1. 什么是黑盒和白盒测试？

黑盒测试：不需要写代码，给输入值，看程序是否能够输出期望的值。  
白盒测试：需要写代码的。关注程序具体的执行流程。

2. Assert.assertEquals(期望的结果,运算的结果);

进行单元测试时，一定先要引入Junit包才可以使用。并且该注解必须在方法名上。

3.  @Before:  
                \* 修饰的方法会在测试方法之前被自动执行  
            \* @After:  
                \* 修饰的方法会在测试方法执行之后自动被执行

4. 一个单元测试用例执行顺序为：@BeforeClass（必须是static void） –> @Before –> @Test –> @After –> @AfterClass （必须是static void）

/\*\*

\* 初始化方法：

\* 用于资源申请，所有测试方法在执行之前都会先执行该方法

\*/

@Before

public void init(){

System.out.println("init...");

}

/\*\*

\* 释放资源方法：

\* 在所有测试方法执行完后，都会自动执行该方法

\*/

@After

public void close(){

System.out.println("close...");

}

5.反射：将类的各个组成部分封装为其他对象，这就是反射机制  
\* 好处：  
            1. 可以在程序运行过程中，操作这些对象。  
            2. 可以解耦，提高程序的可扩展性。

6. JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

7. 获取Class对象的方式：  
        1. Class.forName("全类名")：将字节码文件加载进内存，返回Class对象  
            \* 多用于配置文件，将类名定义在配置文件中。读取文件，加载类  
        2. 类名.class：通过类名的属性class获取  
            \* 多用于参数的传递

        3. 对象.getClass()：getClass()方法在Object类中定义着。（一般用第三种）  
            \* 多用于对象的获取字节码的方式  
8.反射对象功能概述

获取功能

1. 获取成员变量们

\* Field[] getFields() ：获取所有public修饰的成员变量

\* Field getField(String name) 获取指定名称的 public修饰的成员变量

\* Field[] getDeclaredFields() 获取所有的成员变量，不考虑修饰符

\* Field getDeclaredField(String name)

2. 获取构造方法们

\* Constructor<?>[] getConstructors()

\* Constructor<T> getConstructor(类<?>... parameterTypes)

\* Constructor<T> getDeclaredConstructor(类<?>... parameterTypes)

\* Constructor<?>[] getDeclaredConstructors()

3. 获取成员方法们：

\* Method[] getMethods()

\* Method getMethod(String name, 类<?>... parameterTypes)

\* Method[] getDeclaredMethods()

\* Method getDeclaredMethod(String name, 类<?>... parameterTypes)

4. 获取全类名

\* String getName()

通过class字节码对象可以获取该类的方法以及属性

9. Class对象获取Field

Class personClass=Person.class;

Field[] fields=personClass.getFields();

for (Field field : fields) {//iter

System.out.println("getFileds(): "+field);

}

Field field=personClass.getField("a");

System.out.println("getField(): "+field);

System.out.println("=====================================================");

Field a=personClass.getField("a");

Person p=new Person();

Object value=a.get(p);

System.out.println(p);

a.set(p,1);

System.out.println(p);

System.out.println("=====================================================");

Field[] fiels2=personClass.getDeclaredFields();

for (Field field2 : fiels2) {

System.out.println("getDeclaredFields(): "+field2);

}

Field field3=personClass.getDeclaredField("name");

System.out.println("getDeclaredField(): "+field3);

10. Class对象功能\_获取Constructor

构造器最大的用处就是在创建对象时执行初始化，当创建一个对象时，系统会为这个对象的实例进行默认的初始化。如果想改变这种默认的初始化，就可以通过自定义构造器来实现。

如果使用空参数构造方法创建对象，操作可以简化：Class对象的newInstance

Class personClass=Person.class;

Constructor constructor=personClass.getConstructor();

Object p1=constructor.newInstance();

System.out.println(p1);

System.out.println("=========================================");

Constructor constructor1=personClass.getConstructor(String.class,String.class);

Object person1=constructor1.newInstance("aaa","bbb");

System.out.println(person1);

System.out.println("=========================================");

Object person=personClass.newInstance();

System.out.println(person);

11. class对象获取Method

使用method.getParameterTypes();可以获取该方法的参数

Class personClass=Person.class;

Method[] methods=personClass.getMethods();

for (Method method : methods) {

System.out.println(method);

}

System.out.println("=============================");

Method method01=personClass.getMethod("eat");

Person person=new Person();

method01.invoke(person);

System.out.println("==============================");

Method method02=personClass.getMethod("eat",String.class);

method02.invoke(person,"zhangsan")

12.反射

首先要从配置文件中读取该类的全限定类名，然后用class.forName()，来获取该类的class对象，再通过class对象获取并执行成员方法。

Properties properties=new Properties();

ClassLoader cl=ClassTest05.class.getClassLoader();

InputStream is=cl.getResourceAsStream("source.properties");

properties.load(is);

String className=properties.getProperty("className");

String method=properties.getProperty("methodName");

Class personClass=Class.forName(className);

Object person=personClass.newInstance();

Method method1=personClass.getMethod("eat");

method1.invoke(person);

13\* 作用分类：

①编写文档：通过代码里标识的注解生成文档【生成文档doc文档】

②代码分析：通过代码里标识的注解对代码进行分析【使用反射】

③编译检查：通过代码里标识的注解让编译器能够实现基本的编译检查【Override】

注解就是为了简化开发，避免写过多的代码，不利于程序的扩展以及维护        \* @Override    ：检测被该注解标注的方法是否是继承自父类(接口)的  
        \* @Deprecated：该注解标注的内容，表示已过时  
        \* @SuppressWarnings：压制警告  
            \* 一般传递参数all  @SuppressWarnings("all")

14. 编写代码实现自定义注解

public @interface MyAnno {

int value();

Person per();

MyAnno2 anno2();

String[] strs();

/\*String name() default "张三";\*/

/\*String show2();

Person per();

MyAnno2 anno2();

String[] strs();\*/

}

15. 元注解：用于描述注解的注解  
            \* @Target：描述注解能够作用的位置  
                \* ElementType取值：  
                    \* TYPE：可以作用于类上  
                    \* METHOD：可以作用于方法上  
                    \* FIELD：可以作用于成员变量上  
            \* @Retention：描述注解被保留的阶段  
                \* @Retention(RetentionPolicy.RUNTIME)：当前被描述的注解，会保留到class字节码文件中，并被JVM读取到  
            \* @Documented：描述注解是否被抽取到api文档中  
            \* @Inherited：描述注解是否被子类继承