1.对接口的描述正确的是()

A.一个类可以实现多个接口

B.接口可以有非静态的成员变量

C.在jdk8之前，接口可以实现方法

D.实现接口的任何类，都需要实现接口的方法

> 解析： 答案：A A，一个类只能有一个直接父类，但是继承是有传递性的。一个类可以实现多的接口。一个接口可以继承多个类。 B，接口中没有普通变量（普通成员变量），接口中都是常量，默认修饰符：public static final C，JDK8之前，接口中的方法都是默认public abstract的，JDK8之后，接口中可以有static、default的修饰的方法，一旦被修饰，方法必须有方法体（抽象方法是可以没有方法体的），接口中的方法都不能被private和protected修饰，同时外部接口、类只能被public修饰或者不写，但是内部接口、类可以被四个访问修饰符修饰。 D， 实现接口，其实就是需要重写接口中的abstract方法，一旦实现的类没有重写完，那么这个类必须是个抽象类（抽象类中可以没有抽象方法，但是有抽象方法的类必须是抽象类）。

2.在 applet 的方法中，可关闭小应用程序并释放其占用资源的是（ ）

A.stop()

B.paint()

C.init()

D.destroy()

> 解析： 答案：D Applet 类是浏览器类库中最为重要的类，同时也是所有 JAVA 小应用程序的基本类。 一个 Applet 应用程序从开始运行到结束时所经历的过程被称为 Applet 的生命周期。 Applet 的生命周期涉及 init() 、 start() 、 stop() 和 destroy() 四种方法，这 4 种方法都是 Applet 类的成员，可以继承这些方法，也可以重写这些方法，覆盖原来定义的这些方法。除此之外，为了在 Applet 程序中实现输出功能，每个 Applet 程序中还需要重载 paint() 方法： 1、 public void init() init()方法是 Applet 运行的起点。当启动 Applet 程序时，系统首先调用此方法，以执行初始化任务。 2、 public void start() start()方法是表明 Applet 程序开始执行的方法。当含有此 Applet 程序的 Web 页被再次访问时调用此方法。因此，如果每次访问 Web 页都需要执行一些操作的话，就需要在 Applet 程序中重载该方法。在 Applet 程序中，系统总是先调用 init() 方法，后调用 start() 方法。 3、 public void stop() stop()方法使 Applet 停止执行，当含有该 Applet 的 Web 页被其他页代替时也要调用该方法。 4、 public void destroy() destroy()方法收回 Applet 程序的所有资源，即释放已分配给它的所有资源。在 Applet 程序中，系统总是先调用 stop() 方法，后调用 destroy() 方法。 5、 paint(Graphics g) paint(Graphics g)方法可以使 Applet 程序在屏幕上显示某些信息，如文字、色彩、背景或图像等。参数 g 是 Graphics 类的一个对象实例，实际上可以把 g 理解为一个画笔。对象 g 中包含了许多绘制方法，如 drawstring() 方法就是输出字符串。

3.java语言的下面几种数组复制方法中，哪个效率最高?

A.for 循环逐一复制

B.System.arraycopy

C.Array.copyOf

D.使用clone方法

> 解析： 答案：B 复制的效率System.arraycopy>clone>Arrays.copyOf>for循环 这里面在System类源码中给出了arraycopy的方法，是native方法，也就是本地方法，肯定是最快的。而Arrays.copyOf(注意是Arrays类，不是Array)的实现，在源码中是调用System.copyOf的，多了一个步骤，肯定就不是最快的

4.有关会话跟踪技术描述正确的是（）

A.Cookie是Web服务器发送给客户端的一小段信息，客户端请求时，可以读取该信息发送到服务器端

B.关闭浏览器意味着临时会话ID丢失，但所有与原会话关联的会话数据仍保留在服务器上，直至会话过期

C.在禁用Cookie时可以使用URL重写技术跟踪会话

D.隐藏表单域将字段添加到HTML表单并在客户端浏览器中显示

> 解析： 答案：A B C 1.session用来表示用户会话，session对象在服务端维护，一般tomcat设定session生命周期为30分钟，超时将失效，也可以主动设置无效； 2.cookie存放在客户端，可以分为内存cookie和磁盘cookie。内存cookie在浏览器关闭后消失，磁盘cookie超时后消失。当浏览器发送请求时，将自动发送对应cookie信息，前提是请求url满足cookie路径； 3.可以将sessionId存放在cookie中，也可以通过重写url将sessionId拼接在url。因此可以查看浏览器cookie或地址栏url看到sessionId； 4.请求到服务端时，将根据请求中的sessionId查找session，如果可以获取到则返回，否则返回null或者返回新构建的session，老的session依旧存在 5.隐藏域在页面中对于用户是不可见的，在表单中插入隐藏域的目的在于收集或发送信息，以利于被处理表单的程序所使用。浏览者单击发送按钮发送表单的时候，隐藏域的信息也被一起发送到服务器。

5.在使用super和this关键字时，以下描述错误的是（）

A.在子类构造方法中使用super()显示调用父类的构造方法，super()必须写在子类构造方法的第一行，否则编译不通过

B.super()和this()不一定要放在构造方法内第一行

C.this()和super()可以同时出现在一个构造函数中

D.this()和super()可以在static环境中使用，包括static方法和static语句块

> 解析： 答案：B C D 1.super和this都只能位于构造器的第一行，而且不能同时使用，这是因为会造成初始化两次，this用于调用重载的构造器，super用于调用父类被子类重写的方法，由于super()和this()必须在构造函数第一行，所以这一点也表明他俩不能在一个构造函数中 2.super()表示调用父类构造函数、this()调用自己的构造函数，而自己的构造函数第一行要使用super()调用父类的构造函数，所以这俩不能在一个构造函数中会出现重复引用的情况 3.this()和super()都指的是对象，所以，均不可以在static环境中使用。包括：static变量,static方法，static语句块(里面不能使用非static类型的)。因为static修饰的方法不能存在this指针

6.下面有关forward和redirect的描述，正确的是()？

A.forward是服务器将控制权转交给另外一个内部服务器对象，由新的对象来全权负责响应用户的请求

B.执行forward时，浏览器不知道服务器发送的内容是从何处来，浏览器地址栏中还是原来的地址

C.执行redirect时，服务器端告诉浏览器重新去请求地址

D.forward是内部重定向，redirect是外部重定向

E.redirect默认将产生301 Permanently moved的HTTP响应

> 解析： 答案：B C D 1.从地址栏显示来说 forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址. redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL. 2.从数据共享来说 forward:转发页面和转发到的页面可以共享request里面的数据. redirect:不能共享数据. 3.从运用地方来说 forward:一般用于用户登陆的时候,根据角色转发到相应的模块. redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等. 4.从效率来说 forward:高. redirect:低.

7.下面有关forward和redirect的描述，正确的是()？

A.forward是服务器将控制权转交给另外一个内部服务器对象，由新的对象来全权负责响应用户的请求

B.执行forward时，浏览器不知道服务器发送的内容是从何处来，浏览器地址栏中还是原来的地址

C.执行redirect时，服务器端告诉浏览器重新去请求地址

D.forward是内部重定向，redirect是外部重定向

E.redirect默认将产生301 Permanently moved的HTTP响应

> 解析： 答案：B C D 1.从地址栏显示来说 forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址. redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL. 2.从数据共享来说 forward:转发页面和转发到的页面可以共享

> request里面的数据. redirect:不能共享数据. 3.从运用地方来说 forward:一般用于用户登陆的时候,根据角色转发到相应的模块. redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等. 4.从效率来说 forward:高. redirect:低.

8.执行下列代码的输出结果是( )

```
public class Demo{
  public static void main(String args[]){
      int num = 10;
      System.out.println(test(num));
}
public static int test(int b){
      try
      {
        b += 10;
        return b;
      }
      catch(RuntimeException e)
      {
      }
      catch(Exception e2)
      {
      }
      finally
      {
        b += 10;
        return b;
      }
    }
}
```

A.10

B.20

C.30

D.40

> 解析： 答案：C 如果finally块中有return语句的话，它将覆盖掉函数中其他return语句。

9.若有定义语句： int a=10 ; double b=3.14 ; 则表达式 'A'+a+b 值的类型是（）

A.char

B.int

C.double

D.float

> 解析： 答案：C 不同类型运算结果类型向右边靠齐。 char < short < int < float < double

10.下面论述正确的是（）？

A.如果两个对象的hashcode相同，那么它们作为同一个HashMap的key时，必然返回同样的值

B.如果a,b的hashcode相同，那么a.equals(b)必须返回true

C.对于一个类，其所有对象的hashcode必须不同

D.如果a.equals(b)返回true，那么a,b两个对象的hashcode必须相同

> 解析： 答案：D hashCode()方法和equals()方法的作用其实是一样的，在Java里都是用来对比两个对象是否相等一致。 那么equals()既然已经能实现对比的功能了，为什么还要hashCode()呢？因为重写的equals()里一般比较的比较全面比较复杂，这样效率就比较低，而利用hashCode()进行对比，则只要生成一个hash值进行比较就可以了，效率很高。 那么hashCode()既然效率这么高为什么还要equals()呢？ 因为hashCode()并不是完全可靠，有时候不同的对象他们生成的hashcode也会一样（生成hash值得公式可能存在的问题），所以hashCode()只能说是大部分时候可靠，并不是绝对可靠， 所以我们可以得出：
> 1.equals()相等的两个对象他们的hashCode()肯定相等，也就是用equals()对比是绝对可靠的。
> 2.hashCode()相等的两个对象他们的equal()不一定相等，也就是hashCode()不是绝对可靠的。 所有对于需要大量并且快速的对比的话如果都用equals()去做显然效率太低，所以解决方式是，每当需要对比的时候，首先用hashCode()去对比，如果hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用equal()去再对比了）,如果hashCode()相同，此时再对比他们的equals()，如果equals()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！

11.下面程序的输出结果为（）

```
public class Demo {
 public static String sRet = "";
 public static void func(int i)
 {
 try
 {
 if (i%2==0)
 {
 throw new Exception();
 }
 }
 catch (Exception e)
 {
 sRet += "0";
 return;
 }
 finally
```

```
{
sRet += "1";
}
sRet += "2";
}
public static void main(String[] args)
{
func(1);
func(2);
System.out.println(sRet);
}
}
```

A.120

B.1201

C.12012

D.101

> 解析： 答案：B
>
> - 调用func(1),if不符合，直接进入finally，sRet=“1"
> - finally语句中没有返回值，故继续向下执行，sRet="12"
> - 调用func(2),if符合，sRet="120"，此时有返回值!!!
> - 调用finally语句，sRet="1201"
> - 因为已经有返回值了，finally之后的语句也不再执行，sRet="1201"。

12.在java7中，下列不能做switch()的参数类型是?

A.int型

B.枚举类型

C.字符串

D.浮点型

> 解析： 答案：D switch语句后的控制表达式只能是short、char、int、String、long整数类型和枚举类型，不能是float，double和boolean类型。String类型是java7开始支持。

13.以下代码可以使用的修饰符是： （）

A.final

B.static

C.abstract

D.public

> 解析： 答案：C
>
> - 接口中字段的修饰符：public static final（默认不写）
> - 接口中方法的修饰符：public abstract（默认不写） abstract只能修饰类和方法 不能修饰字段

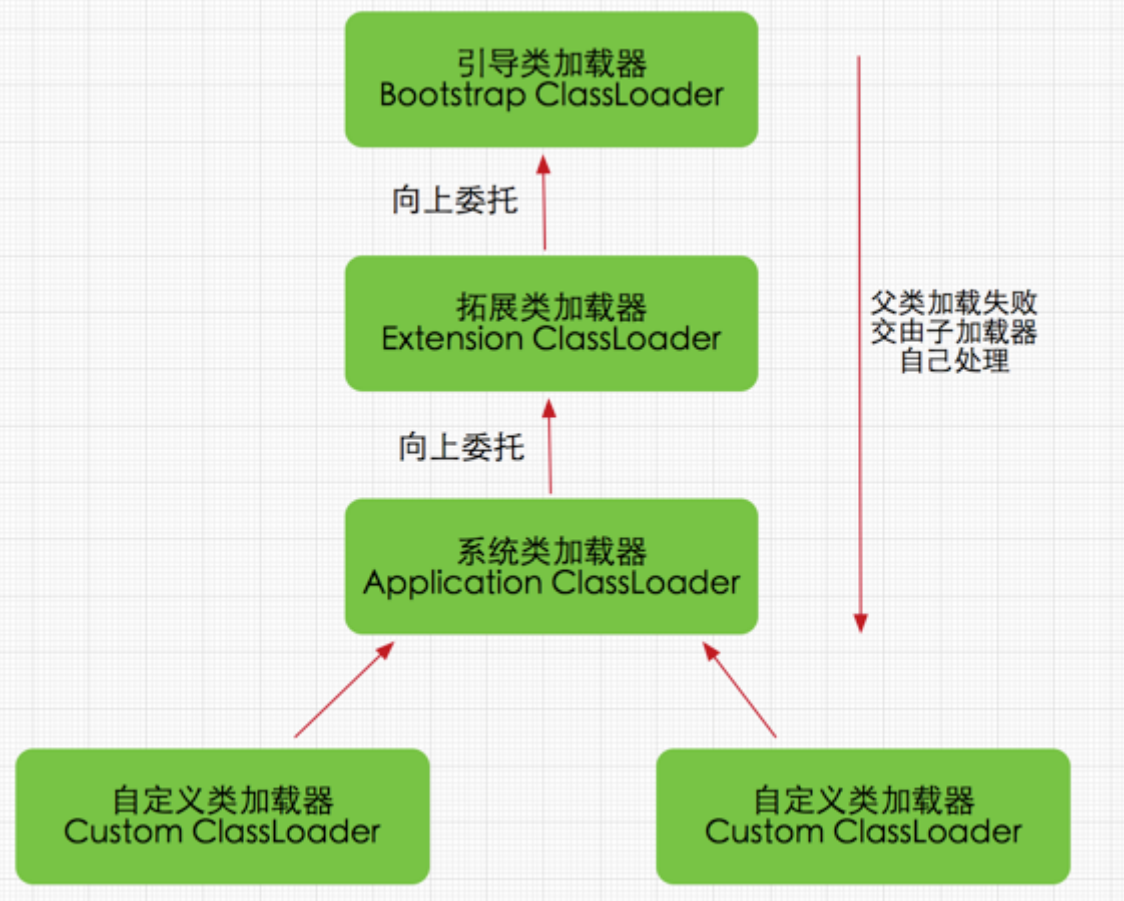
14.下面有关java classloader说法错误的是?

A.Java默认提供的三个ClassLoader是BootStrap ClassLoader，Extension ClassLoader，App ClassLoader

B.ClassLoader使用的是双亲委托模型来搜索类的

C.JVM在判定两个class是否相同时，只用判断类名相同即可，和类加载器无关

D.ClassLoader就是用来动态加载class文件到内存当中用的

> 解析： 答案：C
>
> 
>
>
> 一个jvm中默认的classloader有Bootstrap ClassLoader、Extension ClassLoader、App ClassLoader，分别各司其职：
>
> - Bootstrap ClassLoader 负责加载java基础类，主要是 %JRE_HOME/lib/ 目录下的rt.jar、resources.jar、charsets.jar和class等
> - Extension ClassLoader 负责加载java扩展类，主要是 %JRE_HOME/lib/ext 目录下的jar和class
> - App ClassLoader负责加载当前java应用的classpath中的所有类。 classloader 加载类用的是全盘负责委托机制。 所谓全盘负责，即是当一个classloader加载一个Class的时候，这个Class所依赖的和

> 引用的所有 Class也由这个classloader负责载入，除非是显式的使用另外一个classloader载入。 所以，当我们自定义的classloader加载成功了 com.company.MyClass以后，MyClass里所有依赖的class都由这个classLoader来加载完成。

> 比较两个类是否相等，只有这两个类是由同一个类加载器加载才有意义。否则，即使这两个类是来源于同一个Class文件，只要加载它们的类加载器不同，那么这两个类必定不相等。

补充：

1. 什么是类加载器?

> 把类加载的过程放到Java虚拟机外部去实现，让应用程序决定如何去获取所需要的类。实现这个动作的代码模块称为"类加载器"。

2. 有哪些类加载器，分别加载哪些类

> 类加载器按照层次，从顶层到底层，分为以下三种： (1)启动类加载器 : 它用来加载 Java 的核心库，比如String、System这些类 (2)扩展类加载器 : 它用来加载 Java 的扩展库。 (3) 应用程序类加载器 : 负责加载用户类路径上所指定的类库，一般来说，Java 应用的类都是由它来完成加载的。

3. 双亲委派模型

> 我们应用程序都是由以上三种类加载器互相配合进行加载的，还可以加入自己定义的类加载器。称为 类加载器的双亲委派模型 ，这里类加载器之间的父子关系一般不会以继承的关系来实现，而是都使用 组合关系 来复用父加载器的。

4. 双亲委托模型的工作原理

> 是当一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一个层次的类加载都是如此，因此所有的加载请求最终都应该传送到顶层的启动类加载器中，只有当父加载器反馈自己无法加载这个加载请求的时候，子加载器才会尝试自己去加载。

5. 使用双亲委派模型好处？（原因）

> 第一：可以避免重复加载，当父亲已经加载了该类的时候，子类不需要再次加载。 第二：考虑到安全因素，如果不使用这种委托模式，那我们就可以随时使用自定义的String来动态替代java核心api中定义类型，这样会存在非常大的安全隐患，而双亲委托的方式，就可以避免这种情况，因为String已经在启动时被加载，所以用户自定义类是无法加载一个自定义的类装载器。

15.以下程序执行的结果是：

```
class X{
    Y y=new Y();
    public X(){
        System.out.print("X");
    }
}
class Y{
    public Y(){
```

```
        System.out.print("Y");
    }
}
public class Z extends X{
    Y y=new Y();
    public Z(){
        System.out.print("Z");
    }
    public static void main(String[] args) {
        new Z();
    }
}
```

A.ZYXX

B.ZYXY

C.YXYZ

D.ClaXYZXssLoader就是用来动态加载class文件到内存当中用的

> 解析： 答案：C 初始化过程：
>
> - 初始化父类中的静态成员变量和静态代码块；
> - 初始化子类中的静态成员变量和静态代码块；
> - 初始化父类的普通成员变量和代码块，再执行父类的构造方法；
> - 初始化子类的普通成员变量和代码块，再执行子类的构造方法；

> （1）初始化父类的普通成员变量和代码块，执行 Y y=new Y(); 输出Y （2）再执行父类的构造方法；输出X （3） 初始化子类的普通成员变量和代码块，执行 Y y=new Y(); 输出Y （4）再执行子类的构造方法；输出Z 所以输出YXYZ