**监控组开发岗位面试题**

1.请写出程序在的输出结果

|  |
| --- |
| **public** **class** ObjectSample {  **static** {  System.*out*.println("ObjectSample member");  }  **public** ObjectSample() {  System.*out*.println("ObjectSample construct");  }  **public** **static** **void** main(String[] args) {  **new** ObjectSample2();  }  }  **class** ObjectSample2 **extends** ObjectSample {  **static** {  System.*out*.println("ObjectSample2 member");  }  **public** ObjectSample2() {  System.*out*.println("ObjectSample2 construct");  }  } |

输出结果：

ObjectSample member

ObjectSample2 member

ObjectSample construct

ObjectSample2 construct

分析：

**new** ObjectSample2();会先加载ObjectSample2类，再实例化出一个对象

在类加载的初始化阶段，所有static静态代码块都被按照从父类到子类的顺序收集到类构造器clinit( )中，并执行。所以先打印出ObjectSample member，ObjectSample2 member。

实例化时，先执行父类构造函数，在执行子类构造函数，所以打印出ObjectSample construct、ObjectSample2 construct

2.请写出程序的输出结果

|  |
| --- |
| **public** **class** StringTest {  **public** **static** **void** main(String[] args) {  String a = "abc";  *test*(a);  }  **public** **static** **void** test(String a) {  String b = **new** String("abc");  *print*(a, b);  b = "abc";  *print*(a, b);  b = "a" + "bc";  *print*(a, b);    b = **new** String("a" + "bc");  *print*(a, b);  }    **private** **static** **void** print(String a, String b) {  System.*out*.println((a == b));  }  } |

输出结果：

false

true

true

false

分析：

1. b和a分别是两个String对象，计算a==b时，比较的是a、b两个变量在JVM堆中对应的两个String对象，所以false
2. 相同的字符串在常量池中只有一个，所以b=”abc”时，发现常量池已经有”abc”，b就直接引用到这个对象上，而没有新建对象，所以true
3. 编译时，编译器会把”a”+”bc”直接计算成”abc”，所以跟（2）相同
4. 编译器把”a”+”bc”直接计算成”abc”，并新建一个对象给b，所以false

3.请写出下面程序的运行结果

|  |
| --- |
| **public** **class** ExceptionTest {  **public** **static** **void** main(String[] args) {  *test*(1);  *test*(2);  }    **public** **static** **void** test(**int** i) {  **try** {  **if**(i == 1) {  **throw** **new** Exception();  }  **return**;  } **catch** (Exception e) {  System.*out*.println("catch exception");  **return**;  } **finally** {  System.*out*.println("finally print");  }  }  } |

输出结果：

catch exception

finally print

finally print

分析：

当i==1时，抛出异常执行catch函数，打印catch exception

finally中的代码无论是否异常或return都会执行，所以打印两次finally print

4.编写一个程序要求打印100以内的全部质数（质数定义为除了本身和1以外不能被任何数整除的数，假设1也为质数）

public class Main {

public static void main(String[] args) {

int[] num = primeNum(100);

for (int i=1; i<=100; i++)

if (!num[i])

System.out.println(i);

}

public static int[] primeNum(int n) {

boolean[] num = new boolean[n+1];

for (int i=2; i<=100; i++)

if (!num[i])

for (int j=2\*i; j<=100; j+=i)

num[j] = true;

return num;

}

}

所有质数（1除外）的倍数都不是质数，布尔数组num用来记录该值是否为质数，质数设为false。把质数的倍数都设为true即可，所以1到100遍历一次就行

时间复杂度：O(n) 空间复杂度：O(n)

5.请重构以下程序并给出理由

|  |
| --- |
| **public** **class** RefactorTest {    **public** **void** handle(**int** actionCode) {    **if** (actionCode == 1) {  function1();  } **else** **if** (actionCode == 2) {  function2();  } **else** **if** (actionCode == 3) {  function3();  } **else** **if** (actionCode == 4) {  function4();  } **else** **if** (actionCode == 5) {  function5();  } **else** {  defaultFunction();  }  }    **public** **void** function1() {  // ..  }    **public** **void** function2() {  // ..  }    **public** **void** function3() {  // ..  }    **public** **void** function4() {  // ..  }    **public** **void** function5() {  // ..  }    **public** **void** defaultFunction() {  // ..  }  } |

基类的逻辑清晰明了，代码简洁，易于理解，也方便扩展。有新的需求加进来的时候，只需要创建一个新的子类，在map中生成即可。

public class RefactorTest {  
  
 public void handle(int actionCode) {  
 Func func = new Func();  
 func.handle(actionCode);  
 }  
  
}  
  
class Func {  
 private Map<Integer, Parent> map = new HashMap<>();  
  
 public Func() {  
 add(new Son1());  
 }  
  
 public void handle(int code) {  
 this.map.get(code).function();  
 }  
  
 public void add(Parent parent) {  
 this.map.put(parent.getCode(), parent);  
 }  
}  
  
abstract class Parent {  
 abstract int getCode();  
 abstract void function();  
}  
  
class Class1 extends Parent {  
 public int getCode() {  
 return 1;  
 }  
 public void function() {  
 System.*out*.println("Class1");  
 }  
}  
  
Class2,Class3,Class3...

6.请重构一下程序并给出理由

|  |
| --- |
| **import** java.util.LinkedList;  **import** java.util.List;  **public** **class** RefactorTest2 {  **private** **static** **final** String *FIELD\_ID* = **null**;    **private** **static** **final** String *TABLE\_NAME\_A* = **null**;  **private** **static** **final** String *TABLE\_NAME\_B* = **null**;  **public** List<ValueObjectA> listObjectA() {  List<ValueObjectA> results = **new** LinkedList<ValueObjectA>();    String[] columns = **new** String[] {  *FIELD\_ID* // TabelA Columns ...  };    SQLiteDatabase db = **this**.getReadableDatabase();  Cursor c = db  .query(*TABLE\_NAME\_A*, columns, **null**, **null**, **null**, **null**, **null**);  **if** (c.moveToFirst()) {  **do** {  ValueObjectA objectA = **new** ValueObjectA();  // Set Object A properties  results.add(objectA);  } **while** (c.moveToFirst());  }  db.close();  **return** results;  }  **public** List<ValueObjectB> listObjectB() {  List<ValueObjectB> results = **new** LinkedList<ValueObjectB>();    String[] columns = **new** String[] {  *FIELD\_ID*, // TabelB Columns ...  };    SQLiteDatabase db = **this**.getReadableDatabase();  Cursor c = db  .query(*TABLE\_NAME\_B*, columns, **null**, **null**, **null**, **null**, **null**);  **if** (c.moveToFirst()) {  **do** {  ValueObjectB objectB = **new** ValueObjectB();  // Set Object B properties  results.add(objectB);  } **while** (c.moveToFirst());  }  db.close();  **return** results;  }  **private** SQLiteDatabase getReadableDatabase() {  // **TODO** Auto-generated method stub  **return** **null**;  }  } |

使用泛型，将两个函数复用成一个，减少重复代码，方便修改和扩展到新的数据表

public class RefactorTest2 {

private static final String FIELD\_ID = null;

public <T> List<T> listObject(String tableName, Class<T> clazz) {

List<T> results = new LinkedList<T>();

String[] columns = new String[] {

FIELD\_ID // Tabel Columns ...

};

SQLiteDatabase db = this.getReadableDatabase();

Cursor c = db.query(tableName, columns, null, null, null, null, null);

if (c.moveToFirst()) {

do {

T object = clazz.newInstance();

// Set Object A properties

results.add(object);

} while (c.moveToFirst());

}

db.close();

return results;

}

}

7. 写出jstorm节点间分发有哪些策略？每个分发策略的使用场景？

（1）随机分组，尽量均匀分布到下游Bolt中。使用场景：机器资源有限时通过随机分组保证每个Bolt中分发均匀、流量一致，提供整体利用率

（2）按字段分组，将数据中的某个字段作为分组依据。使用场景：针对数据中某一字段进行计算分析时，按该字段分组提高计算针对性

（3）全局分组，将全量分发到一个Bolt中。使用场景：主要针对事务性处理

8. 写出 storm计算用户uv 方法，输入 （时间，用户id），输入（每天用户uv值）,描述topo节点职责以及各节点的关系？如果需要数据准确，应该做哪些改进？

9. 编写一个SQL要求计算每种性别的每天占比的日环比，数据表如下

|  |  |  |
| --- | --- | --- |
| uid | gender | date（yyyymmdd） |
| UID1 | F | 20170210 |
| UID2 | M | 20170210 |
| UID1 | F | 20170211 |
| … | … | … |

select gender, (t1.c – t2.c)/t2.c ,DATE\_SUB(curdate(), INTERVAL 0 DAY) as date

(select gender, count(\*) as c from table where date = DATE\_SUB(curdate(), INTERVAL 0 DAY) group by gender) t1

left join

(select gender, count(\*) as c from table where date = DATE\_SUB(curdate(), INTERVAL 1 DAY) group by gender) t2

on t1.gender = t2.gender;