**中软国际考题-笔试**

1. 单选题

1、在Junit中@Test(timeout=1000)的作用是（D）

A.设置加载时间 B.设置延迟执行时间 C.设置测试用例执行时间 D.设置超时时间

2、Junit中有一个默认的Runner，是（A）；

1. BlockJUnit4ClassRunner
2. Suite Runner
3. Enclosed Runner

Categories Runner

3、以下哪个注解不属于junit。（ C ）

A . @Test B . @Before C . @Autowired D . @After

4、JUnit是一个什么工具？（A）

A.单元测试 B.数据库开发 C.程序报错 D.翻译代码

5、Junit中提供初始化测试的注解为（D）

A. @Setup B. @BeforeTest C. @init [D.@Before](mailto:D.@Before)

6、下列关于Junit特点描述错误的是（C）

A.JUnit 是一个开放的资源框架，用于编写和运行测试。

B.JUnit 测试允许你编写代码更快，并能提高质量。

C.JUnit 测试不能使用断言来预测结果。

D.Junit提供注释来识别测试方法

7、1、 以下哪个方法属于TestCase类的方法（A）。

A. setUp B.testUp C.setDown D.testFloat

8、Junit是一种（B）工具。

A.黑盒测试 B.白盒测试 C.性能测试 D.压力测试

9、Junit是一种（A）工具。

A.单元测试 B.集成测试 C.性能测试 D.压力测试

10、关于Junit描述，不正确的是：C

A. 软件质量最简单、最有效的保证； B. 是目标代码最清晰、最有效的文档；

C. 不可以优化目标代码的设计； D. 是代码重构的保障；

11、测试方法必须使用( )进行修饰，不能带参数. (C)

A. public int B.private int C.public void D.void

12. 下列哪个注解会在( )每一个测试方法运行后被执行一次。(A)

A.@After B.@Before C.AfterClass D.BeforeClass

1. 多选题
2. 下面哪些项属于Junit中的注解？（A、C、D）
3. BeforeClass B. Junit C. After
4. Ignore E. testFunction

2、junit3与junit4的区别（A、B、D、E）

A.junit4使用org.junit.\*而junit3.8使用的是junit.Framework.\*。

B.Junit4最大的亮点就是引入了注解(annotation)，通过解析注解就可以为测试提供相应的信息，抛弃junit3使用命名约束以及反射机制的方法。

C.Junit3中测试类必须继承TestCase类。

D.JUnit 4是与JUnit3完全不同的API。

E.在JUnit3.8中,可以选择使用若干运行机：文本型，AWT或者Swing。JUnit4仅使用文本测试运行机

3、JUnit的优势？（A、BCD）

A. 极限编程 B. 重构 C. 其他情况 D. 提高工作效率

4、测试可以分为（ ABC ）

A . 黑盒测试 B . 白盒测试 C . 灰盒测试 D . 红盒测试

5、以下关于Junit说法正确的是（B）

A 从软件工程的角度看，单元测试是可有可无的，因为对于实际功能事先影响甚微

B 使用Junit编写单元测试时，要考虑适度原则，不需要为所有的细节编写测试

C Junit可能独立测试出所有情况下代码的表现，因此对于开发人员来说非常重要

D Junit是测试人员常用的测试工具，开发人员并不一定需要掌握

6、下面关于Junit基本用法描述正确的是（A、B、C、D）

A.创建一个名为 TestJunit.java 的测试类。

B.向测试类中添加名为 testPrintMessage() 的方法。

C.向方法中添加 Annotaion @Test。

D.执行测试条件并且应用 Junit 的 assertEquals API 来检查。

7、单元测试的编写原则: （ABCD）（多选）

A.在eclipse中创建1个source folder命名为test(使用Maven后已要求创建)

B.测试类所在的包要求和被测试类的包一致

C.测试类要使用Test作为开头或结尾,例如 --> UserServiceTest

D.测试类的每个方法,都必须是可以独立执行的,不存在顺序或依赖

8、.Junit的常用注解有( ). BC

A.webServlet B.@Before C.@After D.init

9.单元测试的失败原因有( )AB

A.一般是由于测试结果和预期结果不一致引发的，表示测试的这个点发现了问题

B.是由代码异常引起的，它可以产生于测试代码本身的错误，也可以是被测试代码中隐藏的bug.

C.代码有编译错误

D.以上都不是

1. 填空题（每空1分，共10分）
2. 在JUnit4+的版本中可以使用\_\_\_\_\_\_\_\_\_\_ 注解来标注当前方法为测试方法。(@test)
3. 在Junit中方法是\_\_\_\_\_\_\_\_\_\_在执行每个测试方法之前执行的, \_\_\_\_\_\_\_\_\_\_方法是在执行每个测试方法之后执行的。(setup、testDown)
4. Junit是\_\_\_\_\_\_\_\_\_语言的单元测试框架。(Java)
5. 2.使用Junit出现InitializationError的解决方法\_\_\_\_\_\_\_\_\_\_。（项目中添加hamcrest-core-1.3.jar包）
6. JUnit测试框架目前大体分为两种：JUnit 3.8 和 JUnit 4.x
7. Junit是一套框架，继承\_\_\_\_\_\_\_\_\_\_\_类，就可以用Junit进行自动测试了。。(TestCase)
8. 一个完整的单元测试必须包含 初始化 、 运行测试 、 验证结果 清理测试数据 等步骤。
9. Junit API中最重要的程序包是junit.framework，其中包括的核心类有：Assert、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、TestResult。(TestCase、TestSuite)
10. JUnit的四大功能分别是：\_管理测试用例、\_\_\_\_\_\_\_\_\_\_\_\_\_\_、定义测试代码、\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（定义测试环境、检测测试结果）
11. Junit是 java 领域占有率非常高的一个单元 测试框架 ,已几近成为了单元测试的标准。
12. 单元测试常用哪个（ ）方法. (assertEquals)
13. 测试方法前要用（ ）注解. (@Test)
14. 测试方法必须使用( )进行修饰，不能带参数. (public void)
15. 一般使用单元测试会新建一个test目录存放测试代码，在生产部署的时候只需要将( )下代码删除即可(test目录)
16. 测试代码的包应该和被测试代码包( )保持一致(结构)
17. 测试单元中的每个方法必须可以独立测试，方法间不能有任何( )(依赖)
18. 测试类一般使用Test作为( )的后缀(类名)
19. 测试方法使一般用test作为( )的前缀(方法名)
20. 判断题
21. 测试类一般需要和目标类放在一起，但编译成的class文件不要放在一起的。（ F ）
22. 单元测试主要是来测试程序的结果和自己期望的值是否相等。（ T ）
23. Junit测试是程序员测试，是否为黑盒测试。（F）
24. Junit只是单元测试工具，并不能实现回归测试。（T）
25. Junit主要用于测试。（ T ）
26. Junit是xUnit的一个子集。 （ T ）
27. JUnit 测试可以自动运行并且检查自身结果并提供即时反馈。所以也没有必要人工梳理测试结果的报告。 （ T ）
28. JUnit 测试可以被组织为测试套件，包含测试用例，甚至其他的测试套件。（ T ）
29. JUnit 在一个条中显示进度。如果运行良好则是红色；如果运行失败，则变成绿色。（ F ）
30. JUnit 是一个开放的资源框架，用于编写和运行测试。 （ T ）
31. Junit单元测试支持多线程。 （ F ）
32. 单元测试是一个对单一实体（类或方法）的测试。 （ T ）
33. 要想使用Junit必须先安装Junit-jar包。（ T ）
34. Junit是测试工具的包，在正式发布上线时必须包含此包（ F）
35. JUnit 是 Java 的一个框架，所以需要安装JDK。（ T ）
36. BeforeClass注释指出这是附着在静态方法必须执行一次并在类的所有测试之前。发生这种情况时一般是测试计算共享配置方法(如连接到数据库)。 （ T ）
37. JUnit是一种测试代码的框架，测试的目的是：保证代码没错，而不是保证代码正确。（T）
38. 在 JUnit 4 中，单元测试用例的执行顺序为：@BeforeClass->@Before->@Test->@After->@AfterClass。（ T ）
39. maven测试一般文件位于test/java下对应的包下的的测试类。（T）

五、简答题

1、简述Junit的基本功能是什么？

答案：

1. 管理测试用例。修改了哪些代码，这些代码的修改会对哪些部分有影响，通过JUnit将这次的修改做个完整测试。这也就JUnit中所谓的TestSuite。

2. 定义测试代码。这也就是JUnit中所谓的TestCase，根据源代码的测试需要定义每个TestCase，并将TestCase添加到相应的TestSuite方便管理。

3. 定义测试环境。在TestCase测试前会先调用“环境”配置，在测试中使用，当然也可以在测试用例中直接定义测试环境。

4. 检测测试结果。对于每种正常、异常情况下的测试，运行结果是什么、结果是否是我们预期的等都需要有个明确的定义，JUnit在这方面提供了强大的功能。

2、简述在eclipse中如何使用junit？

答案：

第一种方法：

1、下载junit的jar包，目前它的版本是junit4.12，可以从www.junit.org上下载。

2、在要使用Junit的project名上，点击properties--java build path-libraries, 点击Add External JARs,把Junit包点上就行了。如图所示,点击Add External JARs，选择相应的Junit包

第二种方法：在要使用Junit的project名上，点击properties--java build path-libraries, 点击Add library,选择JunitT即可。如图所示，点击Add library，在弹出的对话框中选择Jnuit。

3、JUnit的好处。

答案：

现在世面上有很多的测试工具，比如说NUNIT，PHPUNIT等。但是在JAVA

的世界里面JUnit是最适合我们的单元测试工具。

A：可以使测试代码与产品代码分开。

B：针对某一个类的测试代码通过较少的改动便可以应用于另一个类的测试 。

C：易于集成到测试人员的构建过程中，JUnit和Ant的结合可以实施增量开发。

D：JUnit是公开源代码的，可以进行二次开发。

E：可以方便地对JUnit进行扩展。

4.JUnit的编写原则。

答案：

A: 是简化测试的编写，这种简化包括测试框架的学习和实际测试单元的编写。

B: 是使测试单元保持持久性。

C: 是可以利用既有的测试来编写相关的测试。

5、创建测试类的步骤

答案：

1）在工程下面创建一个名为 test 的 Source Folder ,这个源程序文件夹和 src文件夹的阶级是相等的

2）测试类和目标类的包名要相同，这样的好处是测试类就不要import导入目标类的源码

3）测试类的命名规则：假设目标类的名字是 Calculator，则创建的测试类应命名为 CalculatorTest 或者 TestCalculator

6、什么叫Junit？

答案：

①JUnit是用于编写可复用测试集的简单框架，是xUnit的一个子集。xUnit是一套基于测试驱动开发的测试框架，有PythonUnit、CppUnit、JUnit等。

②Junit测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。

③多数Java的开发环境都已经集成了JUnit作为单元测试的工具，比如Eclipse。

7、什么是一个单元测试用例?

答案：

单元测试用例是一部分代码，可以确保另一端代码（方法）按预期工作。为了迅速达到预期的结果，就需要测试框架。JUnit 是 java 编程语言理想的单元测试框架。

一个正式的编写好的单元测试用例的特点是：已知输入和预期输出，即在测试执行前就已知。已知输入需要测试的先决条件，预期输出需要测试后置条件。

每一项需求至少需要两个单元测试用例：一个正检验，一个负检验。如果一个需求有子需求，每一个子需求必须至少有正检验和负检验两个测试用例。

8、Junit的特性？

答案：1、用于测试期望结果的断言(Assertion)

2、用于共享共同测试数据的测试工具

3、用于方便的组织和运行测试的测试套件

4、图形和文本的测试运行器

9、简述单元测试在整个软件开发过程中的作用

- 单元测试能够保证某个单元代码的质量，让程序员对自己的代码保持信心

- 单元测试可以演示代码的使用方式，是活的说明文档

- 在TDD（测试驱动开发）开发模式中，单元测试是代码的验收规范

10、请说明Junit的执行过程。

答案：beforeClass() 方法首先执行，并且只执行一次。

afterClass() 方法最后执行，并且只执行一次。

before() 方法针对每一个测试用例执行，但是是在执行测试用例之前。

after() 方法针对每一个测试用例执行，但是是在执行测试用例之后。

在 before() 方法和 after() 方法之间，执行每一个测试用例。

11、请列举Junit支持的测试方式，列举三种。

答案：1）测试套件意味着捆绑几个单元测试用例并且一起执行他们。在 JUnit 中，@RunWith 和 @Suite 注释用来运行套件测试。

2）忽略测试：有时可能会发生我们的代码还没有准备好的情况，这时测试用例去测试这个方法或代码的时候会造成失败。@Ignore 注释会在这种情况时帮助我们。

一个含有 @Ignore 注释的测试方法将不会被执行。

如果一个测试类有 @Ignore 注释，则它的测试方法将不会执行。

3）Junit 4 引入了一个新的功能参数化测试。参数化测试允许开发人员使用不同的值反复运行同一个测试。

12、 请说明在Eclipse中如何使用Junit插件。

答案：1）下载 Junit archive，例如选择junit-4.10.jar。

2）设置 Eclipse 环境：打开 eclipse -> 右击 project 并 点击 property > Build Path > Configure Build Path，然后使用 Add External Jar 按钮在函数库中添加 junit-4.10.jar

13、JUnit3和JUnit4的区别: （5分）

答案：

JUnit3中,测试类都需要继承TestCase,而且测试类需要使用testXXX来作为开头,若然希望在测试方法前运行某个初始化方法,这个方法的名称必须是setUp,在测试方法后运行某个方法名称必须是tearDown

在JUnit4中不需要继承TestCase,而是使用了更为方便的annotation,只需要使用@Test来表示,测试前初始化方法使用@Before,测试后释放资源方法使用@After,若只想该方法只执行一次,使用@BeforeClass和@AfterClass

14、列举出Junit的常用注解

答案：

@Before：初始化方法，在任何一个测试方法执行之前，必须执行的代码。

@After：释放资源，在任何一个测试方法执行之后，需要进行的收尾工作。

@Test：测试方法，表明这是一个测试方法。在 JUnit 中将会自动被执行。。而且，它还可以测试期望的异常，例如，我们刚刚的那个空指针异常就可以这样：@Test(expected=NullPointerException.class)。

@Ignore：忽略的测试方法，标注的含义就是“某些方法尚未完成，咱不参与此次测试”。

@BeforeClass：针对所有测试，也就是整个测试类中，在所有测试方法执行前，都会先执行由它注解的方法，而且只执行一次。

@AfterClass：针对所有测试，也就是整个测试类中，在所有测试方法都执行完之后，才会执行由它注解的方法，而且只执行一次。当此 Annotation 也是 JUnit 4 新增的功能，与 @BeforeClass 是一对。

15、测试失败引起的原因是什么?

Failure：一般是由于测试结果和预期结果不一致引发的，表示测试的这个点发现了问题

error：是由代码异常引起的，它可以产生于测试代码本身的错误，也可以是被测试代码中隐藏的bug.

16.Test注解作用

@Test:将一个普通方法修饰成一个测试方法

@Test(excepted=xx.class): xx.class表示异常类，表示测试的方法抛出此异常时，认为是正常的测试通过的

@Test(timeout=毫秒数) :测试方法执行时间是否符合预期

17.BeforeClass注解作用

@BeforeClass： 会在所有的方法执行前被执行，static方法

18.AfterClass注解作用

@AfterClass：会在所有的方法执行之后进行执行，static方法

19.Before注解作用

@Before：会在每一个测试方法被运行前执行一次

20.After注解作用

@After：会在每一个测试方法运行后被执行一次

21.Ignore注解作用

@Ignore：所修饰的测试方法会被测试运行器忽略

22.RunWith注解作用

@RunWith：可以更改测试运行器org.junit.runner.Runner

六、程序实现题

1、现有工具类如下：

public class Calculator {

/\*\*

\* 传入两个参数，求和

\* @param a

\* @param b

\* @return

\*/

public int add(int a,int b){

return a+b;

}

/\*\*

\* 传入两个参数，求差

\* @param a

\* @param b

\* @return

\*/

public int sub(int a,int b){

return a-b;

}

}

根据该类编写对应的Junit测试用例。

|  |
| --- |
| **public class CalculatorTest {**    **@Test**  **//测试 add()方法**  **public void testAdd(){**  **Calculator c = new Calculator();**  **int result = c.add(1, 2);**  **Assert.assertEquals(result, 3);**  **}**    **@Test**  **//测试 sub()方法**  **public void testSub(){**  **Calculator c = new Calculator();**  **int result = c.sub(2, 1);**  **Assert.assertEquals(result, 1);**  **}**    **}** |

2、写一个加减乘除程序，采用junit对其进行测试

（10分）

|  |
| --- |
| import java.util.Scanner;  @Test  public class Test{    Scanner input=new Scanner(System.in);    public Test(){        System.out.print("请选择\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");    System.out.println("1.减法\n2.加法\n3.乘法\n4.除法\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");       int getIt=input.nextInt();      System.out.print("请输入数A：");        int firstone=input .nextInt();      System.out.print("请输入数B：");        int secondone=input.nextInt();       switch(getIt){          case 1:             System.out.print("A-B="+(firstone-secondone));          break;          case  2:             System.out.print("A+B="+firstone+secondone);          break;          case 3:             System.out.print("A\*B="+firstone\*secondone);          break;          case 4:             System.out.print("A/B="+firstone/secondone);          break;          default:             System.out.print("对不起，你输入的数据有误。");  }       }    } |

3、请编写一个具有spring测试环境的测试类。

答案：

@RunWith(SpringJUnit4ClassRunner.class) //使用junit4进行测试

@ContextConfiguration(locations={"classpath:applicationContext.xml"}) //加载配置

public class BaseJunit4Test{

}

4、有如下代码，请为此编写测试用例

*public class* Calculator {

*public double* sum(*double* a,*double* b){

*if*(a<0||b<0){

*throw new* IllegalArgumentException("参数错误");

}

*return* a+b;

}

}

*package* me.maiz.project.mblog;

*import* junit.framework.TestCase;

*import* org.junit.Test;

*import static* org.junit.Assert.\*;

*public class* CalculatorTest *extends* TestCase {

@Test

*public void* testSumException(){

*try* {

*new* Calculator().sum(-1, -1);

}*catch* (IllegalArgumentException e){

*assertEquals*(e.getMessage(),"参数错误");

}

}

@Test

*public void* testSum(){

*try* {

*final double* sum = *new* Calculator().sum(1, 1);

*assertEquals*(sum,2.0);

}*catch* (IllegalArgumentException e){

*fail*();

}

}

}

5、使用忽略测试，给方法divisionWithException添加，请写出具体代码

|  |
| --- |
| public class JunitTest3 {  @Ignore("Not Ready to Run")  @Test  public void divisionWithException() {  System.out.println("Method is not ready yet");  }  } |

6、使用超时测试，并设置超时时间为1秒，写出具体代码：

|  |
| --- |
| public class JunitTest4 {  @Test(timeout = 1000)  public void infinity() {  while (true);  }  } |

### 7、使用Junit测试如下代码：

|  |
| --- |
| public class MyCode {  public int m1() {  System.out.println("数字型");  return 1;  }  public String m2() {  System.out.println("null");  return null;  }  } |

答案：

|  |
| --- |
| import static org.junit.Assert.\*;  import org.junit.Test;  public class TestMyCode {  MyCode mc = new MyCode();  @Test  public void testM1() {  assertEquals(1, mc.m1());  }  @Test  public void testM2() {  assertNull(mc.m2());  }  } |

8、1、编写计算类的测试用例我们仅对“加、减、乘、除”四个方法进行测试。（10分）

|  |
| --- |
| 计算类：  public class Calculator {  private static int result; // 静态变量，用于存储运行结果  public void add(int n) {  result = result + n;  }  public void substract(int n) {  result = result - 1; //Bug: 正确的应该是 result =result-n  }  public void multiply(int n) {  } // 此方法尚未写好  public void divide(int n) {  result = result / n;  }  public void square(int n) {  result = n \* n;  }  public void squareRoot(int n) {  for (; ;) ; //Bug : 死循环  }  public void clear() { // 将结果清零  result = 0;  }  public int getResult(){  return result;  }  }  测试用例类：  public class CalculatorTest {  private static Calculator calculator = new Calculator();  @Before  public void setUp() throws Exception {  calculator.clear();  }  @Test  public void testAdd() {  calculator.add(3);  calculator.add(4);  assertEquals(7, calculator.getResult());  }  @Test  public void testSubstract() {  calculator.add(8);  calculator.substract(3);  assertEquals(5, calculator.getResult());  }  @Ignore("Multiply() Not yet implemented")  @Test  public void testMultiply() {  fail("Not yet implemented");  }  @Test  public void testDivide() {  calculator.add(8);  calculator.divide(2);  assertEquals(4, calculator.getResult());  }  } |