Workshop Link: https://attendee.gotowebinar.com/register/1011365310966029397

Annotations:
—----------
Annotations is a java feature provided by JAVA in its JDK1.5 version.

In Java applications, Annotations are able to provide some description or metadata about the program.

Q)In Java applications, to provide description we have already comments then what is the requirement to use Annotations?
—----------
Ans:
—----
In Java applications, if we provide description along with comments then the Lexical Analyzer will remove comments and its description from the Java applications at the time of compilation. IN Java applications, comments and their description existed up to .java file only, they did not exist up to .class file and up to runtime of the application.

As per the requirements like
1. To simplify the Debugging process.
2. To simplify the testing process.
3. For the Server side applications
4. For the Database related applications like Hibernate applications
   —----
   —-----

We have to bring the description up to the Runtime of the application.

To bring the description up to .java file, up to .class file and up to Runtime of the application we have to use Annotations over the comments.

Q)In Java applications, to provide description at runtime we already have XML documents then what is the requirement to use Annotations?
—----------
Ans:
—----
If we want to use XML documents in Java applications then we are able to get the following problems.
1. We must learn XML tech.
2. Every time we have to check whether the XML document is in a well formed format or not.

3. Every time we have to check whether the XML document was provided at the proper locations or not.
4. Every time Developers must check whether we are using the right parser or not to read data from the XML document.
5. We must use the XML related Java library to read data from the XML documents.

To overcome all the above problems we must use a Java alternative that is “Annotations”.

Note: In JAVA / J2EE applications , we are using Annotations as alternatives to the XML documents.

```
XML Based Technologies          Annotation Based Technologies[XML is opt]
—-----------------------        —-----------------------------------------
Up to JDK1.4  —-------------------> From JDK1.5
JDBC3.x  —-------------------------> JDBC4.x
Servlet 2.5 —-----------------------> Servlet 3.x
Struts 1.x —-------------------------> Struts 2.x
JSF 1.x —---------------------------> JSF 2.x
EJBs 2.x —---------------------------> EJBs 3.x
Spring 2.4 —-------------------------> Spring 2.5
Hibernate 3.2.4 —--------------------> Hibernate 3.2.5
—----
—----
```

Note: Annotations are the alternatives for the XML documents within the Java applications, but Annotations are not the replacement for XML documents outside of the Java applications.

EX: In Servlets applications, we must configure Servlets in deployment Descriptor that is a web.xml file with the following tags.

```
public class LoginServlet extends HttpServlet{
      —-----
}
```

web.xml
—-------
```
<web-app>
      <servlet>
            <servlet-name>loginServlet</servlet-name>
            <servlet-class>LoginServlet</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>loginServlet</servlet-name>
            <url-pattern>/login</url-pattern>
      </servlet-mapping>
</web-app>
```

If we use annotations for the Servlet then we have to prepare the Servlet class like below.

```
@WebServlet(“/login”)
public class LoginServlet extends HttpServlet{
      —----
}
```

If we want to use annotations in Java applications then we have to use the following two steps.

1. Declare annotations
2. Utilize Annotations in Java applications.

Declare Annotations:

```
public @interface AnnotationName{
      DataType memberName() [default value];
      —----
}
```

In Java applications, every annotation is an interface that must be declared with @interface keyword.

In Java applications, there is a common and default super interface for every annotation that is java.lang.annotation.Annotation .

Utilize Annotations in Java applications:
—--
IN Java applications, we use annotations for almost all the programing elements like variables, methods, constructors, classes, abstract classes, interfaces, local variables,.....

Syntax:
@AnnotationName(member1=val1, member2=val2,....)
—-----

EX:
@Table(name=”student”)
class Student{
}

EX:
class Employee{
	@Bank(name=”ICICI Bank”, branch=”AMeerpet”)
	private Account account;
	—----
}

EX:
class Student{
	—----
	@Institute(name=”Durgasoft”, location=”Hyd”)
	public Course getCourse(){
	—---
	}
}

On the basis of the number of members in the annotations , annotations are divided into the following three types.

1. Marker Annotations
2. Single Member Annotations
3. Multi Member Annotations

Marker Annotations:
These annotations do not have members.

EX: @Override, @Entity,....

Single Member Annotations:
These annotations have only one member.
EX: @SuppressWarnings(value="checked")

Multi Member Annotations:
These annotations have more than one member.
EX: @Table(name="emp1", mutable=true)

Java has provided the following two types of annotations as per their nature.

1. Standard Annotations / Predefined Annotations
2. User Defined Annotations / Custom Annotations

Standard Annotations / Predefined Annotations:
—----------
These Annotations are defined by the Java programming Language.

There are two types of Predefined Annotations.
   1. General Purpose Annotations
   2. Meta Annotations

General Purpose Annotations:
—----------
These annotations are very much frequent and very much common in Java applications.
   1. @Override
   2. @Deprecated
   3. @SuppressWarnings
   4. @FuncationalInterface

All the General Purpose annotations are defined in by JAVA in java.lang package.

Meta Annotations:
—----------
These annotations are used to prepare other annotations.
   1. @Inherited
   2. @Documented
   3. @Target

4. @Retention

These Meta Annotations are provided by JAVA in java.lang.annotation package.

1. @Override:

—--------------

In Method Overriding , we have to provide replacement for the superclass method functionality with the subclass method functionality.

In Method Overriding, we must provide the same prototype for both the methods.

In general, in method overriding super class method name and subclass method name must be same , if we provide a mistake in the subclass method name in method overriding then the Compiler will not raise any error and the JVM will not raise any exception, in this case if we access superclass method then we are able to get output from superclass method only, not from the subclass method.

In the above context, in the method overriding, when we provide a mistake in the subclass method name then we want to get an error message , for this we have to use @Override annotation just before the subclass method.

If we provide @Override annotation to a subclass method then the @Override annotation will read subclass method name and it will compare with all methods of the superclass, if any superclass method name is matched with the subclass method name then compiler will not raise any error, if no superclass method name is matched with the subclass method name then the Compiler will raise an error.

EX:

```
class DBDriver{
	public void getDriver(){
		System.out.println("Type-1 Driver Provided By SUN Microsystems");
	}
}
class OracleDBDriver extends DBDriver{
	@Override
	public void getdriver(){
		System.out.println("Type-4 Driver Provided by Oracle Corp.");
	}
}
public class Test{
	public static void main(String[] args) {
```

```
            DBDriver dbDriver = new OracleDBDriver();
            dbDriver.getDriver();
      }
}

D:\java6>javac Test.java
Test.java:7: error: method does not override or implement a method from a
supertype
        @Override
        ^
1 error

D:\java6>

EX:
class DBDriver{
      public void getDriver(){
            System.out.println("Type-1 Driver Provided By SUN Microsystems");
      }
}
class OracleDBDriver extends DBDriver{
      @Override
      public void getDriver(){
            System.out.println("Type-4 Driver Provided by Oracle Corp.");
      }
}
public class Test{
      public static void main(String[] args) {
            DBDriver dbDriver = new OracleDBDriver();
            dbDriver.getDriver();
      }
}


D:\java6>javac Test.java

D:\java6>java Test
Type-4 Driver Provided by Oracle Corp.

D:\java6>
```

2. @Deprecated
—---------------
In Java applications, some predefined methods are declared as deprecated methods that are outdated methods, if we access that outdated methods in java applications then we are able to get the deprecation warning message.

EX:
In Frame class, show() method is a deprecated method, if we use show() method in Frame class and if we compile the program then we are able to get the deprecation message.

EX:

```
import java.awt.*;
public class Test{
      public static void main(String[] args) {
            Frame frame = new Frame();
            frame.show();
            frame.setSize(500,500);
            frame.setBackground(Color.green);
            frame.setTitle("Frame Example");
      }
}
```

```
D:\java6>javac Test.java
Note: Test.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\java6>javac -Xlint:deprecation Test.java
Test.java:5: warning: [deprecation] show() in Window has been deprecated
                frame.show();
                     ^
1 warning
```

IN Java, For the show() method an alternative method was provided by Java in the form of setVisible(true/false) method, if we use this method then the compiler will not provide any deprecation message.

EX:

```
import java.awt.*;
```

```
public class Test{
	public static void main(String[] args) {
		Frame frame = new Frame();
		frame.setVisible(true);
		frame.setSize(500,500);
		frame.setBackground(Color.green);
		frame.setTitle("Frame Example");
	}
}
```

```
D:\java6>javac Test.java

D:\java6>
```

In Java applications, Java has provided a path to declare an user defined method as a deprecated method and getting a deprecation message when we access that deprecated method .

In Java applications to declare a method as a deprecated method then we have to use @Deprecated annotation just above of the method.

```
@Deprecated
void m1(){
}
EX:
class Employee{
	@Deprecated
	public void generateSalary(){
		System.out.println("Generating Salary on the basis of Basic,
Hike, TA");
	}

	public void generateSalaryNew(){
		System.out.println("Generating Salary on the basis of Basic,
Hike, TA, PF, Bonus");
	}
}
public class Test{
	public static void main(String[] args) {
		Employee emp = new Employee();
		emp.generateSalary();
	}
```

```
}


D:\java6>javac Test.java
Note: Test.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\java6>javac  -Xlint:deprecation Test.java
Test.java:14: warning: [deprecation] generateSalary() in Employee has been
deprecated
                emp.generateSalary();
                   ^
1 warning

D:\java6>
```

3. @SuppressWarnings

—----------

In Java applications, when we perform unsafe operations or unchecked operations then the compiler will provide some warning messages, here to suppress or to hide or to remove warning messages we have to use @SuppressWarnings annotation.

EX:

In Java applications, Collection objects are able to allow different types of elements, here allowing different types of elements is unsafe operation, in the case of Collections we are able to get warning messages. In this context, to provide safe operations with the Collections we have to use Generic type declaration in the Collections.

```
List list = new Arraylist();
list.add(10);
list.add(“abc”);
list.add(new Employee());
```

It allows all the types of elements, it is unsafe operation, here compiler will raise warning message

```
List<String> list = new ArrayList<String>();
list.add(“abc”);
list.add(“xyz”);
list.add(10); —> Error
list.add(new Employee()); —> Error
```

It allows only String type elements, it is safe operation, where the compiler will not provide warning messages.

EX:

```
import java.util.*;
class Bank{
      public void getCustomersList(){
            List list = new ArrayList();
            list.add("Durga");
            list.add("Nag");
            list.add("Anil");
            list.add("Ramesh");
            System.out.println(list);
      }
}
public class Test{
      public static void main(String[] args) {
            Bank bank = new Bank();
            bank.getCustomersList();
      }
}
```

```
D:\java6>javac Test.java
Note: Test.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\java6>java Test
[Durga, Nag, Anil, Ramesh]

D:\java6>javac -Xlint:unchecked Test.java
Test.java:5: warning: [unchecked] unchecked call to add(E) as a member of the
raw type List
                list.add("Durga");
                        ^
  where E is a type-variable:
    E extends Object declared in interface List
Test.java:6: warning: [unchecked] unchecked call to add(E) as a member of the
raw type List
                list.add("Nag");
                        ^
  where E is a type-variable:
    E extends Object declared in interface List
```

```
Test.java:7: warning: [unchecked] unchecked call to add(E) as a member of the
raw type List
                list.add("Anil");
                        ^
  where E is a type-variable:
    E extends Object declared in interface List
Test.java:8: warning: [unchecked] unchecked call to add(E) as a member of the
raw type List
                list.add("Ramesh");
                        ^
  where E is a type-variable:
    E extends Object declared in interface List
4 warnings

D:\java6>
```

EX:

```
import java.util.*;
class Bank{
      public void getCustomersList(){
            List<String> list = new ArrayList<String>();
            list.add("Durga");
            list.add("Nag");
            list.add("Anil");
            list.add("Ramesh");
            System.out.println(list);
      }
}
public class Test{
      public static void main(String[] args) {
            Bank bank = new Bank();
            bank.getCustomersList();
      }
}
```

```
D:\java6>javac Test.java

D:\java6>java Test
[Durga, Nag, Anil, Ramesh]

D:\java6>
```

EX:

```
import java.util.*;
class Bank{
      @SuppressWarnings("unchecked")
      public void getCustomersList(){
            List list = new ArrayList();
            list.add("Durga");
            list.add("Nag");
            list.add("Anil");
            list.add("Ramesh");
            System.out.println(list);
      }
}
public class Test{
      public static void main(String[] args) {
            Bank bank = new Bank();
            bank.getCustomersList();
      }
}
```

```
D:\java6>javac Test.java

D:\java6>java Test
[Durga, Nag, Anil, Ramesh]

D:\java6>
```

4. @FunctionalIterface :

If any interface has only one abstract method then that interface is called Functional interface.

EX: Runnable
Comparable
ActionListener

If we declare any interface with exactly one abstract method then that interface is by default functional interface, no need to use any annotation, but if we want to make an interface as functional interface explicitly then we have to use @FunctionalInterface annotation.

If we declare an interface with the @FunctionalInterface annotation then that interface will not allow more than one abstract method, if we provide more than one abstract method then the compiler will raise an error.

EX:

```java
@FunctionalInterface
interface I{
   void m1();
  // void m2(); -----> Error
   //void m3();------> Error
}
class A implements I{
   public void m1(){
       System.out.println("m1-A");
   }

}
public class Main {
   public static void main(String[] args) {
       I i = new A();
       i.m1();

   }
}
```

Meta Annotations:
—----------------
These annotations are used to define other annotations.

@Inherited
@Documented
@Target
@Retention

@Inherited:
—----------
In general, if we define inheritance relation between classes, automatically all variables and methods are inherited from superclass to subclass , but if we provide annotations at super class then that annotations are not inheritable to the subclasses.

Note: In Java applications, by default all the annotations are not inheritable.

EX:

```
@interface Persistable{
}

@Persistable
class Employee{
}

class Manager extends Employee{
}
```

In the above code, only Employee is persistable , Manager is not Persistable.

If we declare @persistable annotation with @Inherited annotation then that annotation is inherited from superclass to subclass.

EX:

```
@Inherited
@interface Persistable{
}


@Persistable
class Employee{
}

class Manager extends Employee{
}
```

In the above code, both Employee and Manager are persistable.

@Documented

—----------

In general, if we prepare API documentation for any java file by using javadoc command then the API documentation will be created with all the classes, interfaces, methods, variables, constructors,.... Which we used in the java file.

In Java, by default annotations are not documentable, that is, annotations are not included in the API documentation if we create it for the java file by using javadoc command.

EX:

```
@interface Persistable{
}

@Persistable
class Employee{
}
```

If we prepare API documentation for the class Employee then all the Employee class members are included in the API documentation except @persistable annotation.

If we want to include Annotation data in the API documentation then we have to declare that annotation with @Documented Annotation.

EX:

```
@Documented
@interface Persistable{
}

@Persistable
class Employee{
}
```

If we prepare API documentation for the class Employee then all the Employee class members are included in the API documentation including @Persistable annotation.

3. @Target:

In general, annotations are applied to the programming elements like variables, methods, classes, constructors,.....

In Java applications, if we want to define the target elements for a particular annotation then we have to use @Target annotation.

Syntax:

```
@Target(ElementType.XXX)
```

Where XXX may be the constants from the ElementType enum like TYPE(class / abstract classes / interfaces), METHOD, CONSTRUCTOR, FIELD,....

EX1:

```
@Target(ElementType.TYPE)
@interface Persistable{
}
```

Here @Persistable annotation is applicable for only classes, abstract c;asses and interfaces.

EX2:

```
@Target(ElementType.METHOD)
@interface Persistable{
}
```

Here @Persistable annotation is applicable for only Methods.

EX3:

```
@Target(ElementType.FIELD, ElementType.METHOD)
@interface Persistable{
}
```

Here @Persistable annotation is applicable for only variables and Methods.

4. @Retention :

The main purpose of this annotation is to define lifetime to the annotations like up to .java file or up to .class file or up to Runtime.

Syntax: @Retention(RetentionPolicy.XXX)

Where XXX may have the constants like SOURCE , CLASS and RUNTIME from the RetentionPolicy enum.

EX:

```
@Retention(RetentionPolicy.SOURCE)
@interface Persistable{
}
```

The above annotation is available up to Source code.

EX:

```
@Retention(RetentionPolicy.CLASS)
@interface Persistable{
}
```

The above annotation is available up to Source code and .class file.

EX:

```
@Retention(RetentionPolicy.RUNTIME)
@interface Persistable{
}
```

The above annotation is available up to Source code, .class file and up to RUNTIME of the application.

Custom Annotations or User Defined Annotations:
—--
These annotations are defined by the developers as per their application requirements.

To prepare and use user defined annotations we have to use the following steps.

1. Declare user defined Annotation.
2. Use User defined annotations in the Java applications
3. Access data from the User defined Annotations.

Declare user defined Annotation:
—------------------------------
To declare user defined annotations we have to use the following syntax.

```
[
@Inherited
@Documented
@Target(ElementType.XXX)
@Retention(RetentionPolicy.XXX)
]
@interface Annotation{
      DataType memberName() [default Value];
}
```

Use User defined annotations in the Java applications
—---
In Java applications we are able to use annotations for the variables, methods, classes, abstract classes, interfaces,..... as per the annotations target declaration.

IN Java applications, we are able to provide annotations just above of the programming elements.

If the annotation is class level then provide the annotation just above of the class declaration.

```
@Annotation(member1=val1, member2=val2,...)
public class Employee{
}
```

If the annotation is a method level annotation then provide the annotation just above of the method.

```
public class Employee{
      —--
      @Annotation(member1=val1, member2=val2,...)
      public ACcount getAccount(){
            —---
      }
}
```

If the annotation is a field level annotation then provide annotation just above of the variable.

```
public class Employee{
      @AnnotationName(member1=val1, member2=val2,...)
      public Account account;
}
```

—---
—---

Access data from the User defined Annotations:
—--
To access the data from User defined annotations , first we have to create a java.lang.Class object for the respective class.

a. If the annotation is class level annotation :
      --Get Annotation object from class:
            public Annotation getAnnotation(Class cls)
      --Access the members from the Annotation object.
            System.out.println(ann.memberName());
b. If the annotation is a Field level annotation:
      –-Get Field object from the Class object
            public Field getField(String name)
      --Get Annotation Object from the Field object

```
            public Annotation getAnnotation(Class cls)
      --Access the members from the Annotation object.
            System.out.println(ann.memberName());

c. If the annotation is a method level annotation:
      --Create Method object
            public Method getMethod(String name)
      --Get Annotation Object
            public Annotation getAnnotation(Class cls)
      --Access the members from the Annotation object.
            System.out.println(ann.memberName());
-----
-----
```

EX:

INstitute.java

```java
package com.durgasoft.annotations;

import java.lang.annotation.*;

@Inherited
@Documented
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Institute {
   String name() default "Durgasoft";
   String branch() default "Ameerpet";
   String website() default "www.durgasoft.com";
   String phone() default "040-123456";
}
```

Student.java

```java
package com.durgasoft.entities;

import com.durgasoft.annotations.Institute;

@Institute(
       name="Durga Software Solutions",
       branch = "S R Nagar",
```

```
        phone = "040-987123"
)
public class Student {
   private String sid;
   private String sname;
   private String saddr;

   public Student(String sid, String sname, String saddr)
{
        this.sid = sid;
        this.sname = sname;
        this.saddr = saddr;
   }

   public void getStudentDetails(){
        System.out.println("Student Details");
        System.out.println("----------------------");
        System.out.println("Student Id       : "+sid);
        System.out.println("Student Name     : "+sname);
        System.out.println("Student Address  : "+saddr);
   }
}
```

Main.java

```
import com.durgasoft.annotations.Institute;
import com.durgasoft.entities.Student;

import java.lang.annotation.Annotation;

public class Main {
   public static void main(String[] args) {
        Student student = new Student("S-111", "Durga",
"Hyd");
        student.getStudentDetails();
        System.out.println();

        Class cls = student.getClass();
```

```
        Annotation annotation =
cls.getAnnotation(Institute.class);
        Institute institute = (Institute) annotation;
        System.out.println("Institute Details");
        System.out.println("------------------------");
        System.out.println("Name      :
"+institute.name());
        System.out.println("Branch    :
"+institute.branch());
        System.out.println("Website   :
"+institute.website());
        System.out.println("Phone     :
"+institute.phone());
    }
}
```

EX:

Account.java

```
package com.durgasoft.entities;

public class Account {
    private String accNo;
    private String accHolderName;
    private String accType;
    private int balance;

    public Account(String accNo, String accHolderName,
String accType, int balance) {
        this.accNo = accNo;
        this.accHolderName = accHolderName;
        this.accType = accType;
        this.balance = balance;
    }

    public void getAccountDetails(){
        System.out.println("Account Details");
        System.out.println("----------------------");
```

```
        System.out.println("Account Number       :
"+accNo);
        System.out.println("Account Holder Name  :
"+accHolderName);
        System.out.println("Account Type         :
"+accType);
        System.out.println("Account Balance      :
"+balance);
    }
}
```

Employee.java

```
package com.durgasoft.entities;

import com.durgasoft.annotations.Bank;

public class Employee {
    private int eno;
    private String ename;
    private float esal;
    private String eaddr;

    @Bank(
            name="AXIS Bank",
            phone = "040-99887766"
    )
    private Account account;

    public Employee(int eno, String ename, float esal,
String eaddr, Account account) {
        this.eno = eno;
        this.ename = ename;
        this.esal = esal;
        this.eaddr = eaddr;
        this.account = account;
    }
```

```
    public void getEmployeeDetails(){
        System.out.println("Employee Details");
        System.out.println("--------------------");
        System.out.println("Employee Number    : "+eno);
        System.out.println("Employee Name      : "+ename);
        System.out.println("Employee Salary    : "+esal);
        System.out.println("Employee Address   : "+eaddr);
        System.out.println();
        account.getAccountDetails();
    }
}
```

Bank.java

```
package com.durgasoft.annotations;

import java.lang.annotation.*;

@Inherited
@Documented
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Bank {
    String name() default "ICICI Bank";
    String branch() default "Ameerpet";
    String website() default "www.icicibank.com";
    String phone() default "040-12345566";

}
```

Main.java

```
import com.durgasoft.annotations.Bank;
import com.durgasoft.entities.Account;
import com.durgasoft.entities.Employee;

import java.lang.reflect.Field;

public class Main {
    public static void main(String[] args)throws Exception {
        Account account = new Account("abc123", "Durga", "Savings", 50000);
        Employee employee = new Employee(111, "Durga", 5000, "Hyd", account);
```

```
        employee.getEmployeeDetails();
        System.out.println();

        Class cls = employee.getClass();
        Field field = cls.getDeclaredField("account");
        field.setAccessible(true);
        Bank bank = field.getAnnotation(Bank.class);
        System.out.println("Bank Details");
        System.out.println("-------------------");
        System.out.println("Bank Name     : "+bank.name() );
        System.out.println("Bank Branch   : "+bank.branch());
        System.out.println("Bank Website  : "+bank.branch());
        System.out.println("Bank Phone    : "+bank.phone());
    }
}
```

```
Employee Details
--------------------
Employee Number    : 111
Employee Name      : Durga
Employee Salary    : 5000.0
Employee Address   : Hyd

Account Details
-----------------------
Account Number       : abc123
Account Holder Name  : Durga
Account Type         : Savings
Account Balance      : 50000

Bank Details
-------------------
Bank Name     : AXIS Bank
Bank Branch   : Ameerpet
Bank Website  : Ameerpet
Bank Phone    : 040-99887766
```

EX:

Course.java

```
package com.durgasoft.app44.annotations;

import java.lang.annotation.*;
```

```java
@Inherited
@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Course {
   String cid();
   String cname();
   int cfee();

}
```

Student.java

```java
package com.durgasoft.app44.entities;

import com.durgasoft.app44.annotations.Course;

public class Student {
   private String sid;
   private String sname;
   private String saddr;

   public Student(String sid, String sname, String saddr)
{
        this.sid = sid;
        this.sname = sname;
        this.saddr = saddr;
   }

   @Course(cid="C-111", cname="Fullstack Java",
cfee=30000)
   public void getStudentDetails(){
        System.out.println("Student Details");
        System.out.println("-------------------------");
        System.out.println("Student Id       : "+sid);
        System.out.println("Student Name     : "+sname);
        System.out.println("Student Address  : "+saddr);
   }
```

```
}
```

Main.java

```
import com.durgasoft.app44.annotations.Course;
import com.durgasoft.app44.entities.Student;

import java.lang.reflect.Method;

public class Main {
   public static void main(String[] args) throws
Exception {
       Student student = new Student("S-111", "Durga",
"Hyd");
       student.getStudentDetails();
       System.out.println();

       Class cls = student.getClass();
       Method method =
cls.getDeclaredMethod("getStudentDetails");
       Course course =
method.getAnnotation(Course.class);
       System.out.println("Course Details");
       System.out.println("---------------------");
       System.out.println("Course Id     :
"+course.cid());
       System.out.println("Course Name   :
"+course.cname());
       System.out.println("Course Cost   :
"+course.cfee());

   }
}
```

```
Student Details
------------------------
Student Id       : S-111
Student Name     : Durga
```

```
Student Address  : Hyd

Course Details
----------------------
Course Id     : C-111
Course Name   : Fullstack Java
Course Cost   : 30000
```