1. Create a custom annotation called@Test which can be only applied on a method implying that the following method is a testcase

package anno;

import java.lang.annotation.\*;

import java.lang.annotation.RetentionPolicy;

import java.lang.reflect.Method;

@Retention (RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

@interface Test{

String s();

}

class testcase{

@Test (s="HI")

void testcase()

{

}

}

public class annooo {

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

testcase t = new testcase();

Method m=t.getClass().getMethod("HI");

Test t1=m.getAnnotation(Test.class);

System.out.println(t1.s());

}

}

Output:

HI

1. Build a custom annotation called @info

package anno;

import java.lang.annotation.Annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

//Marker annotation

//single value annotation

//multi value annotation

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@interface info{

int AuthorID();

String Date();

String Time();

int Version();

}

@info (AuthorID=1,Date="18-01-2022",Time="4pm",Version=1)

class developer{

String Author,Supervisor,desc;

public developer(String Author,String Supervisor,String desc)

{

this.Author = Author;

this.Supervisor = Supervisor;

this.desc=desc;

System.out.println("Author:" + this.Author+ " "+ "Supervisor:" +this.Supervisor+" "+"desc:" +this.desc);

}

}

public class exp {

public static void main(String[] args)

{

developer t=new developer("Ram","Raj","Goodmrng");

Class c=t.getClass();

Annotation a=c.getAnnotation(info.class);

info t1=(info)a;

System.out.println("AuthorId:" +t1.AuthorID());

System.out.println("Date:" +t1.Date());

System.out.println("Time:"+t1.Time());

}

}

Output:

Author:Ram Supervisor:Raj desc:Goodmrng

AuthorId:1

Date:18-01-2022

Time:4pm

1. Create a custom annotation called @Execute to be applied on methods. Placing the @Execute method on a method implies that method should be invoked using Reflection API.

**package** year;

**import** java.lang.reflect.Method;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

**import** java.lang.reflect.Method;

@Target (value = ElementType.***METHOD***)

@Retention (RetentionPolicy.***RUNTIME***)

**@interface** Execute

{

**int** Sequence();

}

**class** sequence {

@Execute(Sequence=2)

**public** **void** method1()

{

System.***out***.println("Method 1");

}

@Execute(Sequence=1)

**public** **void** method2()

{

System.***out***.println("Method 2");

}

@Execute(Sequence=3)

**public** **void** method3()

{

System.***out***.println("Method 3");

}

}

**public** **class** sequence1 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

sequence s = **new** sequence();

Method[] methods = s.getClass().getMethods();

**for** (Method m : methods) {

Execute ex = m.getAnnotation(Execute.**class**);

**if** (ex != **null**) {

**try** {

m.invoke(s);

}

**catch** (Exception e)

{

e.printStackTrace();

}

}

}

}

}

Output:

Method 2

Method 3

Method 1