作答提示:

Test1

**\*请在 20 分钟内作答完毕 \*下面所有题目请依据 JDK8 版本作为标准.**

**\*请在你认为正确的选项框上打勾 \*作答过程中请勿使用电子设备**

# 条件语句switch(expr) 中, expr 可以是那些数据类型

□ boolean □ byte □ short □ double □ char

**□** int □ long □ String □ enum □ Character

# 下面语句写法正确的是

□ float f=3.4 + 1; float f = 3.4f +1； □ short s1 = 1; s1 = s1 + 1; s1+=s1

□ double v1 = 1.1; v1+= 1; □ int i = 0; i = i+++i

# 关于数组以下描述正确的是

□ ArrayList 的访问元素的时间复杂度是 O(1), 而 LinkedList 则是O(n)

□ ArrayList 比 LinkedList 使用更多的内存

□ 对于插入操作, 有序数组比无序数组效率更高

□ ArrayList 是线程安全的,而 LinkedList 则不是

# 执行下面的语句,输出的结果为

Map<String,String> map = new HashMap<>();

String a = "a";

map.put(a, "a");

String b = new String("a");

map.put(b,"b");

String c = new String("a").intern();

map.put(c,"c");

System.out.println(map.get(a));

System.out.println(map.get(b));

System.out.println(map.get(c));

□ c c c □ a b c □ c b c □a b a

# 以下描述正确的是

□ 并发和并行是同一个概念的不同说法(一个处理多个，多个处理多个)

□ 进程是操作系统资源分配的基本单位,多个线程可以共享进程的资源

□ 调用Thread 实例的 start 方法后,此时该线程进入 运行态(Running)

□ 调用 Thread 实例的 interrupt 方法可以立刻中断线程

# 执行下面的语句,输出的结果为

Integer a = 128;

Integer b = 128;

System.out.println(a == b);

Integer c = 1;

Integer d = 1;

System.out.println(c == d);

Integer e = new Integer(1);

Integer f = new Integer(1);

System.out.println(e == f);

□ true true true □ true true false □ false true false □ false false false

# 给定下面的类

class Hello extends Thread{

@Override

public void run() {

new Thread(new World()).run();

System.out.println("hello");

}

}

class World implements Runnable{

@Override

public void run() {

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

}

System.out.println("world");

}

}

则以下语句的输出结果为

public static void main(String[] args) {

new Hello().run();

}

□ hello world □ world hello □ 程序报错

# 关于访问修饰符下面描述正确的是

□ private : 在同一类内可见。可以修饰：变量、方法,不能修饰类（外部类）  
□ default (即缺省，什么也不写）: 在同一包内可见。可以修饰：类、接口、变量、方法。  
□ protected : 同一包内的类和所有子类可见。可以修饰：变量、方法,不能修饰类（外部类）。  
□ public : 对所有类可见。可以修饰：类、接口、变量、方法

# 执行下面的语句, c 的最终结果是

int a = 0, b = 0, c = 0;

if (a++ == 1 || b++ == 0) {

if (--a == -1 && --b == 2) {

c += a + b;

} else if (a-- > 0 || ++b > 0 && ++c == 0) {

c += a + b;

}

}

□ 0 □ 1 □ 2 □ 3

# 给定以下类

class Human{

private String a = "yes";

protected String b = "no";

protected void yes(){

System.out.println(a);

}

private void no() {

System.out.println(b);

}

public void say() {

yes();

no();

}

}

class Man extends Human{

String b = "yes";

void no() {

System.out.println(b);

}

}

class Woman extends Human{

String a = "no";

protected void yes(){

System.out.println(a);

}

}

则下面语句的输出结果为

public static void main(String[] args) {

new Man().say();

new Woman().say();

}

□ yes no no yes □ no yes yes no □ yes no no no □ no no yes yes

# 给定 Parent类如下

class Parent{  
 final int add(int a,int b) {  
 return a + b;  
 }  
 protected void foo(String b){  
 }  
 String fun(boolean isFun) {  
 return "true";  
 }  
 private void foo(boolean noob) {  
 }  
}

下面 Child 子类写法正确的是

//写法A

class Child extends Parent{  
 int add(int a,int b) {  
 return a + b;  
 }  
 protected void foo(String b){  
 }  
 String fun(boolean isFun) {  
 return "true";  
 }  
}

//写法B  
class Child extends Parent{  
 protected void foo(String b){  
 }  
 String fun(boolean isFun) {  
 return "true";  
 }  
 @Override  
 void foo(boolean noob) {  
 }  
}  
//写法C  
class Child extends Parent{  
 private

void foo(String b){  
 }  
 String fun(boolean isFun) {  
 return "true";  
 }  
 void foo(boolean noob) {  
 }  
}

//写法D  
class Child extends Parent{  
 protected void foo(String b){  
 }  
 boolean fun(boolean isFun) {  
 return isFun;  
 }  
 void foo(boolean noob) {  
 }  
}

//写法E  
class Child extends Parent{  
 public void foo(String b){  
 }  
 protected String fun(boolean isFun) {  
 return "true";  
 }  
 final void foo(boolean noob) {  
 }  
}

# 给定如下类

class Human{  
 {  
 System.out.println("Human static block");  
 }  
 Human(){  
 System.out.println("Human Constructor");  
 }  
 public static void foo() {  
 System.out.println("Human static Method");  
 }  
}  
class Man extends Human{  
 {  
 System.out.println("Man static block");  
 }  
 Man(){  
 System.out.println("Man Constructor");  
 }  
}

则下面语句的输出结果是

public static void main(String[] args) {  
 new Man().foo();  
}

|  |  |
| --- | --- |
| □ Human static block  Human Constructor  Man static block  Man Constructor  Human static Method | □ Human static block  Man static block  Human Constructor  Man Constructor  Human static Method |
| □ Human static block  Man static block  Human static Method  Human Constructor  Man Constructor | □ Human Constructor  Man Constructor  Human static block  Man static block  Human static Method |

# 给定以下类

class ReadJob implements Runnable {  
 ReentrantReadWriteLock lock;  
 private AtomicInteger count;  
 public ReadJob(ReentrantReadWriteLock lock, AtomicInteger count) {  
 this.lock = lock;  
 this.count = count;  
 }  
 @Override  
 public void run() {  
 ReentrantReadWriteLock.ReadLock readLock = lock.readLock();  
 readLock.lock();  
 try {  
 Thread.sleep(1000);  
 count.incrementAndGet();  
 } catch (InterruptedException e) {  
 } finally {  
 readLock.unlock();  
 }  
 }  
}  
class WriteJob implements Runnable {  
 ReentrantReadWriteLock lock;  
 private AtomicInteger count;  
 public WriteJob(ReentrantReadWriteLock lock, AtomicInteger count) {  
 this.lock = lock;  
 this.count = count;  
 }  
 @Override  
 public void run() {  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 }  
 ReentrantReadWriteLock.WriteLock writeLock = lock.writeLock();  
 writeLock.lock();  
 try {  
 count.addAndGet(count.get());  
 } finally {  
 writeLock.unlock();  
 }  
 }  
}

下面语句的输出结果是 8

public static void main(String[] args) throws InterruptedException {  
 ReentrantReadWriteLock lock = new ReentrantReadWriteLock();  
 AtomicInteger count = new AtomicInteger(0);  
 ExecutorService executorService = Executors.newFixedThreadPool(4);  
 executorService.submit(new ReadJob(lock, count));  
 executorService.submit(new WriteJob(lock, count));  
 executorService.execute(new ReadJob(lock, count));  
 executorService.execute(new WriteJob(lock, count));  
 executorService.shutdown();  
 executorService.awaitTermination(2000, TimeUnit.MILLISECONDS);  
 System.out.println(count.get());  
}

# 下面有关抽象类和接口描述正确的是

□ 抽象类可以有构造器,但是接口不能有构造器

□ 接口方法默认修饰符是public, 也可以定义为 private 或者 protected

□ 一个类可以实现多个接口,也可以实现多个抽象类

□ 接口的字段默认都是 static 和 final 的, 而抽象类的字段声明可以是任意的

# 下面有关TCP和UDP描述正确的是

□ UDP传输数据效率比 TCP 好,但是 TCP 比 UDP 可靠性更高

□ TCP 3次握手是由服务端发起的.

□ TCP 建立链接要握手3次,断开链接也要握手3次

□ TCP 是属于传输层, IP 是属于网络层

# 下列关于HTTP请求消息GET和POST 方法描述正确的是

□ POST请求的数据在地址栏不可见  
□ GET请求提交的数据在理论上没有长度限制  
□ POST请求对发送的数据的长度限制在240——255个字符  
□ GET请求提交数据更加安全

# 关于 HTTP 响应代码,以下描述正确的是

□ 200 代表成功

□ 300 代表客户端重定向

□ 401 代表用户需要验证

□ 404 代表服务器没有该资源

□ 500 代表服务器内部错误

# 给定两个关系数据库表:

表名: user

字段: id,name,age(其中 id 代表身份证,name 代表姓名,age 代表年龄)

表名: cash

字段: id,money,bank (其中 id 代表身份证,money 代表存款,bank代表银行名称)

请按下面要求写出相应的 SQL 语句

1. 查询在任一银行存款大于1000的用户的姓名

Select distinct id, name from user u left join cash c on u.id= c.id and c.money >1000;

1. 查询在所有银行累计总存款大于10000 的用户的姓名

SELECT

nc.NAME

FROM

(

SELECT

sum( c.money ) AS totalCash,

u.NAME

FROM

user u

LEFT JOIN cash c ON u.id = c.id

GROUP BY

u.id

) nc

WHERE

nc.totalCash > 10000;

1. 查询每个银行中,总存款大于5000,且年龄为20 至30 的用户的姓名和银行名称

SELECT

nc.NAME ,nc.bank

FROM

(

SELECT

sum( c.money ) AS totalCash,

u.NAME ,

GROUP\_CONCAT(c.bank

) as bank

FROM

user u

LEFT JOIN cash c ON u.id = c.id

where age >=20 and age <=30

GROUP BY

u.id

) nc

WHERE

nc.totalCash > 5000 ;