1. (单选题)用于指定注解存活时间的元注解为（              ）

A. `@Target`：定义注解可以用于哪些Java元素（类、方法等）。

B. `@Retention`：定义注解保留的阶段（源代码、类文件、运行时）。

C. `@Documented`：指示注解信息应包含在javadoc中。

D. `@Inherited`：允许注解被继承，子类继承父类的注解。

2.(单选题)为了使得注解在程序运行时被反射读取，则应设置@Retention取值为（     ）

A. RetentionPolicy.SOURCE

B. RetentionPolicy.CLASS

C. RetentionPolicy.RUNTIME：

D. RetentionPolicy.JVM

3. (单选题)@Override注解只在源码阶段保留，它的@Retention取值为（            ）

A. RetentionPolicy.SOURCE

B. RetentionPolicy.CLASS

C. RetentionPolicy.RUNTIME

D. RetentionPolicy.JVM

4. (单选题)用于指定注解运用地方的元注解为（              ）

A. @Target

B. @Retention

C. @Documented

D. @Inherited

5. (单选题)设置一个注解只能修饰方法和类型（包括：类、接口、枚举）的元注解代码应为

A. @Target(ElementType.METHOD,ElementType.TYPE)

B. @Target({ElementType.METHOD,ElementType.TYPE})

C. @Retention(ElementType.METHOD,ElementType.TYPE)

D.@Retention({ElementType.METHOD,ElementType.TYPE})

6. (单选题)以下不属于Java注释的为（ ）

A. //这是单行注释

B./\*

这是多行注释

\*/

C./\*\*

 \* 这是文档注释

 \* \*/

1. <!这是一个单行注释 >

7. (多选题)抑制一个类上的所有警告，则能在该类上使用的代码为（ ）

A. @SuppressWarnings("all")

B. @SuppressWarnings(value="all")

C. @SuppressWarnings(value={"all"})

D. @SuppressWarnings(["all"])

E. @SuppressWarnings(value=["all"])

8. (多选题)关于Junit说法正确的是（ ）

A. Junit框架，可以独立于main方法完成单元测试

B. @Before注解修饰的方法，只在所有的测试方法前运行一次，而不会在每一个测试方法前都运行一次

C. Junit框架中，不能使用断言判断测试结果是否正确

D. 使用@Test注解的方法，必须是public void类型，且不能有任何参数。

E. Junit支持一个项目中所有方法的自动化测试

9. (填空题)已知Student的类定义为：

package fafu.jsxy;

public class Student {

    public void study() {

        System.out.println("studying");

    }

    public void speak(){

        System.out.println("speaking");

    }

    private void sleep(){  //私有方法

        System.out.println("sleeping");

    }

}

TestAnnotation类的功能为：通过反射，执行Student类中所有方法。使用反射的好处为：不需要知道Student类中定义的方法的名字，就能够执行Student类中的所有的方法。

请在下划线处填写正确的代码。

package fafu.jsxy;

import java.lang.reflect.Method;

public class TestAnnotation {

    public static void main(String[] args) throws Exception{

        Class \_\_\_studentClass\_\_\_\_ = Student.class; //获得Student的字节码对象

        Student student = (\_\_Student\_\_\_)studentClass.newInstance(); //通过字节码对象，创建Student对象

        \_\_\_Method\_\_\_\_[] methods = studentClass.getDeclaredMethods(); //通过字节码对象，获得Student类中的所有方法

        for (Method m:methods){

            m.setAccessible(true);  //暴力反射

            m.\_\_\_invoke\_\_(student); //执行方法

        }

    }

}