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.\*;

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

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

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

**@interface** Test

{

String str();

}

//Applying annotation @Test

**class** Hello

{

@Test(str="This is Test Annotation")

**public** **void** testCase(){}

}

//Accessing annotation @Test

**class** first

{

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

{

Hello h=**new** Hello();

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

Test ts=m.getAnnotation(Test.**class**);

System.***out***.println(ts.str());

}

}

2) Build arcustom 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) Authorld: <<Developers ID>>- (Mandatory Input)

b) Author: <<Developers name>>- (Optional Input)

c) Supervisor: <<Developers Supervisor>>-(Optional Input)

d) Date: <<String Date">>- (Mandatory Input)

Time: <<String Time">>- (Mandatory Input)

f) Version: <<Numerical Version >> - (Mandatory Input)

g) Description: <<Description of the class, method, or property >>-(Optional Input)

------------------------------------class secondClass-----------------------------------------

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

**public** **class** secondClass

{

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

**throws** NoSuchMethodException, SecurityException

{

second h = **new** second();

h.display();

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

info st = m.getAnnotation(info.**class**);

System.***out***.println("Author id : " +st.ID());

System.***out***.println("Author name: " +st.name());

System.***out***.println("Superviser name: " +st.superviser());

System.***out***.println("Date: " +st.date());

System.***out***.println("Time: " +st.time());

System.***out***.println("Version: " +st.version());

}

}

-------------------------class second--------------------------------

**import** java.lang.annotation.Documented;

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

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

@Documented

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

**@interface** info

{

**int** ID();

String name();

String superviser();

String date();

String time();

**int** version();

}

// Applying annotation.

**public** **class** second

{

// Importing annotation on a method display().

@info(ID = 632, name = "AJAY", superviser = "CAPG",date="18/01/2022",time="11:35:50 hrs",version=11)

**public** **void** display()

{

System.***out***.println("Hello @info annotation");

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

}

}

3) 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 (invoking the method using Reflection API is out of scope of this assignments). The annotation @Execute should have an optional property "sequence" which can be given values such as 1, 2, 3... In the order of priority. In case the sequence property is not used the API may Invoke methods random order.

------------------------------this is class thirdMain-------------------------

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

**public** **class** thirdMain {

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

third\_a ta = **new** third\_a();

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

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

Execute exe = method.getAnnotation(Execute.**class**);

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

**try** {

method.invoke(ta);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

}

}

--------------------------------this is class third\_a---------------------------------------------

**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;

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

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

**@interface** Execute

{

**int** Sequence();

}

**public** **class** third\_a

{

@Execute(Sequence=2)

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

{

System.***out***.println("This is method 1");

}

@Execute(Sequence=1)

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

{

System.***out***.println("This is method 2");

}

@Execute(Sequence=3)

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

{

System.***out***.println("This is method 3");

}

}