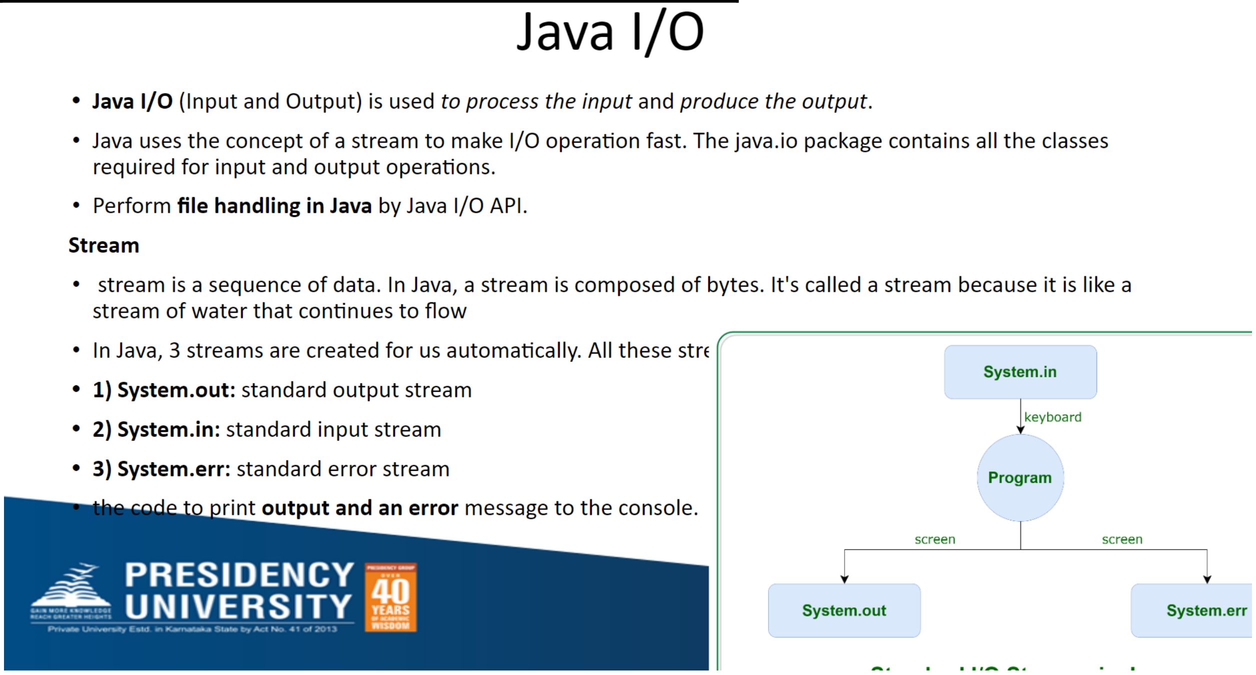
Define stream.	
List three streams for java IO.	

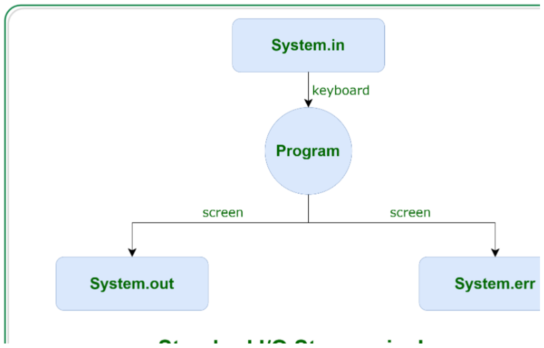


Java I/O

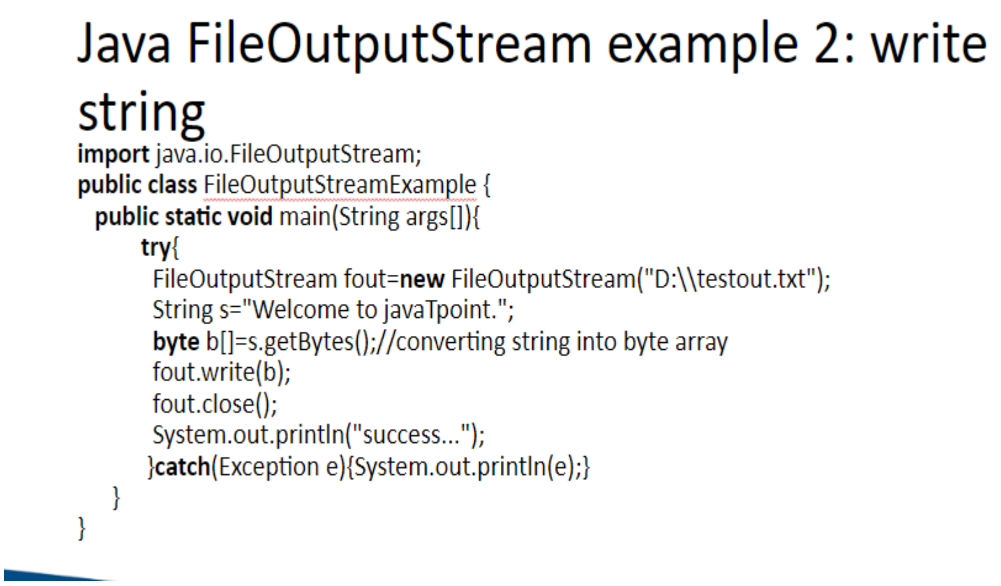
- **Java I/O** (Input and Output) is used *to process the input* and *produce the output*.
- Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- Perform **file handling in Java** by Java I/O API.

Stream

- stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow
- In Java, 3 streams are created for us automatically. All these stre
- **1) System.out:** standard output stream
- **2) System.in:** standard input stream
- **3) System.err:** standard error stream
- the code to print **output and an error** message to the console.

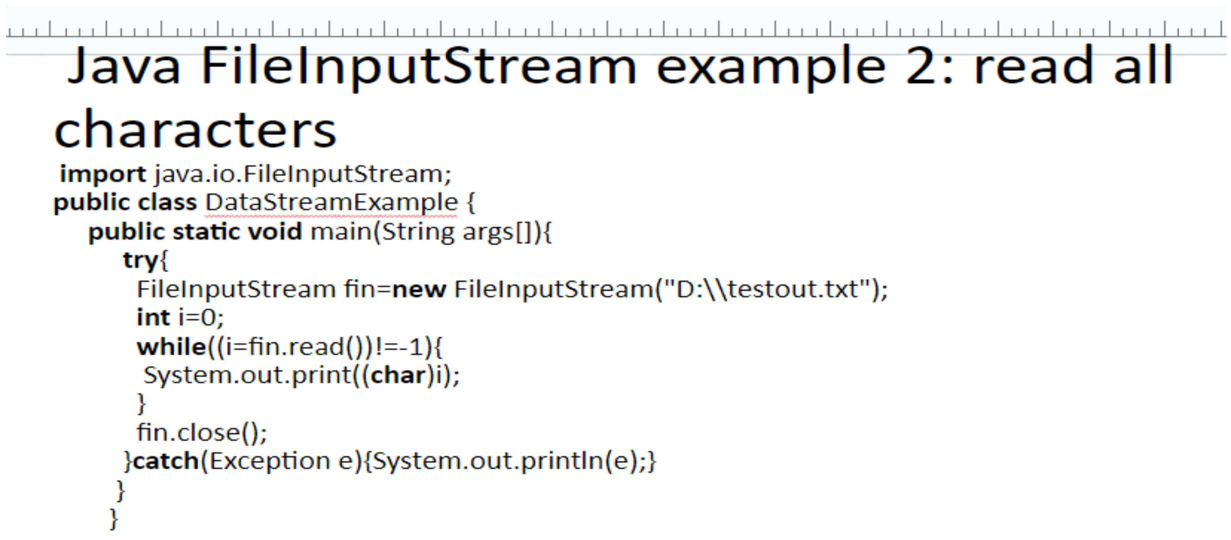


3	Review a java program to read/write a text file.



Java FileOutputStream example 2: write string

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
  public static void main(String args[]){
        try{
          FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
          String s="Welcome to javaTpoint.";
          byte b[]=s.getBytes();//converting string into byte array
          fout.write(b);
          fout.close();
          System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
```



Java FileInputStream example 2: read all characters

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=0;
          while((i=fin.read())!=-1){
           System.out.print((char)i);
          }
          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

4	Define serialization and deserialization.

Serialization and Deserialization

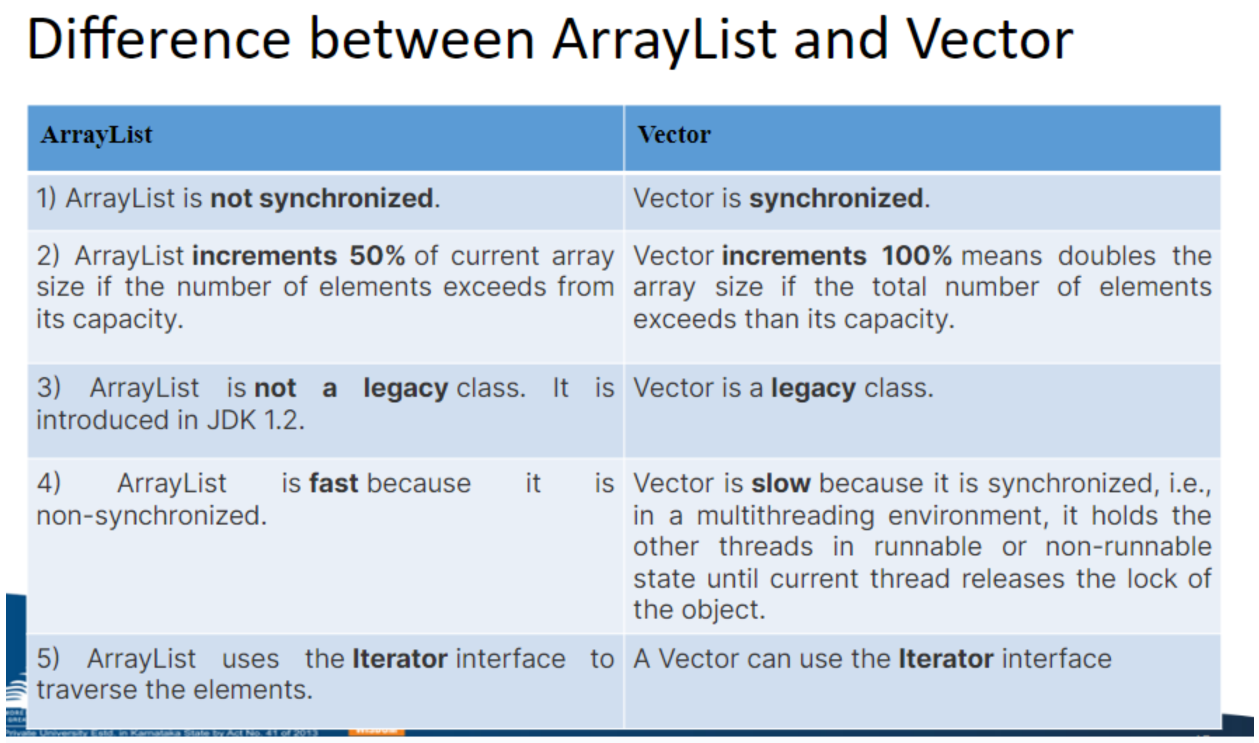
- **Serialization in Java** is a mechanism of *writing the state of an object into a byte-stream*. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.
- The reverse operation of serialization is called *deserialization* where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

5	List advantages of generics.

Advantage of Java Generics

1. **Type-safety:** We can hold only a single type of objects in generics. It doesnot allow to store other objects.
2. **Type casting is not required:** There is no need to typecast the object.
3. **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

6	Distinguish between ArrayList and Vector.

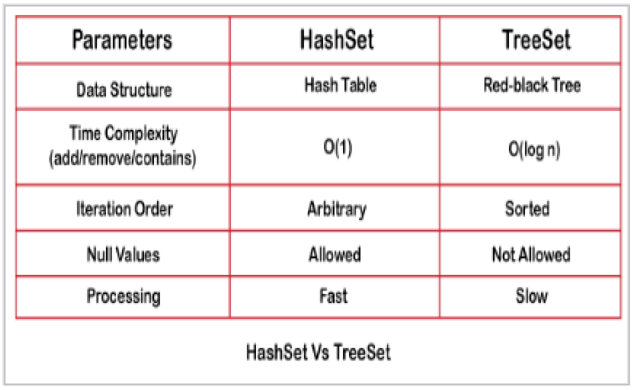


Difference between ArrayList and Vector

ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity.	Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
5) ArrayList uses the **Iterator** interface to traverse the elements.	A Vector can use the **Iterator** interface

7	Difference and similarities between HashSet and TreeSet.

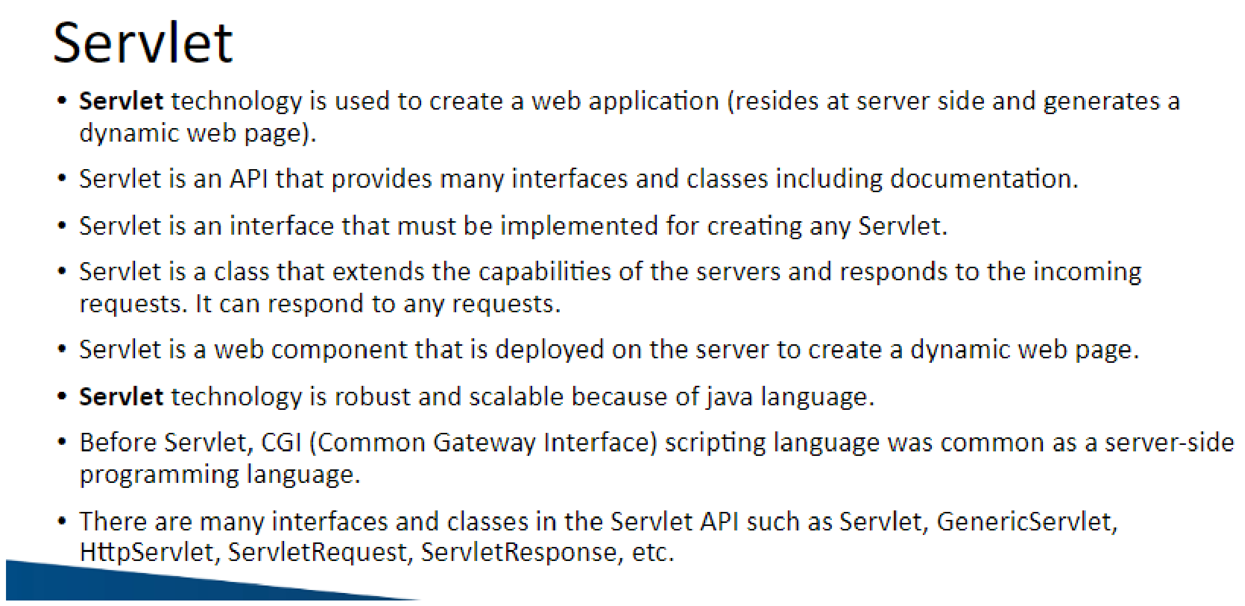
differences and similarities between HashSet and TreeSet.



Parameters	HashSet	TreeSet
Data Structure	Hash Table	Red-black Tree
Time Complexity (add/remove/contains)	O(1)	O(log n)
Iteration Order	Arbitrary	Sorted
Null Values	Allowed	Not Allowed
Processing	Fast	Slow

HashSet Vs TreeSet

15	Define Servlet.



Servlet

- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.
- **Servlet** technology is robust and scalable because of java language.
- Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

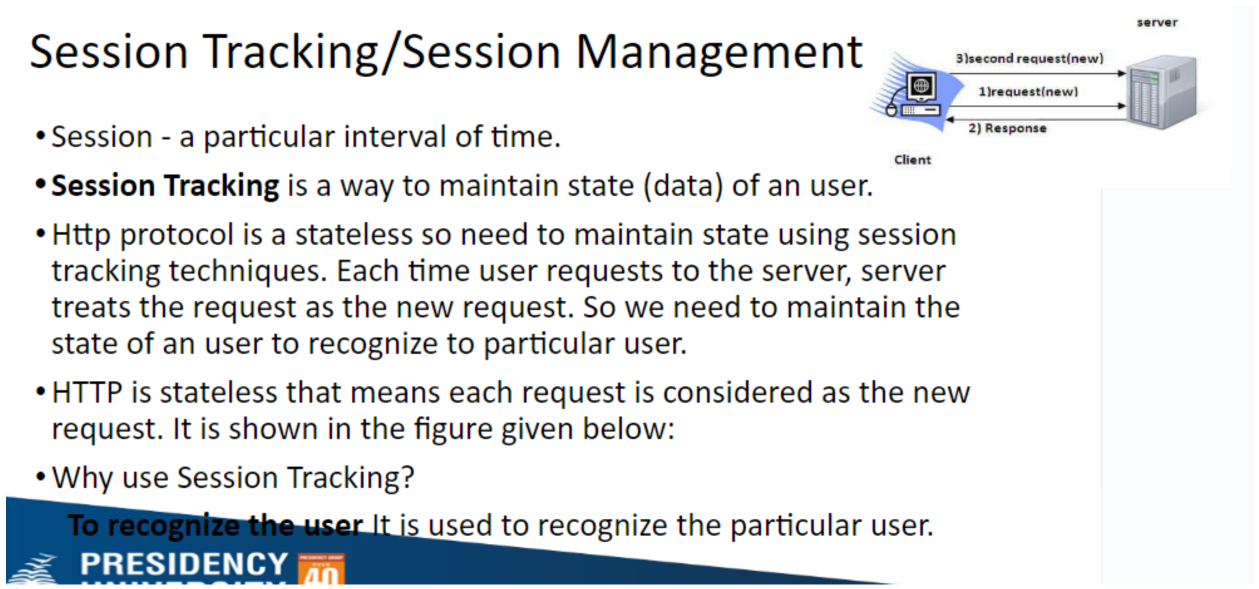
16	List the advantages of servlet

Advantages of Servlet

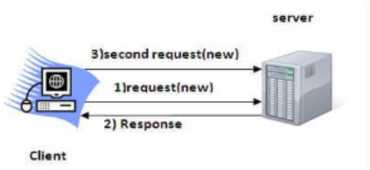
- There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

17	Define session and cookies

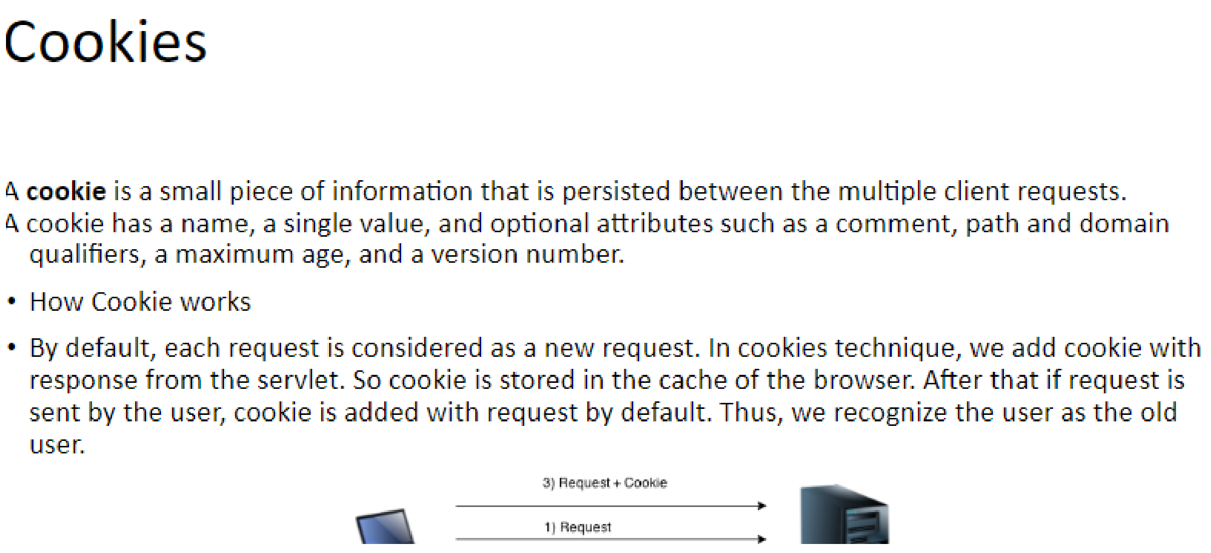


Session Tracking/Session Management



- Session - a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user.
- Http protocol is a stateless so need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:
- Why use Session Tracking?

  To recognize the user It is used to recognize the particular user.



Cookies

A **cookie** is a small piece of information that is persisted between the multiple client requests.
A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

- How Cookie works
- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

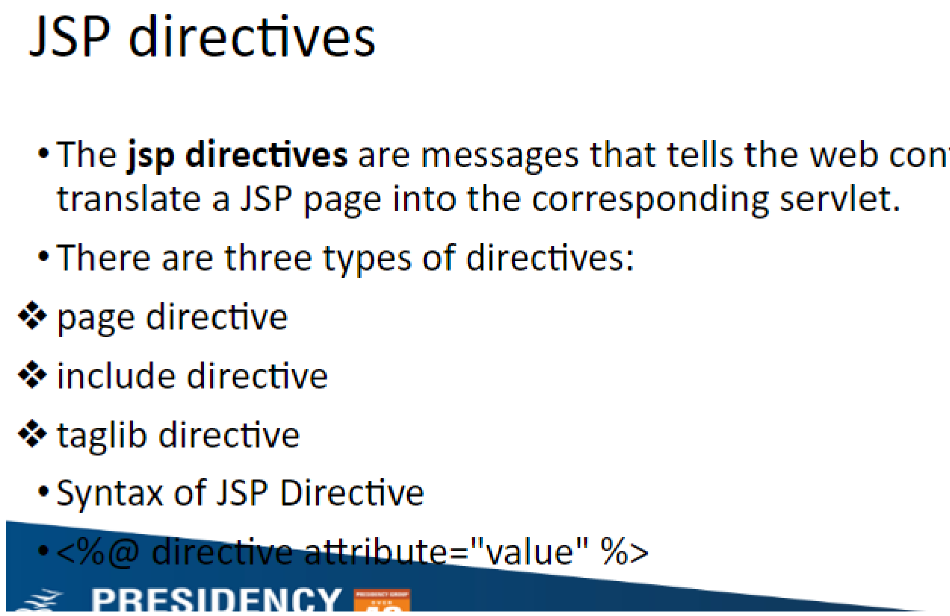
19	List the three types of scripting elements in JSP

JSP Scripting elements

- The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

➢ scriptlet tag
➢ expression tag
➢ declaration tag

20	List out the JSP directives



JSP directives

- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- There are three types of directives:

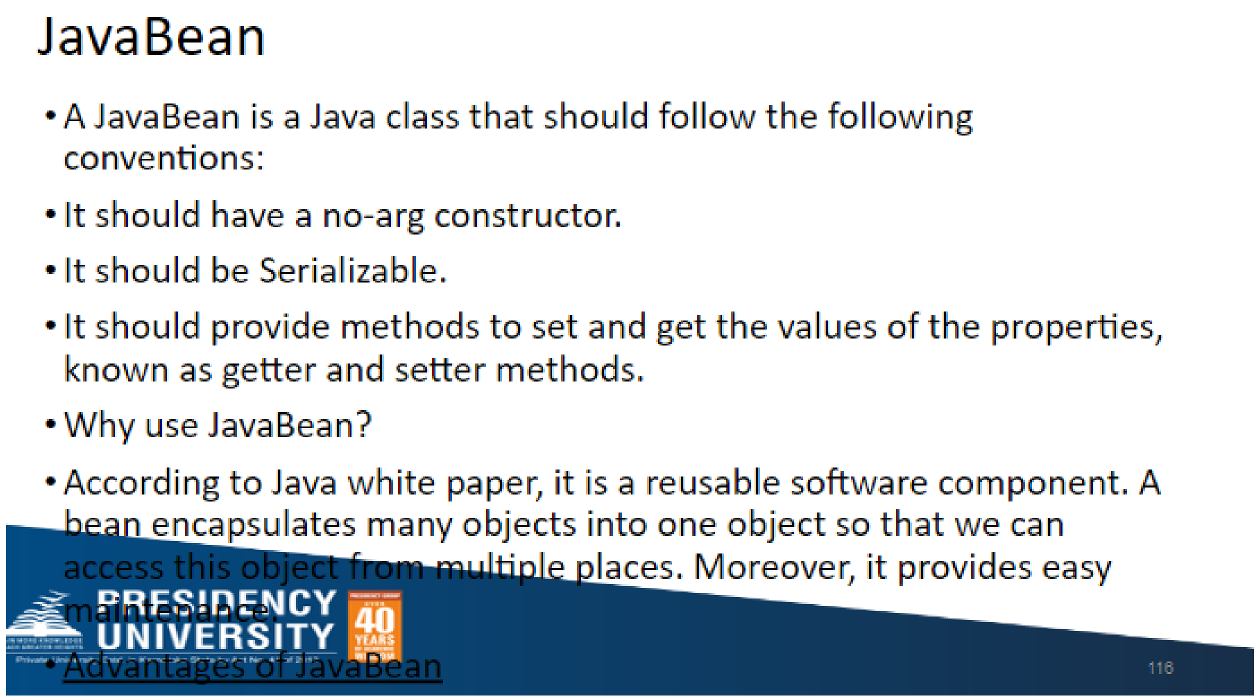
❖ page directive

❖ include directive

❖ taglib directive

- Syntax of JSP Directive
- <%@ directive attribute="value" %>

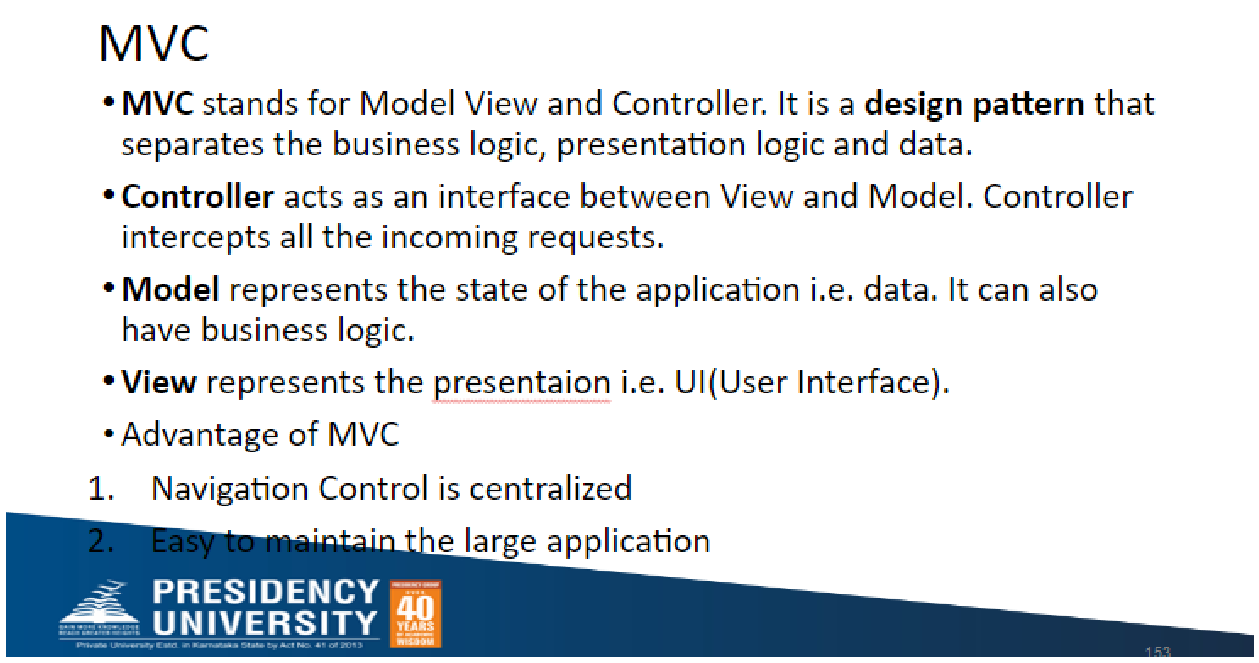
21	Define java bean



JavaBean

- A JavaBean is a Java class that should follow the following conventions:
- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.
- Why use JavaBean?
- According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.
- Advantages of JavaBean

24	Outline the advantages of MVC.



MVC

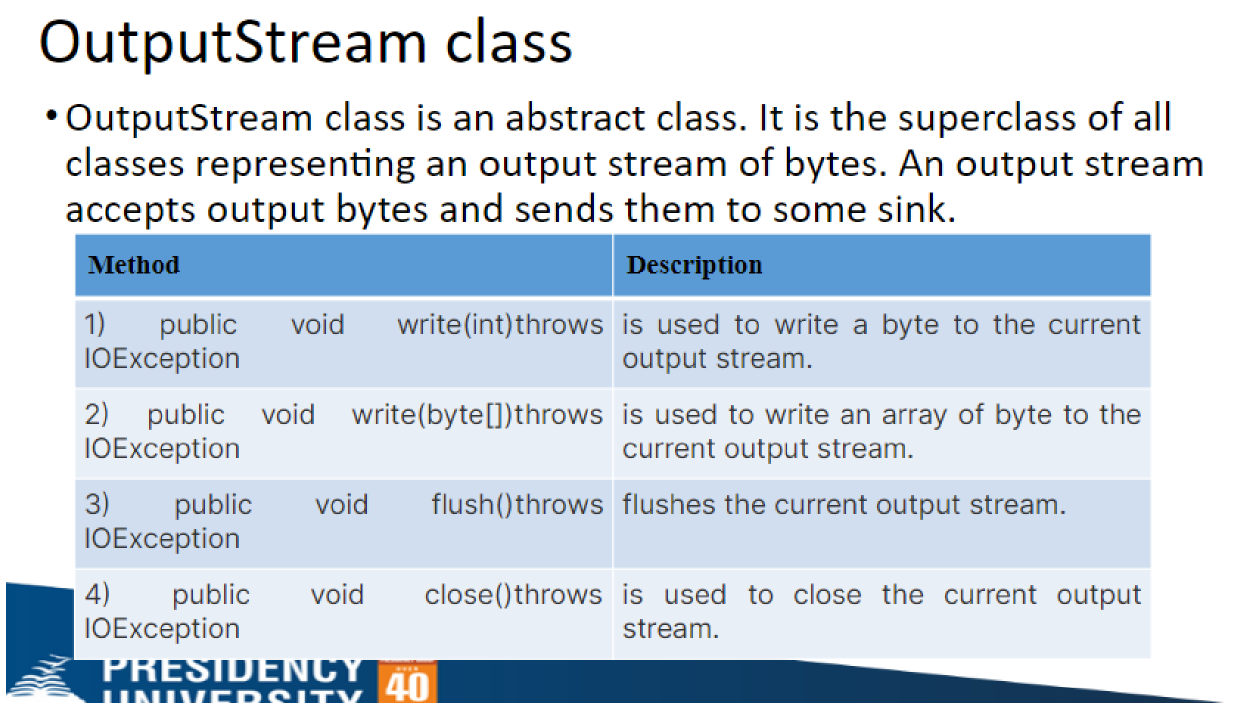
- **MVC** stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.
- **Controller** acts as an interface between View and Model. Controller intercepts all the incoming requests.
- **Model** represents the state of the application i.e. data. It can also have business logic.
- **View** represents the presentaion i.e. UI(User Interface).
- Advantage of MVC

1. Navigation Control is centralized
2. Easy to maintain the large application

153

PART - B & C

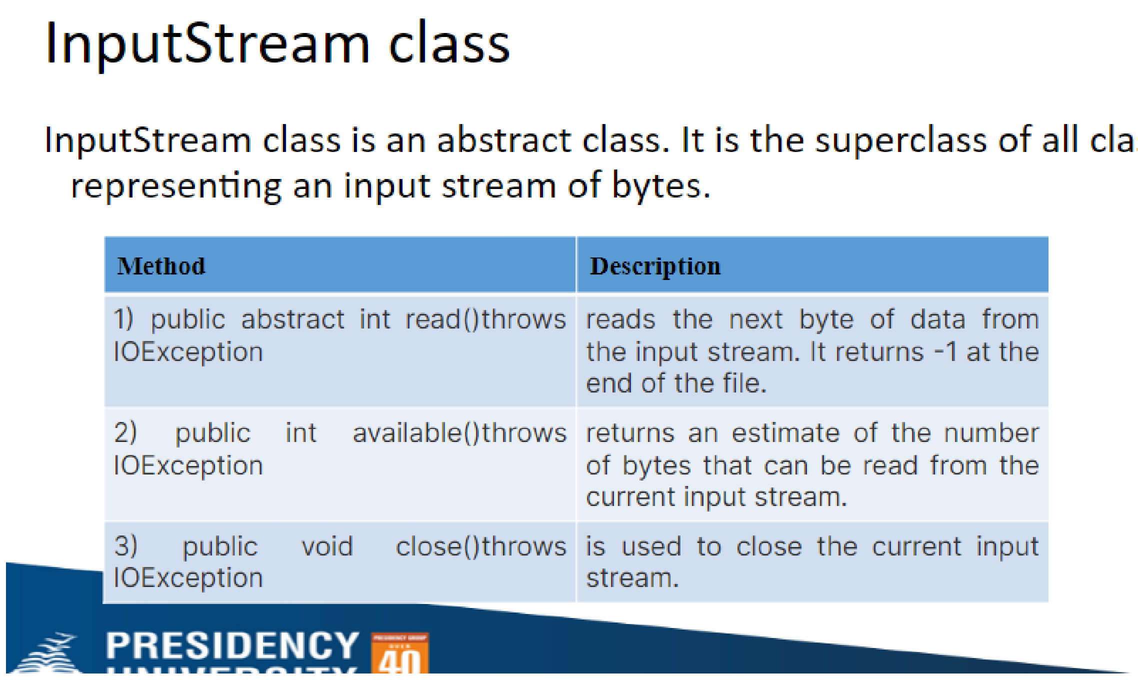
9	Describe the I/O classes in java.



OutputStream class

- OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Method	Description
1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.

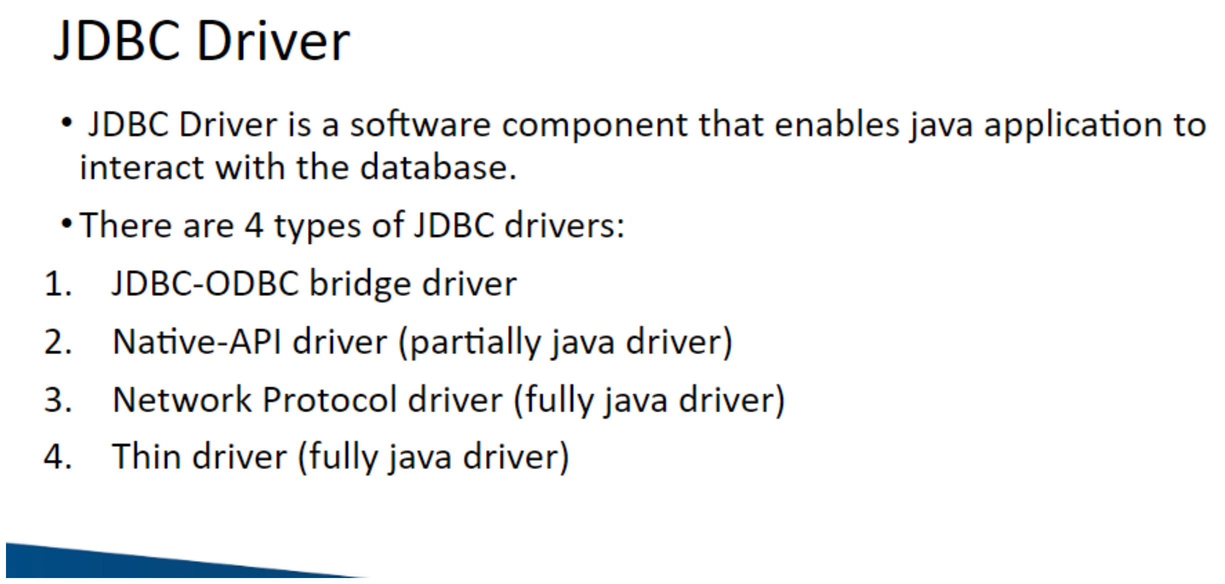


InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

10	Explain different types of JDBC drivers.



JDBC Driver

- JDBC Driver is a software component that enables java application to interact with the database.
- There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

- Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC met the ODBC function calls.

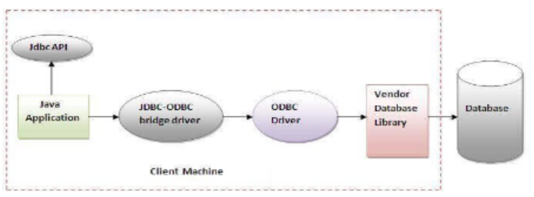


Figure-JDBC-ODBC Bridge Driver

Native-API driver

- The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.
- Advantage:
- performance upgraded than JDBC-ODBC bridge driver.
- Disadvantage:
- The Native driver needs to be installed on the each
- The Vendor client library needs to be installed on cli

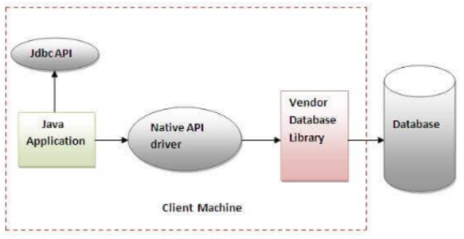
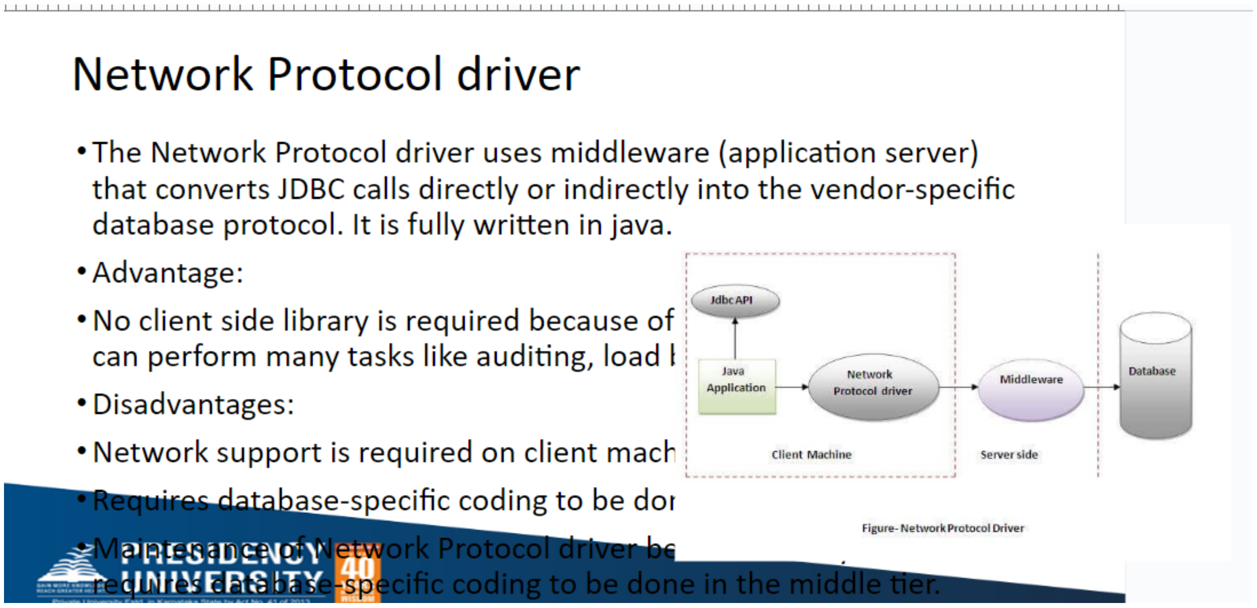


Figure- Native API Driver



Network Protocol driver

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.
- Advantage:
- No client side library is required because of
  can perform many tasks like auditing, load
- Disadvantages:
- Network support is required on client mach
- Requires database-specific coding to be dor
- Maintenance of Network Protocol driver be
  requires database-specific coding to be done in the middle tier.

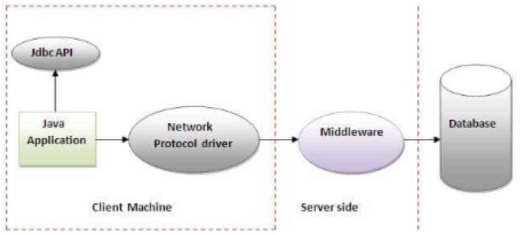
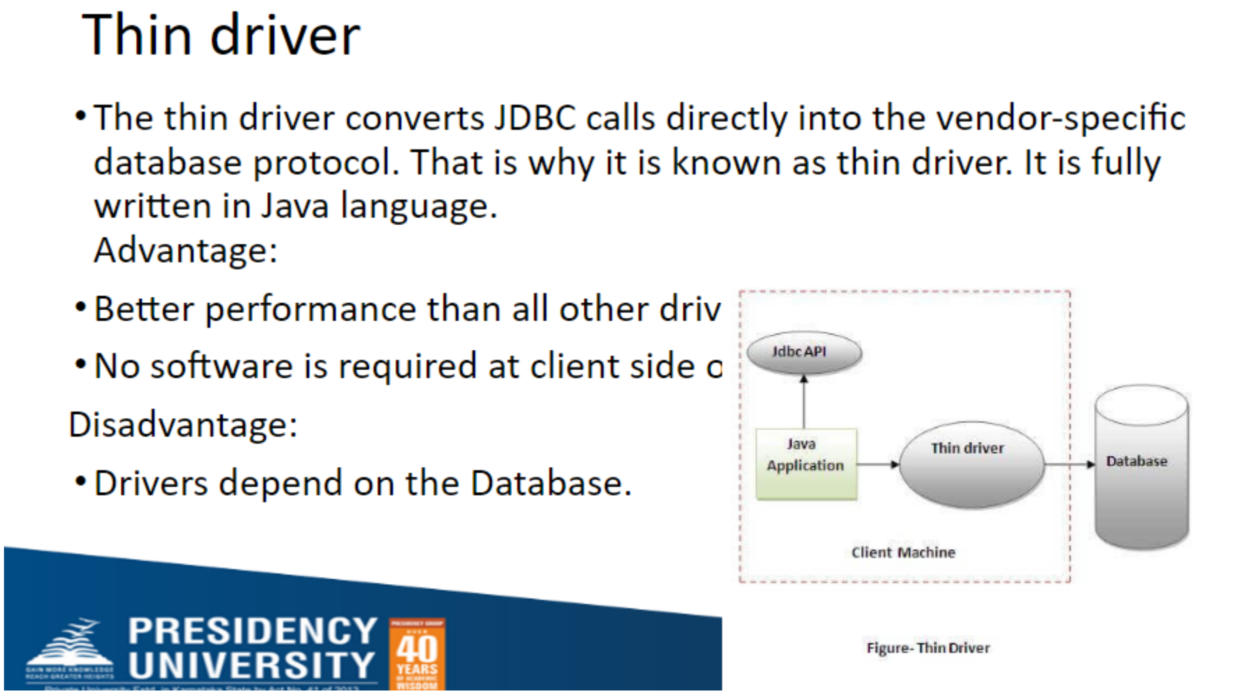


Figure- Network Protocol Driver



Thin driver

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.
  Advantage:
- Better performance than all other driv
- No software is required at client side o

Disadvantage:

- Drivers depend on the Database.

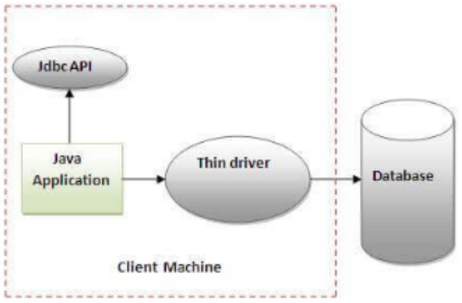


Figure- Thin Driver

11	Demonstrate Generic method and class with example.

Generic class

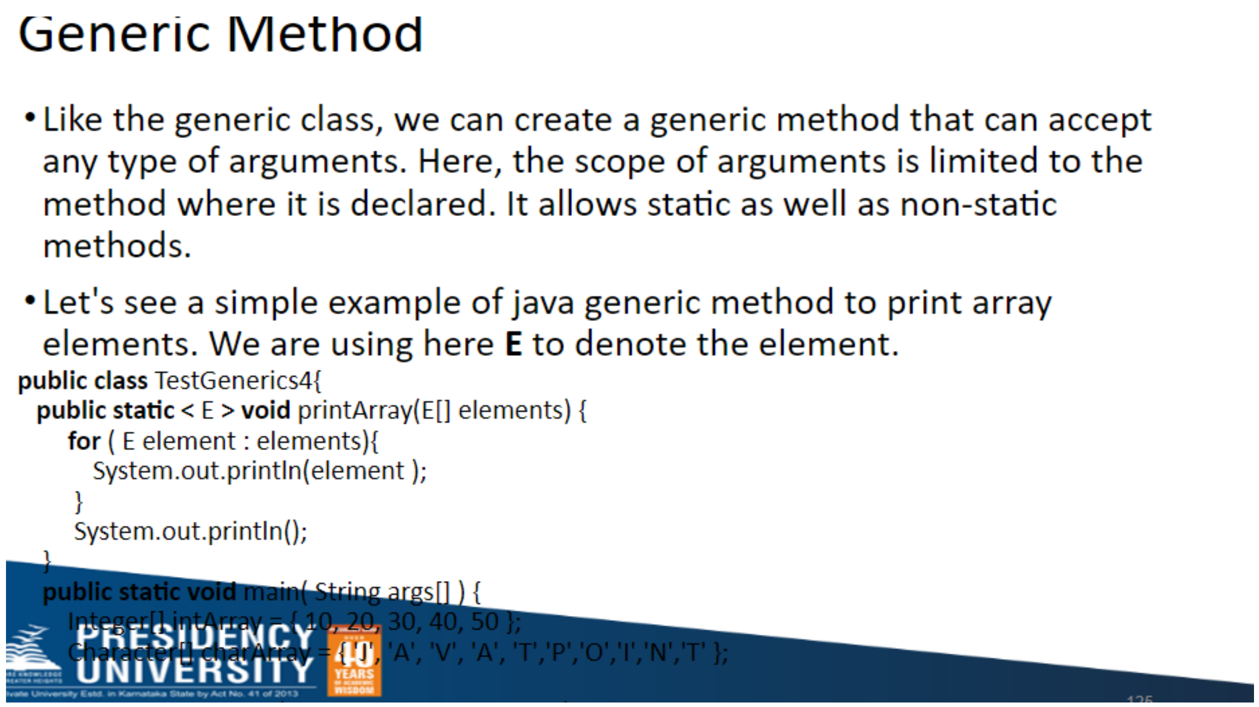
- A class that can refer to any type is known as a generic class. Here, we are using the T type parameter to create the generic class of specific type.
- **Creating a generic class:**

```
class MyGen<T>{
      T obj;
      void set(T obj){this.obj=obj;}
      T get(){return obj;}
}
```

- The T type indicates that it can refer to any type (like String, Integer, and Employee). The type you specify for the class will be used to store and retrieve the data.

Using generic class

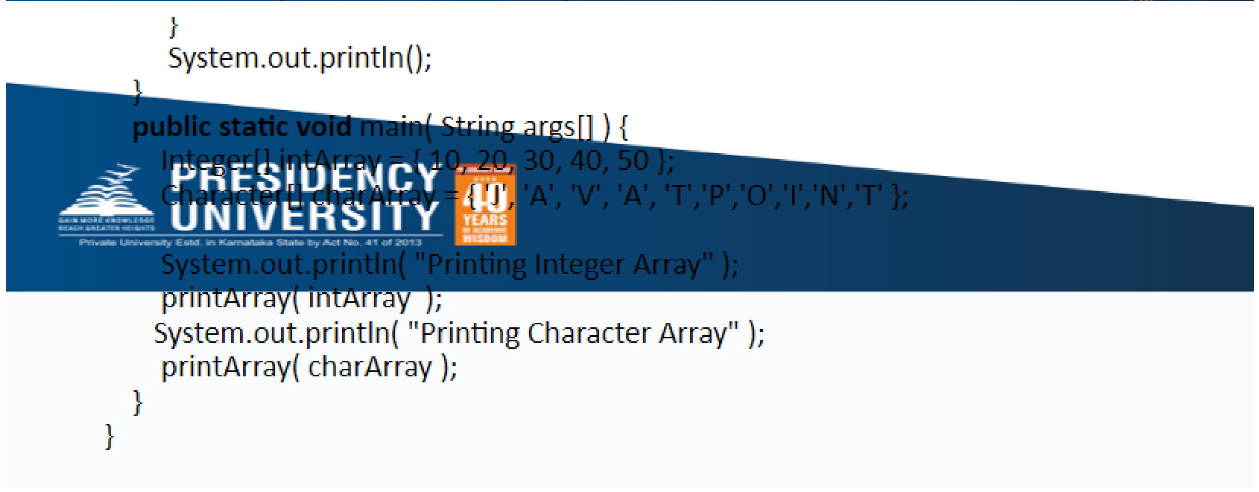
```
class TestGenerics3{
      public static void main(String args[]){
            MyGen<Integer> m=new MyGen<Integer>();
            m.add(2);
            //m.add("vivek");//Compile time error
            System.out.println(m.get());
      }
}
```



Generic Method

- Like the generic class, we can create a generic method that can accept any type of arguments. Here, the scope of arguments is limited to the method where it is declared. It allows static as well as non-static methods.
- Let's see a simple example of java generic method to print array elements. We are using here **E** to denote the element.

```
public class TestGenerics4{
  public static < E > void printArray(E[] elements) {
      for ( E element : elements){
        System.out.println(element );
      }
      System.out.println();
  }
  public static void main( String args[] ) {
      Integer[] intArray = { 10, 20, 30, 40, 50 };
      Character[] charArray = {'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };
```



```
        }
        System.out.println();
    }
    public static void main( String args[] ) {
        Integer[] intArray = { 10, 20, 30, 40, 50 };
        Character[] charArray = {'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };
```

```
        System.out.println( "Printing Integer Array" );
        printArray( intArray  );
       System.out.println( "Printing Character Array" );
        printArray( charArray );
    }
}
```

13	**Describe about Annotations and its types with example.**

Annotation

- Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.
- Annotations in Java are used to provide additional information, so it is an alternative option for XML and Java marker interfaces.

Built-In Java Annotations

- Built-In Java Annotations used in Java code

@Override

@SuppressWarnings

@Deprecated

- Built-In Java Annotations used in other annotations

@Target

@Retention

@Inherited

@Documented

@Override

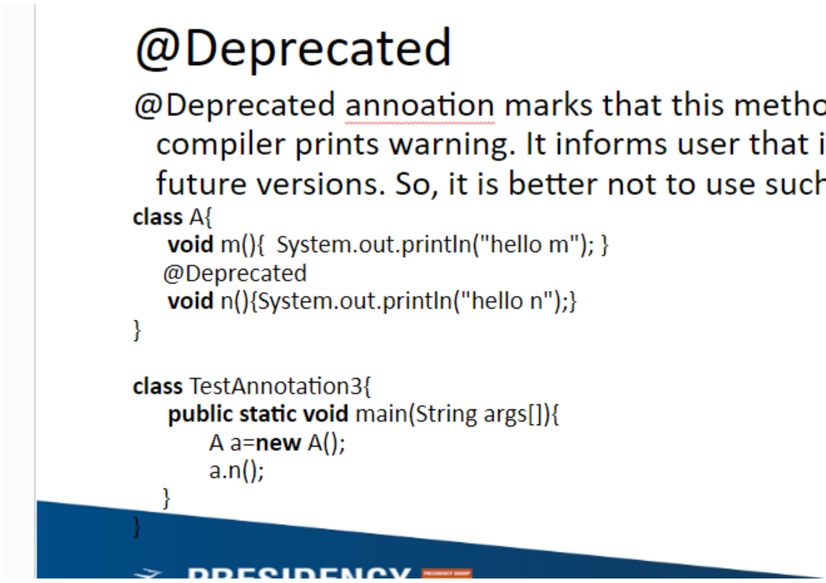
- @Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.
- Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

```
class Animal{
    void eatSomething() { System.out.println("eating something"); }
}
 class Dog extends Animal{
   @Override
   void eatsomething() { System.out.println("eating foods"); }//should be eatSomething
}
 class TestAnnotation1{
        public static void main(String args[]) {
            Animal a=new Dog();
            a.eatSomething();
       }
}
```

@SuppressWarnings

- @SuppressWarnings annotation: is used to suppress warnings issued by the compiler.
- If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.

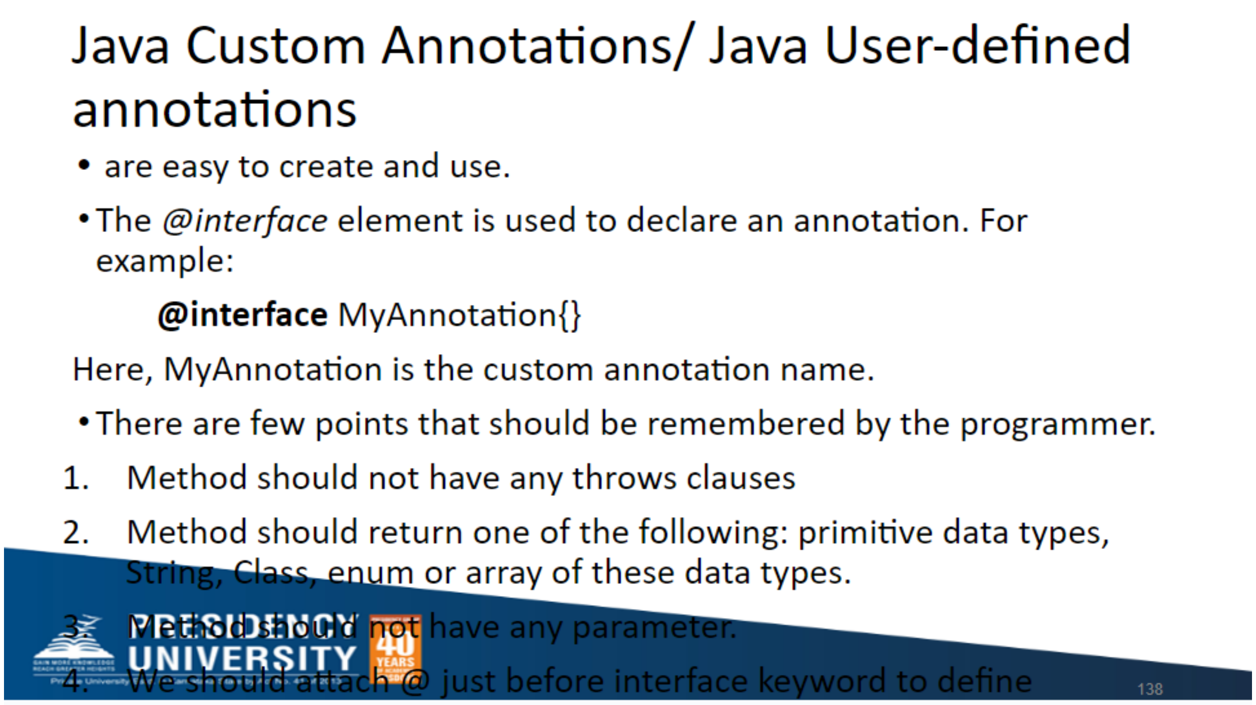
```
import java.util.*;
class TestAnnotation2{
     @SuppressWarnings("unchecked")
     public static void main(String args[]){
           ArrayList list=new ArrayList();
           list.add("sonoo");
           list.add("vimal");
           list.add("ratan");
          for(Object obj:list)
                System.out.println(obj);
```



@Deprecated

@Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

```
class A{
    void m(){  System.out.println("hello m"); }
   @Deprecated
    void n(){System.out.println("hello n");}
}

class TestAnnotation3{
    public static void main(String args[]){
         A a=new A();
         a.n();
    }
}
```



Java Custom Annotations/ Java User-defined annotations

- are easy to create and use.
- The *@interface* element is used to declare an annotation. For example:

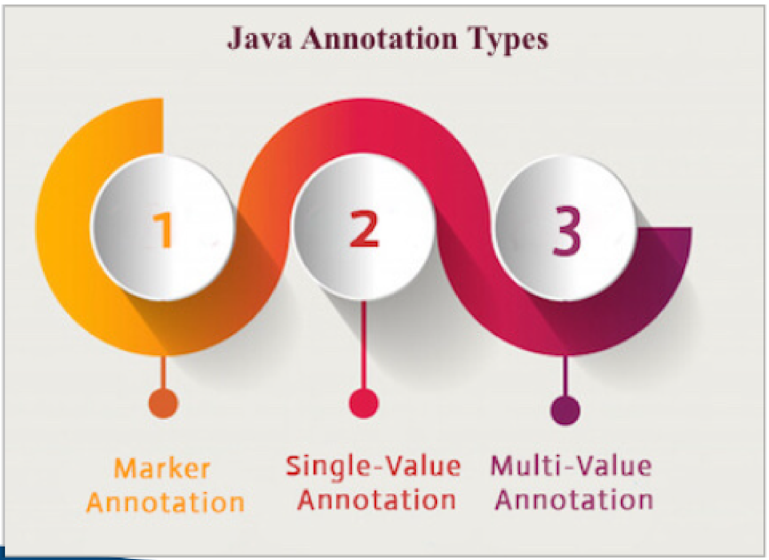
    @interface MyAnnotation{}

Here, MyAnnotation is the custom annotation name.

- There are few points that should be remembered by the programmer.

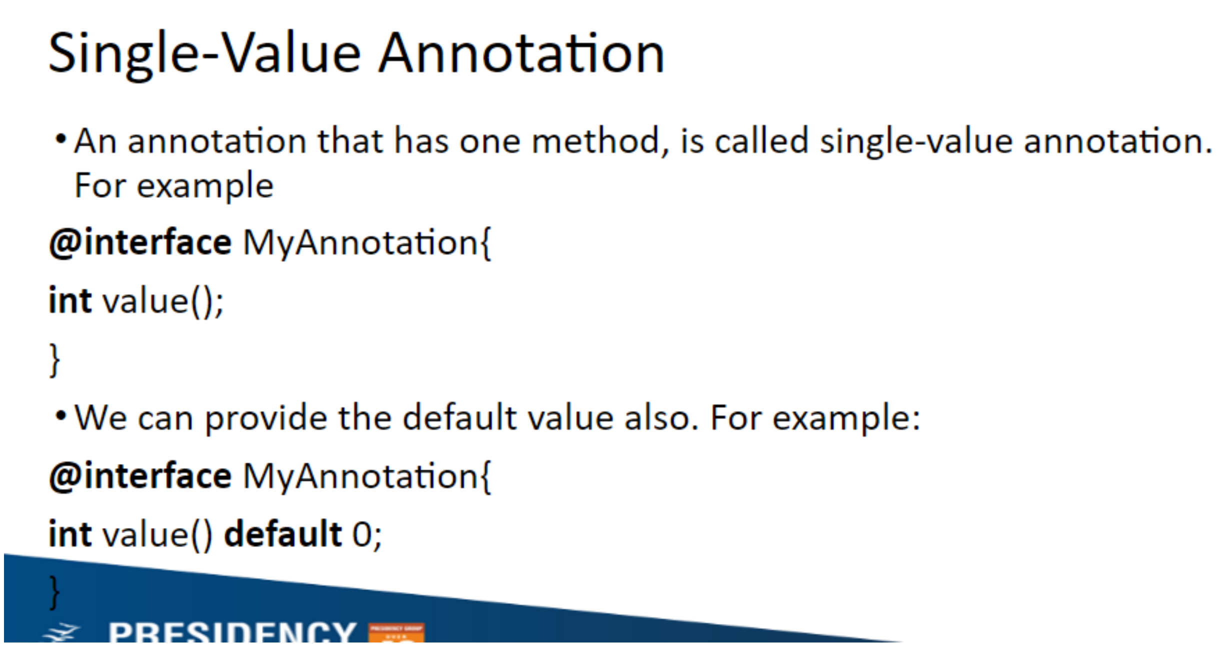
1. Method should not have any throws clauses
2. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
3. Method should not have any parameter.
4. We should attach @ just before interface keyword to define

Types of Annotation



Marker Annotation

- An annotation that has no method, is called marker annotation. For example:
- **@interface** MyAnnotation{}
- The @Override and @Deprecated are marker annotations.



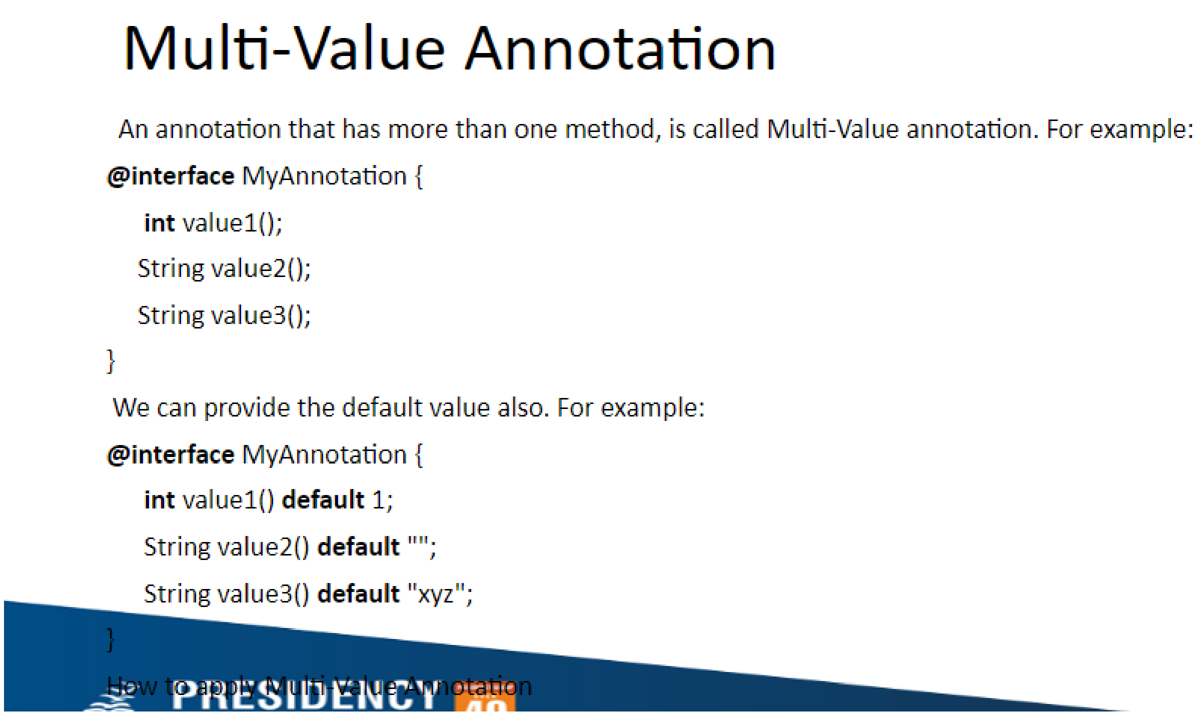
Single-Value Annotation

- An annotation that has one method, is called single-value annotation. For example

```
@interface MyAnnotation{
int value();
}
```

- We can provide the default value also. For example:

```
@interface MyAnnotation{
int value() default 0;
}
```



Multi-Value Annotation

An annotation that has more than one method, is called Multi-Value annotation. For example:

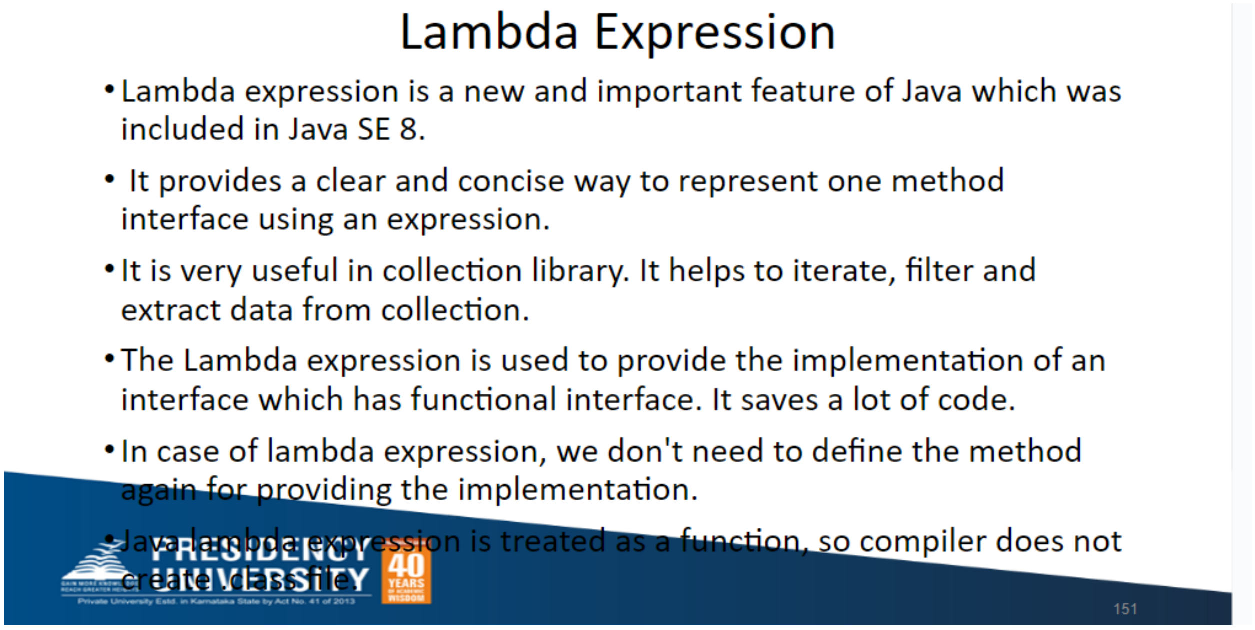
```
@interface MyAnnotation {
     int value1();
    String value2();
    String value3();
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation {
     int value1() default 1;
     String value2() default "";
     String value3() default "xyz";
}
```

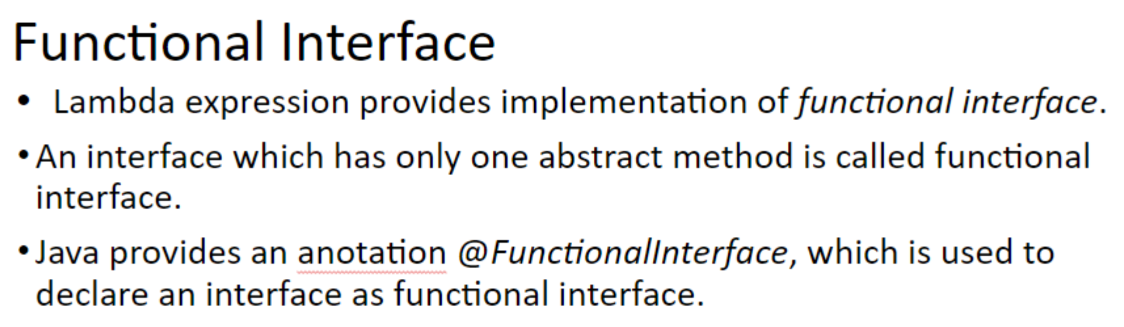
How to apply Multi-Value Annotation

14	**Explain about Lambda expression and its uses**



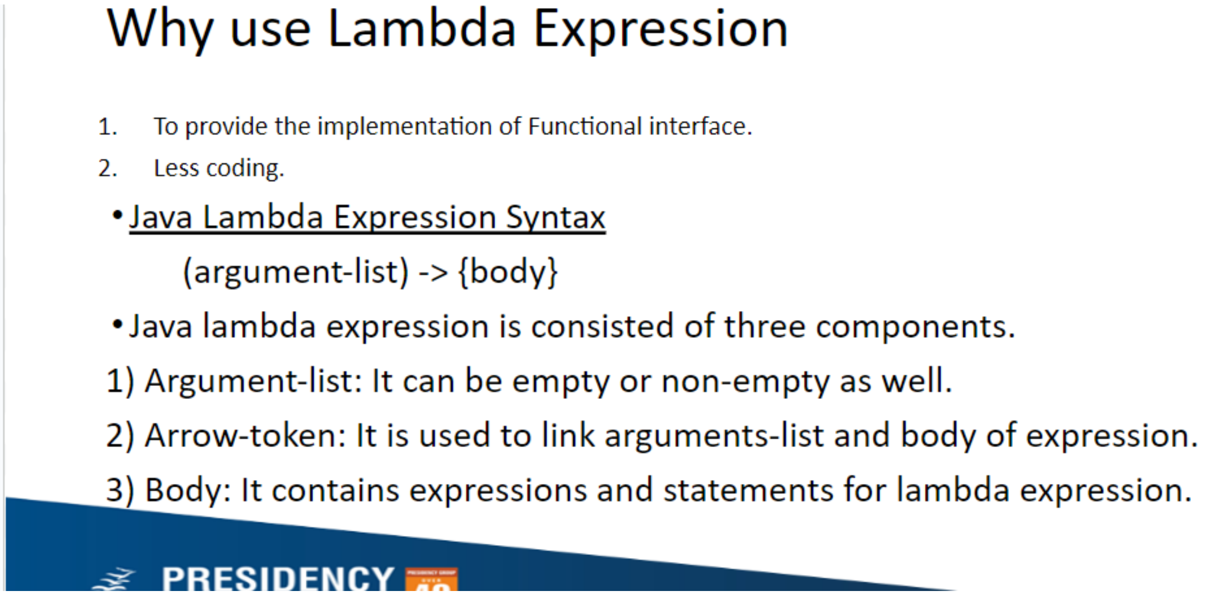
Lambda Expression

- Lambda expression is a new and important feature of Java which was included in Java SE 8.
- It provides a clear and concise way to represent one method interface using an expression.
- It is very useful in collection library. It helps to iterate, filter and extract data from collection.
- The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code.
- In case of lambda expression, we don't need to define the method again for providing the implementation.
- Java lambda expression is treated as a function, so compiler does not create .class file.



Functional Interface

- Lambda expression provides implementation of *functional interface*.
- An interface which has only one abstract method is called functional interface.
- Java provides an anotation *@FunctionalInterface*, which is used to declare an interface as functional interface.



Why use Lambda Expression

1. To provide the implementation of Functional interface.
2. Less coding.

- Java Lambda Expression Syntax

  (argument-list) -> {body}

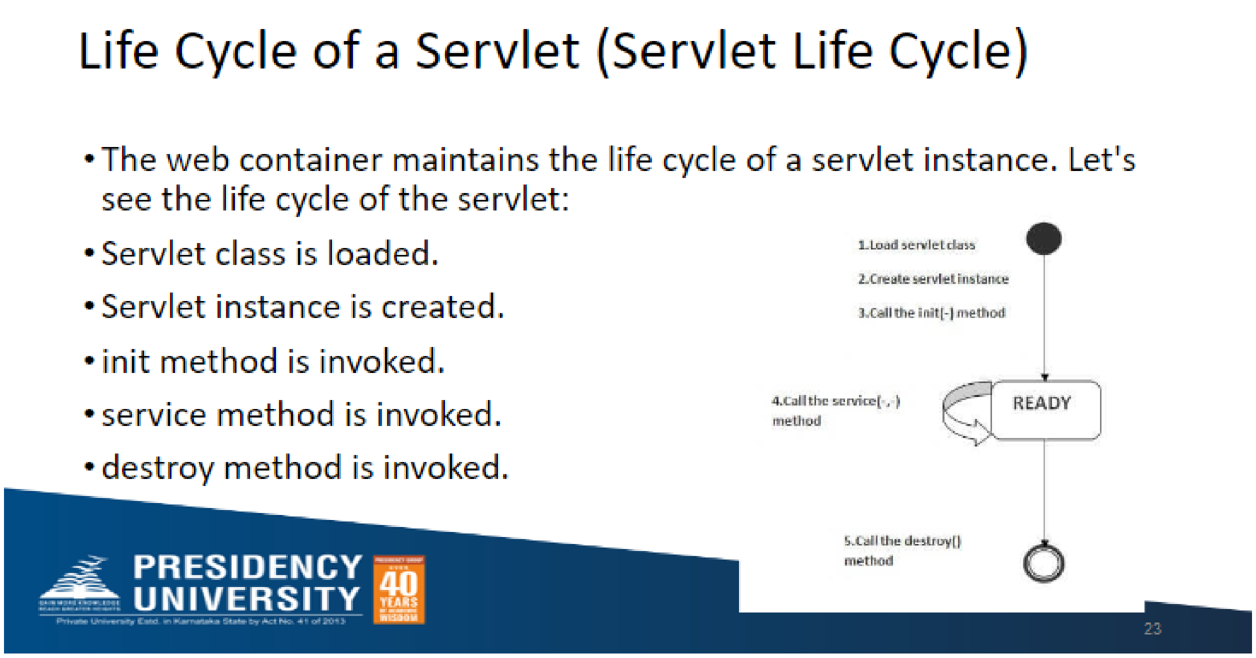
- Java lambda expression is consisted of three components.

1) Argument-list: It can be empty or non-empty as well.

2) Arrow-token: It is used to link arguments-list and body of expression.

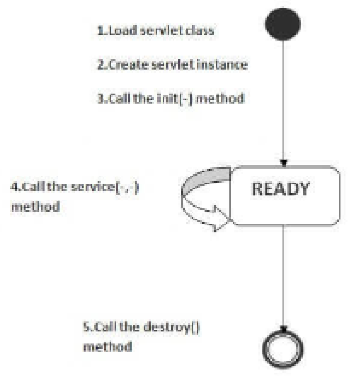
3) Body: It contains expressions and statements for lambda expression.

25	**Explain the Servlet Life Cyle and its architecture**



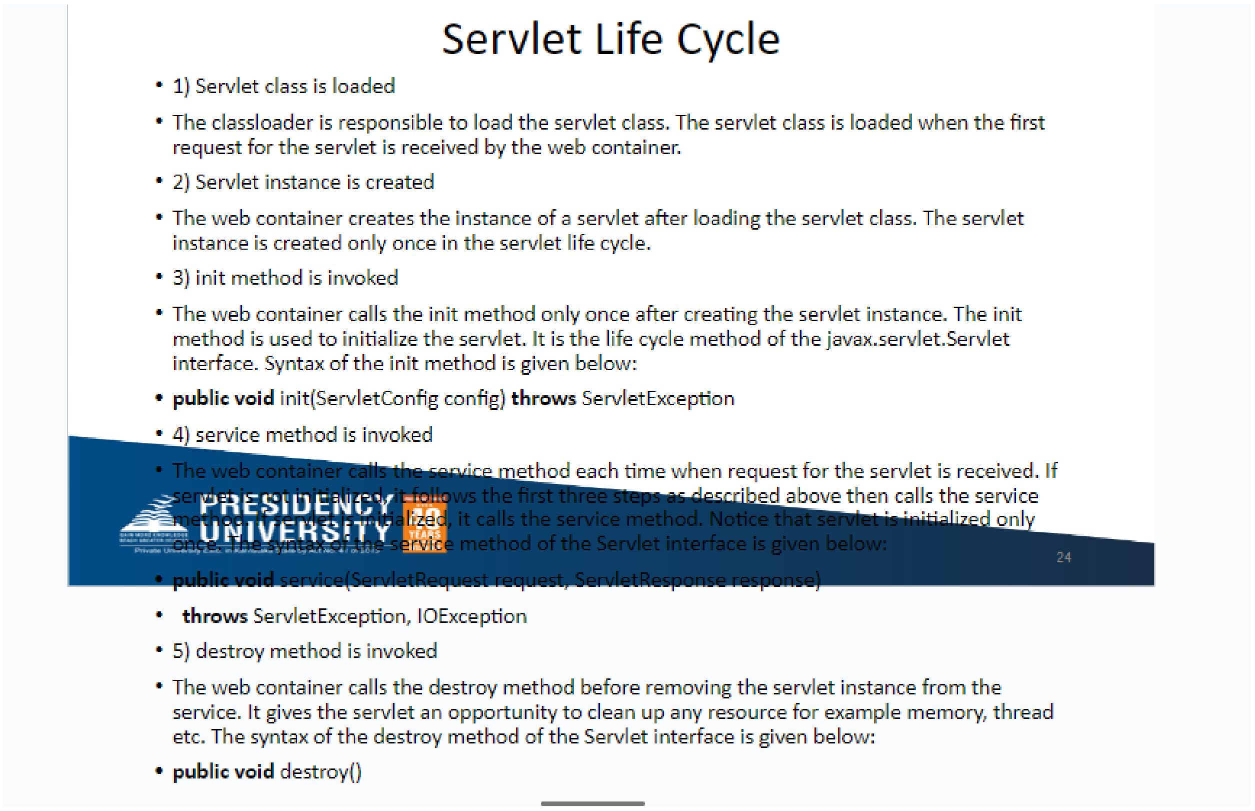
Life Cycle of a Servlet (Servlet Life Cycle)

- The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:
- Servlet class is loaded.
- Servlet instance is created.
- init method is invoked.
- service method is invoked.
- destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

PRESIDENCY UNIVERSITY

23



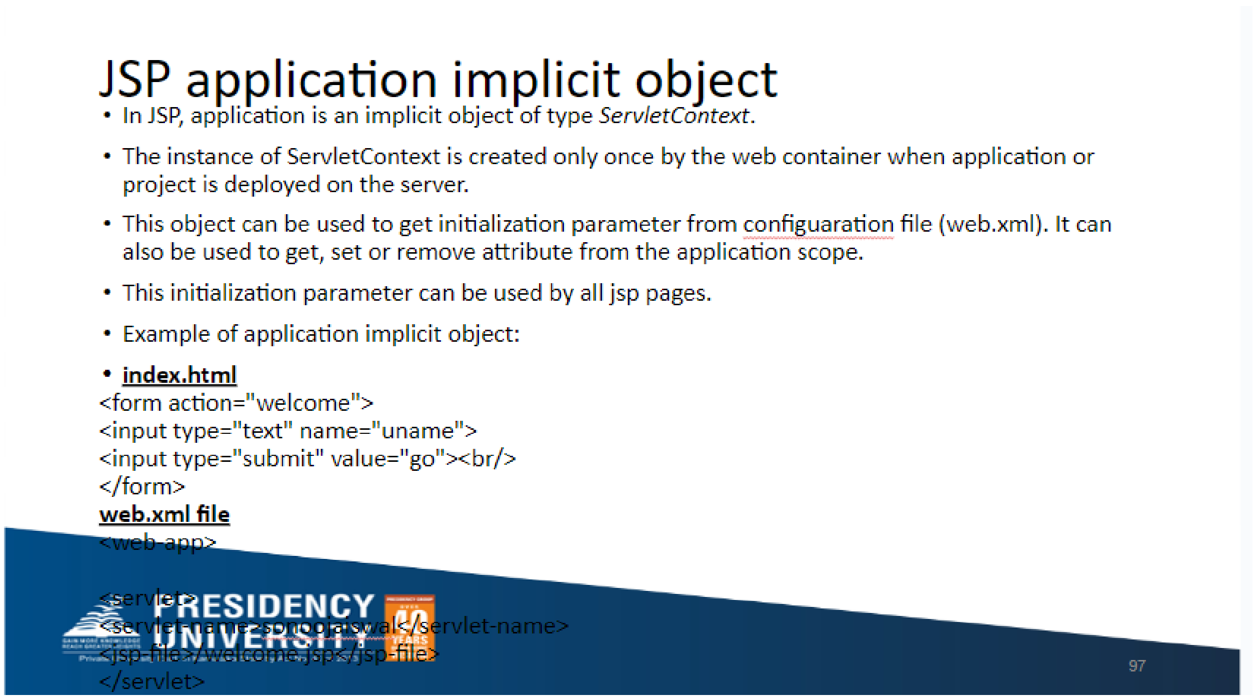
Servlet Life Cycle

- 1) Servlet class is loaded
- The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.
- 2) Servlet instance is created
- The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.
- 3) init method is invoked
- The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:
- **public void** init(ServletConfig config) **throws** ServletException
- 4) service method is invoked
- The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:
- **public void** service(ServletRequest request, ServletResponse response)
- **throws** ServletException, IOException
- 5) destroy method is invoked
- The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:
- **public void** destroy()

PRESIDENCY UNIVERSITY

24

27	**Explain JSP implicit objects.**



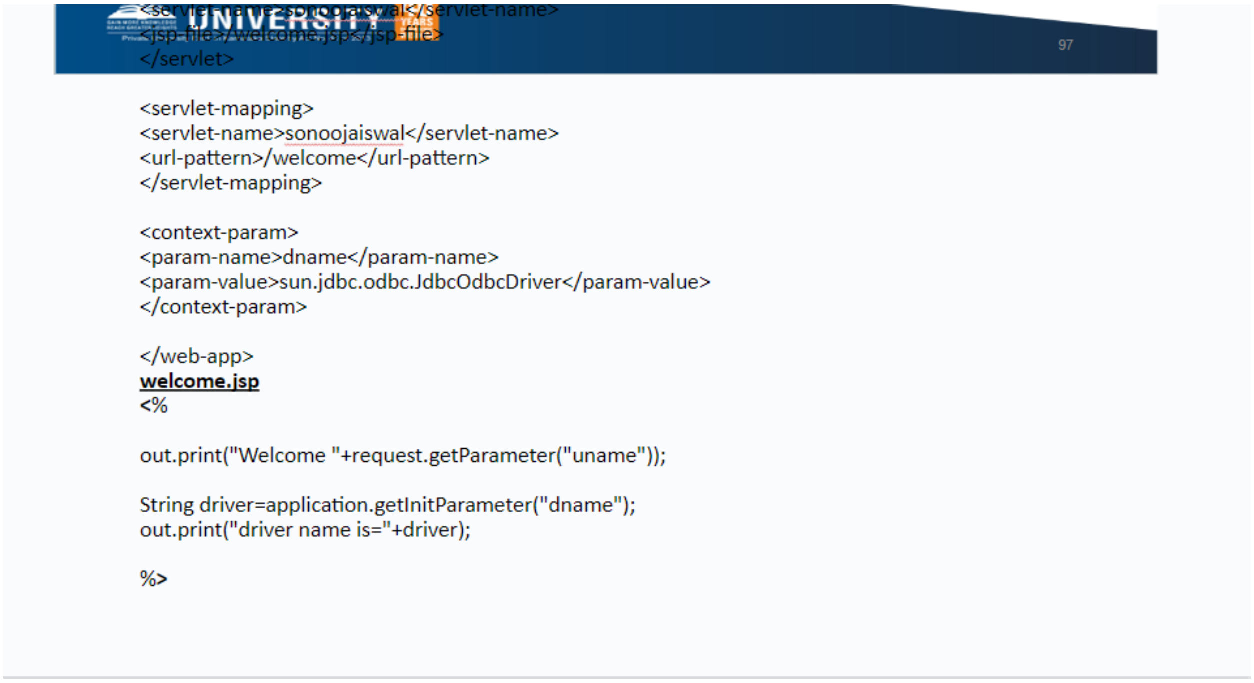
JSP application implicit object

- In JSP, application is an implicit object of type *ServletContext*.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.
- This initialization parameter can be used by all jsp pages.
- Example of application implicit object:
- **index.html**

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>
```



```
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>
```

```
<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

</web-app>
```

welcome.jsp

```
<%

out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);

%>
```

28	**Explain about JSP directives.**

JSP directives

- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- There are three types of directives:

❖ page directive

❖ include directive

❖ taglib directive

- Syntax of JSP Directive
- <%@ directive attribute="value" %>

PRESIDENCY

JSP page directive

- The page directive defines attributes that apply to an entire JSP page.
- Syntax of JSP page directive
- <%@ page attribute="value" %>
- Attributes of JSP page directive

o import

o contentType

o extends

o info

o buffer

o language

o isELIgnored

o isThreadSafe

o autoFlush

PRESIDENCY UNIVERSITY

import

- The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.Example of import attribute
- <html>
- <body>
-
- <%@ page **import**="java.util.Date" %>
- Today is: <%= **new** Date() %>
-
- </body>
- </html>

contentType

- The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".
- Example of contentType attribute

<html>
<body>

<%@ page contentType=application/msword %>
Today is: <%= **new** java.util.Date() %>

</body>
</html>

Extends & info

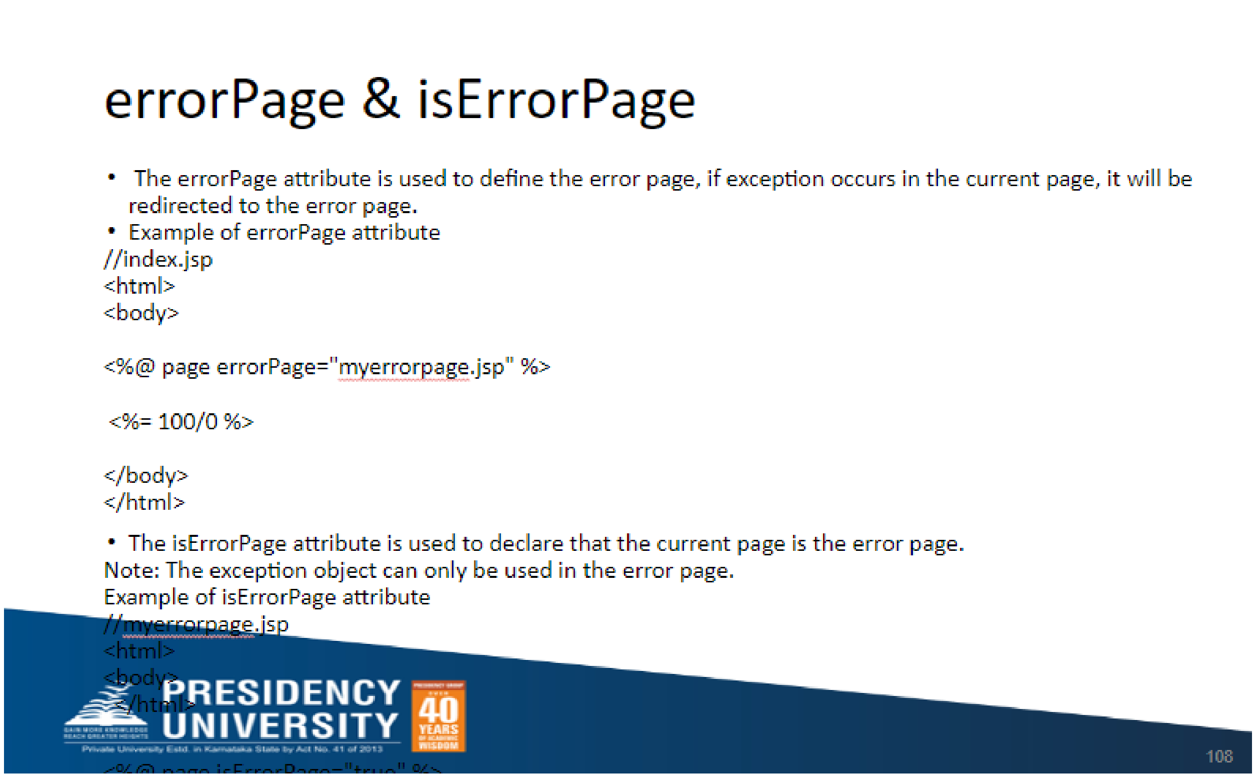
- The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.
- This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.
- Example of info attribute

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

- The web container will create a method getServletInfo() in the

language & isELIgnored & isThreadSafe

- The language attribute specifies the scripting language used in the JSP page. The default value is "java".
- We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expressio Language is enabled by default. We see Expression Language later.
- <%@ page isELIgnored="true" %>//Now EL will be ignored
- Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize th multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If yc make the value of isThreadSafe attribute like:<%@ page isThreadSafe="false" %>
- The web container in such a case, will generate the servlet as:
- **public class** SimplePage_jsp **extends** HttpJspBase
- **implements** SingleThreadModel{
- ....
- }



errorPage & isErrorPage

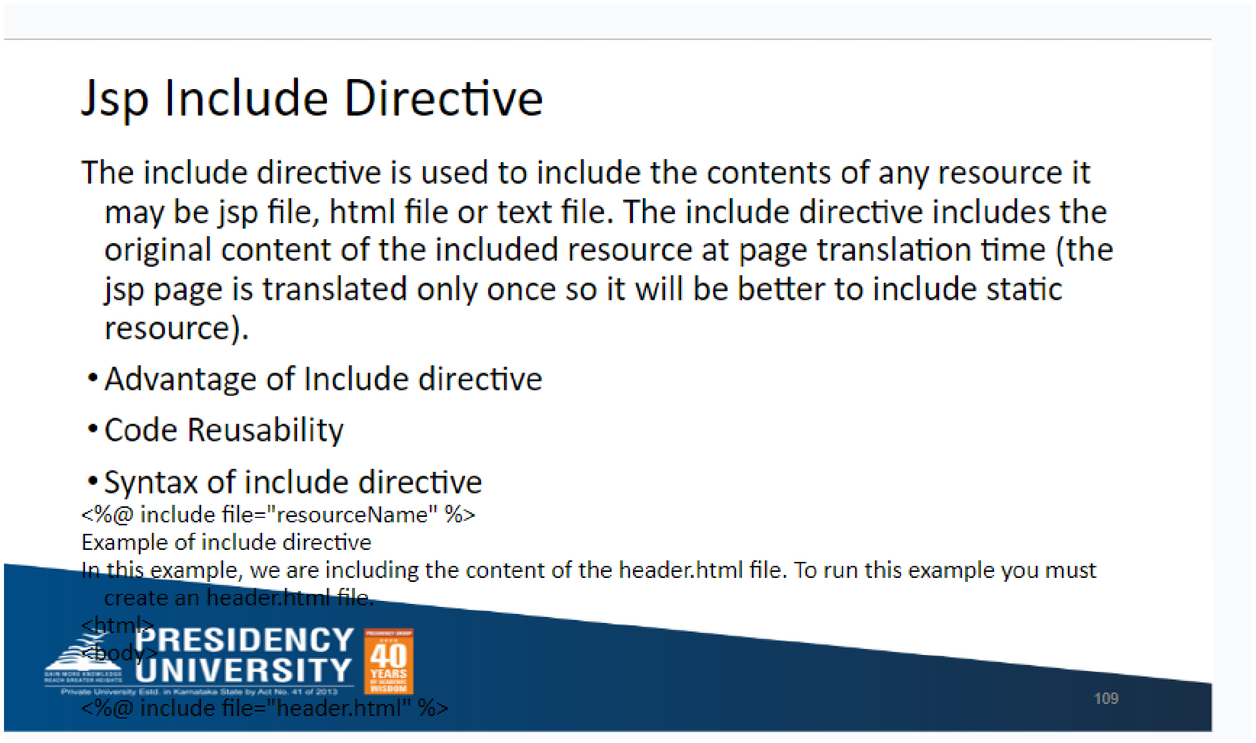
- The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.
- Example of errorPage attribute

```
//index.jsp
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</html>
```

- The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.
Example of isErrorPage attribute

```
//myerrorpage.jsp
<html>
<body>
  </html>
<%@ page isErrorPage="true" %>
```



Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

- Advantage of Include directive
- Code Reusability
- Syntax of include directive

```
<%@ include file="resourceName" %>
```

Example of include directive
In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>
<%@ include file="header.html" %>
```

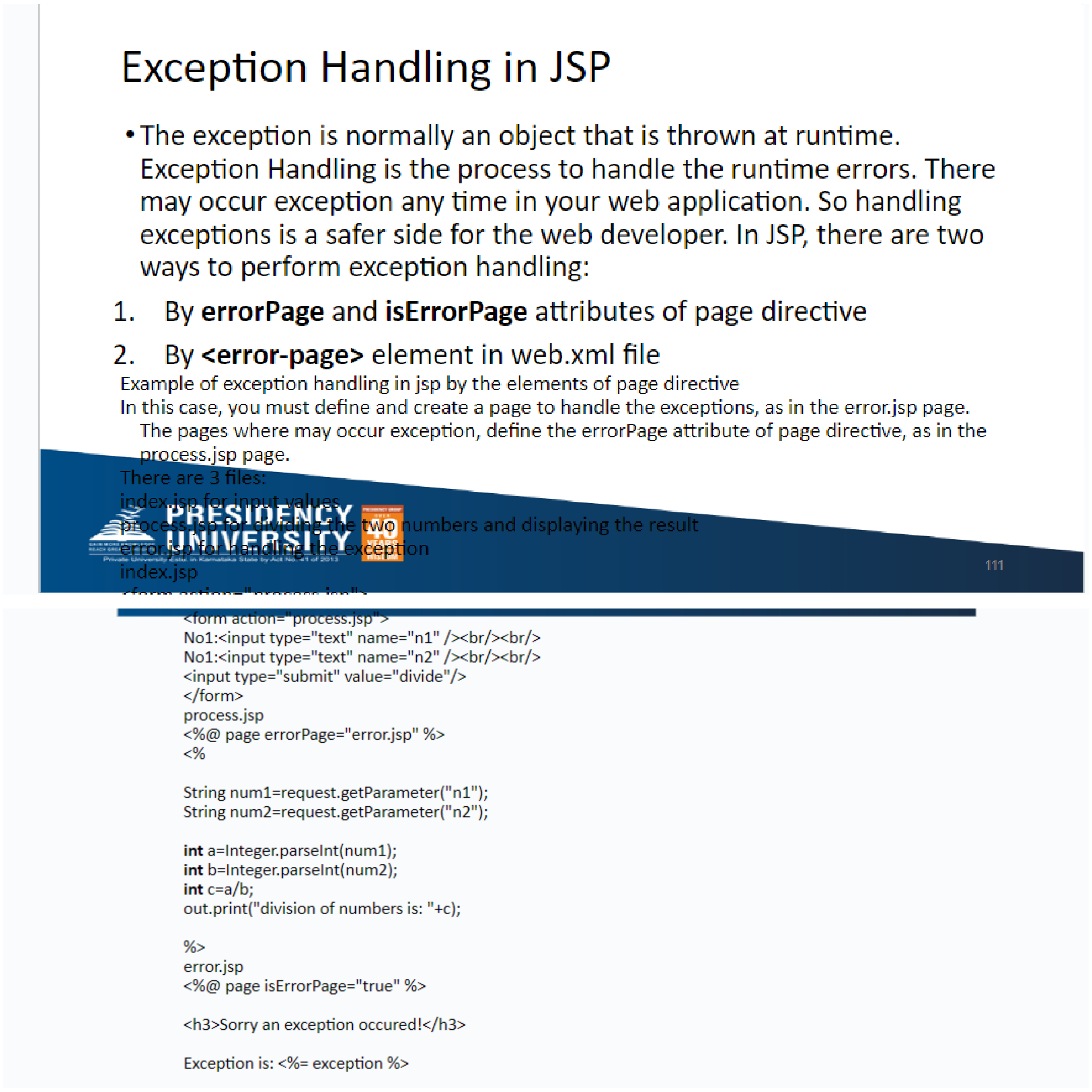
create an header.html file.

```
<html>
<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
</html>
```

JSP Taglib directive

- The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.
- Syntax JSP Taglib directive
- <%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
- Example of JSP Taglib directive
- In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```



Exception Handling in JSP

- The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

Example of exception handling in jsp by the elements of page directive

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

index.jsp for input values

process.jsp for dividing the two numbers and displaying the result

error.jsp for handling the exception

index.jsp

```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

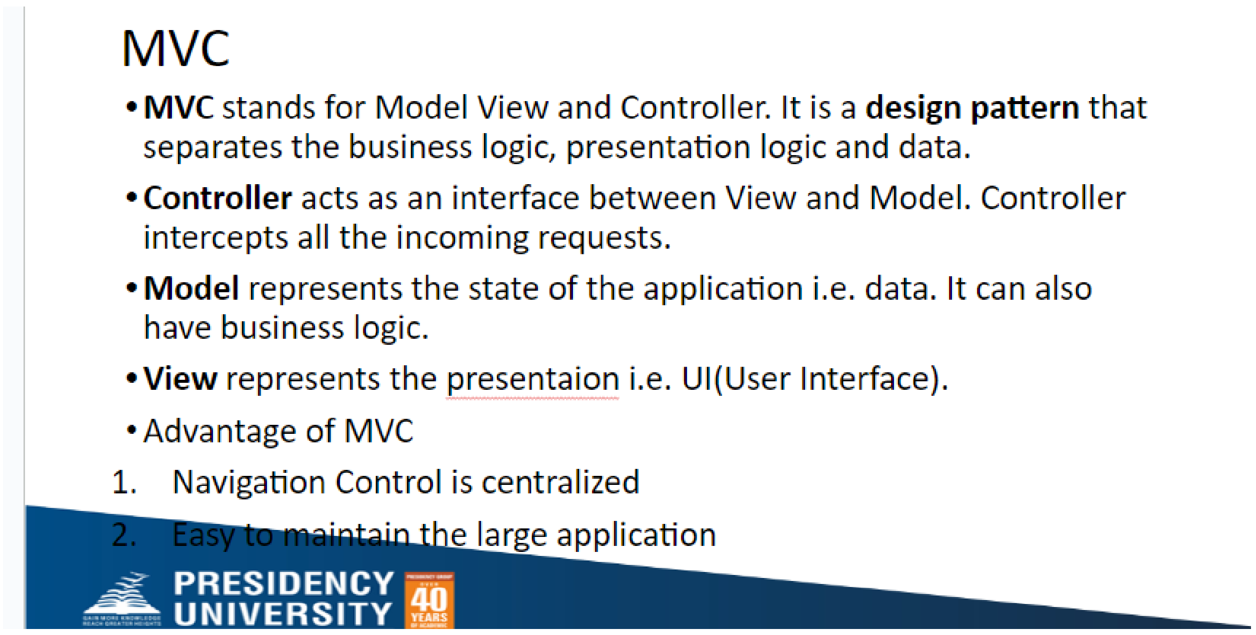
process.jsp

```
<%@ page errorPage="error.jsp" %>
<%

String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);

%>
```

error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```

30	**Illustrate MVC architecture with neat diagram.**



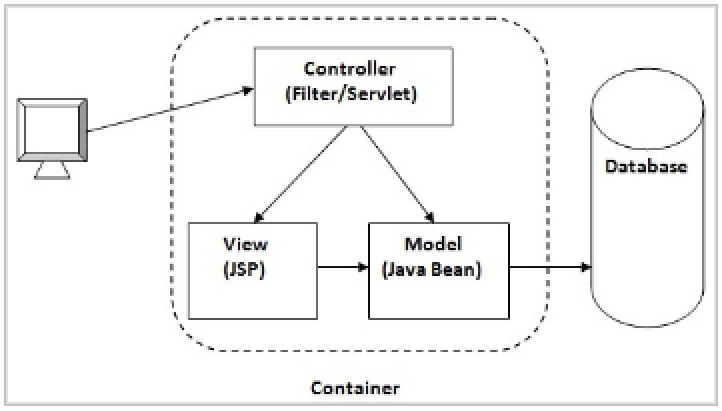
MVC

- **MVC** stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.
- **Controller** acts as an interface between View and Model. Controller intercepts all the incoming requests.
- **Model** represents the state of the application i.e. data. It can also have business logic.
- **View** represents the presentaion i.e. UI(User Interface).
- Advantage of MVC

1. Navigation Control is centralized
2. Easy to maintain the large application

PRESIDENCY UNIVERSITY

(Model 2) Architecture



MVC Example

- In this example, we are using servlet as a controller, jsp as a view component, Java Bean class as a model.
- In this example, we have created 5 pages:
- **index.jsp** a page that gets input from the user.
- **ControllerServlet.java** a servlet that acts as a controller.
- **login-success.jsp** and **login-error.jsp** files acts as view components.
- **web.xml** file for mapping the servlet.

File: index.jsp

```
<form action="ControllerServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: ControllerServlet

```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ControllerServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletRespo
  nse response)
        throws ServletException, IOException {
     response.setContentType("text/html");
     PrintWriter out=response.getWriter();
```

```
        String name=request.getParameter("name");
        String password=request.getParameter("password");

        LoginBean bean=new LoginBean();
        bean.setName(name);
        bean.setPassword(password);
        request.setAttribute("bean",bean);

        boolean status=bean.validate();

        if(status){
            RequestDispatcher rd=request.getRequestDispatcher("login-succ
ess.jsp");
            rd.forward(request, response);
        }
        else{
            RequestDispatcher rd=request.getRequestDispatcher("login-erro
r.jsp");
            rd.forward(request, response);
        }
```

File: LoginBean.java

```
package com.javatpoint;
public class LoginBean {
private String name,password;

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getPassword() {
    return password;
}
public void setPassword(String password) {
    this.password = password;
}
public void setPassword(String password) {
   this.password = password;
}
public boolean validate(){
   if(password.equals("admin")){
      return true;
   }
   else{
      return false;
   }
}
}
```

File: login-success.jsp

```
<%@page import="com.javatpoint.LoginBean"%>

<p>You are successfully logged in!</p>
<%
LoginBean bean=(LoginBean)request.getAttribute("bean");
out.print("Welcome, "+bean.getName());
%>
```

File: login-error.jsp

```
<p>Sorry! username or password error</p>
<%@ include file="index.jsp" %>
```