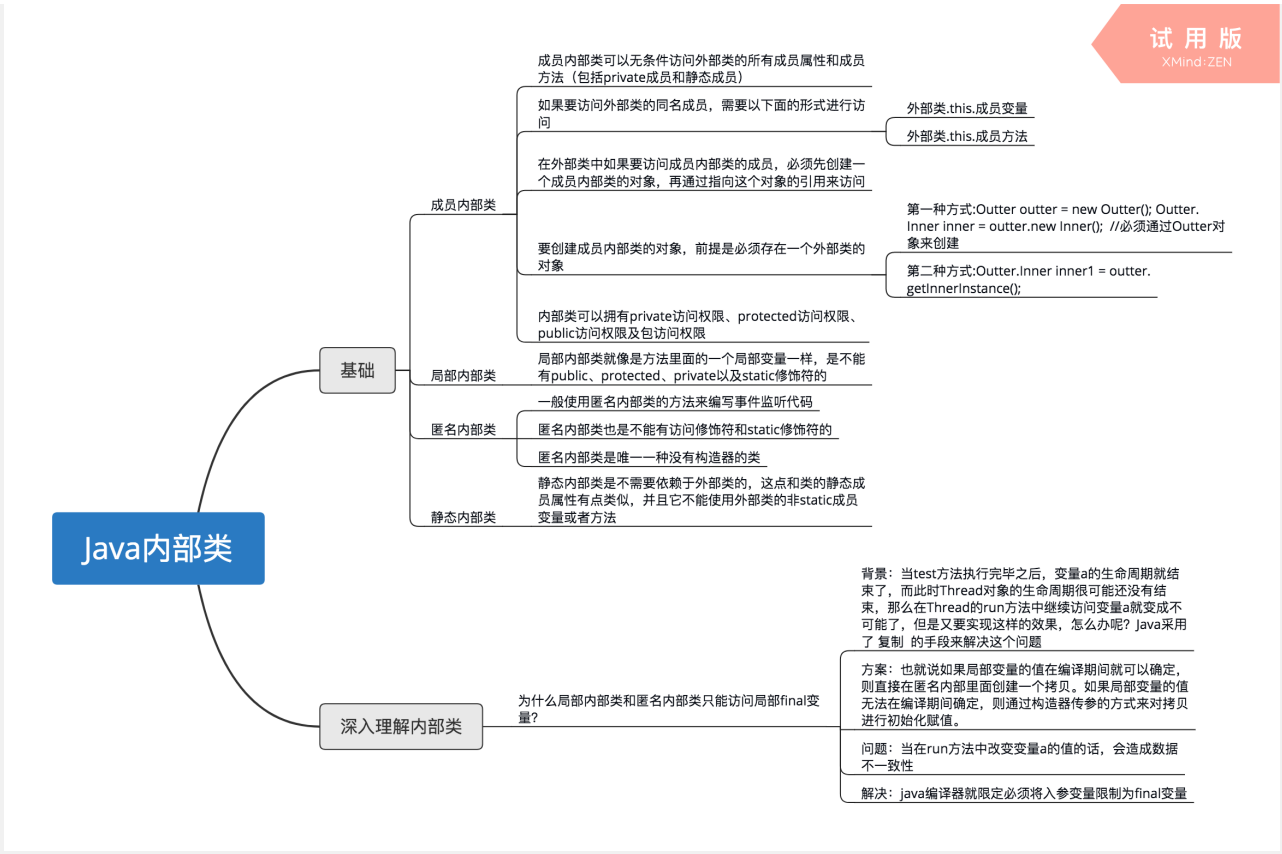
B.private

C.abstract

D.final

答案：C D

解析：



局部内部类是放在代码块或方法中的，不能有访问控制修饰符，且不能用static修饰

49.Java.Thread的方法resume()负责重新开始被以下哪个方法中断的线程的执行（）。

A.stop

B.sleep

C.wait

D.suspend

答案：D

解析：

> 线程的五大状态及其转换
>
> - resume与suspended一起使用 wait与notify(notifyAll)一起使用 sleep会让线程暂时不执行 suspend() 和 resume() 方法：两个方法配套使用，suspend()使得线程进入阻塞状态，并且不会自动恢复，必须其对应的 resume() 被调用，才能使得线程重新进入可执行状态。
> - 线程从创建、运行到结束总是处于下面五个状态之一：新建状态、就绪状态、运行状态、阻塞状态及死亡状态。
> - 1.新建状态(New)： 当用new操作符创建一个线程时， 例如new Thread(r)，线程还没有开始运行，此时线程处在新建状态。 当一个线程处于新生状态时，程序还没有开始运行线程中的代码
> - 2.就绪状态(Runnable)
> - 一个新创建的线程并不自动开始运行，要执行线程，必须调用线程的start()方法。 当线程对象调用start()方法即启动了线程，start()方法创建线程运行的系统资源， 并调度线程运行run()方法。当start()方法返回后，线程就处于就绪状态。 处于就绪状态的线程并不一定立即运行run()方法，线程还必须同其他线程竞争CPU 时间，只有获得CPU时间才可以运行线程。因为在单CPU的计算机系统中，不可能 同时运行多个线程，一个时刻仅有一个线程处于运行状态。因此此时可能有多个线程处于就绪状态。对多个处于就绪状态的线程是由Java运行时系统的线程调度 程序(thread scheduler)来调度的。
> - 3.运行状态(Running) 当线程获得CPU时间后，它才进入运行状态，真正开始执行run()方法.
> - 4.阻塞状态(Blocked) 线程运行过程中，可能由于各种原因进入阻塞状态: 1>线程通过调用sleep方法进入睡眠状态； 2>线程调用一个在I/O上被阻塞的操作，即该操作在输入输出操作完成之前不会返回到它的调用者； 3>线程试图得到一个锁，而该锁正被其他线程持有； 4>线程在等待某个触发条件； ..... 所谓阻塞状态是正在运行的线程没有运行结束，暂时让出CPU，这时其他处于就绪状态的线程就可以获得CPU时间， 进入运行状态。
> - 5.死亡状态(Dead) 有两个原因会导致线程死亡： run方法正常退出而自然死亡， 一个未捕获的异常终止了run方法而使线程猝死。 为了确定线程在当前是否存活着（就是要么是可运行的，要么是被阻塞了），需要使用isAlive方法。如果是 可运行或被阻塞，这个方法返回true； 如果线程仍旧是new状态且不是可运行的， 或者线程死亡了， 则返回false.

50.以下程序的运行结果是？

```
public class TestThread {
    public static void main(String args[]) {
        Runnable runner = new Runnable() {
            @Override
            public void run() {
                System.out.print("foo");
            }
        };
        Thread t = new Thread(runner);
        t.run();
        System.out.print("bar");
    }
}
```

A.foobar

B.barfoo

C.foobar或者barfoo都有可能

D.Bar

E.Foo

F.程序无法正常运行

答案：A

解析：

> - 线程的启动方式只能通过start这种方式启动才能真正的实现多线程的效果
> - 如果是手动调用run方法和普通方法调用没有区别，所以这个还是按照顺序执行首先执行run方法之后，执行输出语句所以最终得到结果foobar.
> - 调用start（）后，线程会被放到等待队列，等待CPU调度，并不一定要马上开始执行，只是将这个线程置于可动行状态。然后通过JVM，线程Thread会调用run（）方法，执行本线程的线程体。

51.java8中，下面哪个类用到了解决哈希冲突的开放定址法

A.LinkedHashSet

B.HashMap

C.ThreadLocal

D.TreeMap

答案：C

解析：

> - ThreadLocalMap中使用开放地址法来处理散列冲突
> - HashMap中使用的是分离链表法 之所以采用不同的方式主要是因为：在ThreadLocalMap中的散列值分散得十分均匀，很少会出现冲突。并且ThreadLocalMap经常需要清除无用的对象，使用纯数组更加方便。

52.关于抽象类与接口，下列说法正确的有？

A.优先选用接口，尽量少用抽象类

B.抽象类可以被声明使用，接口不可以被声明使用

C.抽象类和接口都不能被实例化。

D.以上说法都不对

答案：AC

解析：

抽象类:

> - 含有abstract修饰符的class即为抽象类,abstract类不能创建的实例对象。
> - 含有abstract方法的类必须定义为abstract class，abstract class类中的方法不必是抽象的。abstract class
> - 类中定义抽象方法必须在具体(Concrete)子类中实现,所以，不能有抽象构造方法或抽象静态方法,如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为abstract类型。

接口:

> 接口中的所有方法都必须是抽象的。接口中的方法定义默认为public abstract类型，接口中的成员变量类型默认为public static final。

两者之间的联系和区别：

抽象类	接口
可以有构造方法	不能有构造方法
可以有普通成员变量	没有普通成员变量
可以包含非抽象的普通方法	所有方法必须都是抽象的，不能有非抽象的普通方法
访问类型可以是public，protected	只能是public类型的，并且默认即为public abstract类型

抽象类	接口
可以包含静态方法	不能包含静态方法
可以包含静态成员变量,静态成员变量的访问类型可以任意	可以包含静态成员变量,但接口中定义的变量只能是public static final类型，并且默认即为public static final类型

53.下面有关Java的说法正确的是（ ）

A.一个类可以实现多个接口

B.抽象类必须有抽象方法

C.protected成员在子类可见性可以修改

D.通过super可以调用父类构造函数

E.final的成员方法实现中只能读取类的成员变量

F.String是不可修改的，且java运行环境中对string对象有一个对象池保存

答案:A C D F

解析:

这里解释一下E和F选项 E:

> final 的成员方法除了能读取类的成员变量，还能读取类变量

F:

> String 声明的是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象

54.下列说法错误的有（ ）

A.在类方法中可用this来调用本类的类方法

B.在类方法中调用本类的类方法时可直接调用

C.在类方法中只能调用本类中的类方法

D.在类方法中绝对不能调用实例方法

答案： A C D

解析:

成员方法

> 又称为实例方法，非静态成员函数, 暗含this指针

静态方法

> 又称为类方法,静态成员函数,由static修饰，与类对象无关,缺少this指针

- this是对象，及有实例 故A错；
- 静态成员函数可以直接引用该类的静态成员变量和函数，但不允许直接引用非静态成员变量（若要引用，则需 通过传递参数的方式得到对象名）故B对
- 可以通过继承，来调用父类的 类方法。故C错；
- 可以生成实例，然后通过this来调用实例方法，故D对

55.以下哪些jvm的垃圾回收方式采用的是复制算法回收()

A.新生代串行收集器

B.老年代串行收集器

C.并行收集器

D.新生代并行回收收集器

E.老年代并行回收收集器

F.cms收集器

答案：A D

解析:

两个最基本的java回收算法：复制算法和标记清理算法 复制算法:

> 两个区域A和B，初始对象在A，继续存活的对象被转移到B。此为新生代最常用的算法

标记清理算法：

> 一块区域，标记可达对象（可达性分析），然后回收不可达对象，会出现碎片，那么引出标记-整理算法：多了碎片整理，整理出更大的内存放更大的对象

两个概念：新生代和年老代

- 新生代：初始对象，生命周期短的
- 永久代：长时间存在的对象 整个java的垃圾回收是新生代和年老代的协作，这种叫做分代回收。
- Serial New收集器是针对新生代的收集器，采用的是复制算法
- Parallel New（并行）收集器，新生代采用复制算法，老年代采用标记整理
- Parallel Scavenge（并行）收集器，针对新生代，采用复制收集算法
- Serial Old（串行）收集器，新生代采用复制，老年代采用标记整理
- Parallel Old（并行）收集器，针对老年代，标记整理

- CMS收集器，基于标记清理
- G1收集器：整体上是基于标记 整理 ， 局部采用复制

56.以下定义一维数组的语句中，正确的是：（）

A.int a [10]

B.int a []=new [10]

C.int a[] int a []=new int [10]

D.int a []={1,2,3,4,5}

答案：D

解析:

Java一维数组有两种初始化方法:

- 1、静态初始化

```
int array[] = new int[]{1,2,3,4,5}
```

或者

```
int array[] = {1,2,3,4,5}
```

注意：写成一下形势也是错误的

```
int array[] = new int[5]{1,2,3,4,5}
```

- 2、动态初始化

```
array[0] = 1;
array[1] = 2;
array[2] = 3;
array[3] = 4;
array[4] = 5;
```

静态与动态初始化的区别就在于，前者是声明的时候就初始化，后者是先声明，再动态初始化。

57.有关线程的叙述正确的是()

A.可以获得对任何对象的互斥锁定。

B.通过继承Thread类或实现Runnable接口，可以获得对类中方法的互斥锁定。

C.线程通过使用synchronized关键字可获得对象的互斥锁定。

D.线程的创建只能通过继承Thread类来实现。

答案：C

解析:

- 互斥锁指的是只有一个线程可以访问该对象。
- 如果变量用volatile修饰，则该变量是线程共享的，无法获得该变量的互斥锁
- 采用synchronized修饰符实现的同步机制叫做互斥锁机制，它所获得的锁叫做互斥锁。每个对象都有一个monitor(锁标记)，当线程拥有这个锁标记时才能访问这个资源，没有锁标记便进入锁池。任何一个对象系统都会为其创建一个互斥锁，这个锁是为了分配给线程的，防止打断原子操作。每个对象的锁只能分配给一个线程，因此叫做互斥锁。

58.Java的Daemon线程，setDaemon( )设置必须要?

A.在start之前

B.在start之后

C.前后都可以

答案：A

解析:

java线程是一个运用很广泛的重点知识,我们很有必要了解java的daemon线程. 首先我们必须清楚的认识到java的线程分为两类: 用户线程和daemon线程

- 2：守护线程。 守护线程是服务用户线程的线程，在它启动之前必须先set。
- 1.用户线程 **通过Thread.setDaemon(false)设置为用户线程；** 用户线程可以简单的理解为用户定义的线程,当然包括main线程
- 2.daemon线程 **通过Thread.setDaemon(true)设置为守护线程，如果不设置，默认用户线程** daemon线程是为我们创建的用户线程提供服务的线程,比如说jvm的GC等等,这样的线程有一个非常明显的特征: 当用户线程运行结束的时候,daemon线程将会自动退出.(由此我们可以推出下面关于daemon线程的几条基本特点):
    - 守护线程创建的过程中需要先调用setDaemon方法进行设置,然后再启动线程.否则会报出IllegalThreadStateException异常.
    - 由于daemon线程的终止条件是当前是否存在用户线程,所以我们不能指派daemon线程来进行一些业务操作,而只能服务用户线程.
    - daemon线程创建的子线程任然是daemon线程. **守护线程是服务用户线程的线程，在它启动之前必须先set。**

59.下列说法正确的是

A.在类方法中可用this来调用本类的类方法

B.在类方法中调用本类的类方法可直接调用

C.在类方法中只能调用本类的类方法

D.在类方法中绝对不能调用实例方法

答案： B

解析:

- 在类方法中调用本类的类方法可直接调用。
- 实例方法也叫做对象方法。类方法是属于整个类的，而实例方法是属于类的某个对象的。
- 由于类方法是属于整个类的，并不属于类的哪个对象，所以类方法的方法体中不能有与类的对象有关的内容。即类方法体有如下限制：
  - (1) 类方法中不能引用对象变量；
  - (2) 类方法中不能调用类的对象方法；
  - (3) 在类方法中不能使用super、this关键字。
  - (4)类方法不能被覆盖。 如果违反这些限制，就会导致程序编译错误。
- 与类方法相比，对象方法几乎没有什么限制：
  - (1) 对象方法中可以引用对象变量，也可以引用类变量；
  - (2) 对象方法中可以调用类方法；
  - (3) 对象方法中可以使用super、this关键字。

60.What is displayed when the following is executed;

```
double d1=-0.5;
System.out.println("Ceil d1="+Math.ceil(d1));
System.out.println("floor d1="+Math.floor(d1));
```

A.

Ceil d1=-0.0 floor d1=-1.0

B.

Ceil d1=0.0 floor d1=-1.0

C.

Ceil d1=-0.0 floor d1=-0.0

D.

Ceil d1=0.0 floor d1=0.0

E.

Ceil d1=0 floor d1=-1

答案：A

解析:

- ceil：天花板数，向上取整。
- floor：地板数，向下取整 ceil 和 floor 方法 上都有一句话：If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument，意思为：如果参数是 NaN、无穷、正 0、负 0，那么结果与参数相同， 如果是 -0.0，那么其结果是 -0.0