1.Create a custom annotation called @Test which can be only applied on a method implying that the following method is a test-case. (Is it possible to restrict the annotation to be applied only on a non-static method?)

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@interface Test{

}

class TestMethod{

@Test

void run() {

System.out.println("HELLO");

}

}

public class Annotation1 {

public static void main(String[] args) {

TestMethod s = new TestMethod();

s.run();

}

}

2. Build a custom annotation called @Info, which can be used by developers on a class, a property, or a method. The developer can provide the following information when using this annotation:

a) AuthorID: <> - (Mandatory Input) b) Author: <> - (Optional Input) c) Supervisor: <> - (Optional Input) d) Date: <> - (Mandatory Input) e) Time: <> - (Mandatory Input) f) Version: <> - (Mandatory Input) g) Description: <> - (Optional Input)

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

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

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

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

@Target(ElementType.***TYPE***)

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

**@interface** info{

**int** AuthorID();

String Author()**default**"None";

String Supervisor() **default**"None";

String Date();

String Time();

String Version();

String Description() **default**"NA";

}

@info(AuthorID=2244,Author="Aparna",Date="23-02-2022",Time="10:21",Version="Java 8")

**class** developer{

**void** dev() {

System.***out***.println("Details");

}

}

**public** **class** Annotation2 {

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

developer s = **new** developer();

s.dev();

@SuppressWarnings("rawtypes")

Class a = s.getClass();

@SuppressWarnings("unchecked")

java.lang.annotation.Annotation p = a.getAnnotation(info.**class**);

info i = (info)p;

System.***out***.println(i);

}

}

3.

**import** java.lang.annotation.\*;

**import** java.lang.reflect.\*;

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

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

**@interface** Execute{

**int** Sequence();

}

**@interface** Order{

**int** value();

}

**class** Myclass{

@Order(value=1)

@Execute(Sequence=2)

**public** **void** myMethod1(){System.***out***.println("one");}

@Order(value=2)

@Execute(Sequence=1)

**public** **void** myMethod2() {System.***out***.println("Two");}

@Order(value=3)

@Execute(Sequence=3)

**public** **void** myMethod3() {System.***out***.println("Three");}

}

**public** **class** Annotation3 {

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

Myclass h=**new** Myclass();

Method m=h.getClass().getMethod("myMethod1");

Method z=h.getClass().getMethod("myMethod2");

Method q=h.getClass().getMethod("myMethod3");

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

Execute ma=z.getAnnotation(Execute.**class**);

Execute no=q.getAnnotation(Execute.**class**);

System.***out***.println(manno);

System.***out***.println(ma);

System.***out***.println(no);

}}