
UNIT 1 INTRODUCTION TO J2EE ARCHITECTURE AND DESIGN PATTERN

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Web Server and Web Container
- 1.3 Introduction to J2EE
- 1.4 Design Patterns
 - 1.4.1 MVC
 - 1.4.2 Repository Design Pattern
 - 1.4.3 Singleton
 - 1.4.4 Factory
- 1.5 Building Java Application JAR and WAR and deployment in Tomcat
- 1.6 Summary
- 1.7 Solutions/Answer to Check Your Progress
- 1.8 References/Further Reading

1.0 INTRODUCTION

You are well aware of the java programming language, which is an object oriented programming language. Java technology provides the specific environment in which Java programming language applications run. The J2EE platform provides runtime environment for developing and running large-scale, multi-tiered online/web and internet applications. This unit will give you an introduction of J2EE and its architecture. Also in this unit you will be introduced to some well-known design patterns. Design patterns give readymade explanations/solutions or templates to commonly occurring problems in the programming language. Design patterns are the best solutions that are tested and verified prototypes that can speed up your development process. There are many design patterns but it is not possible to cover all patterns in this course. This unit covers MVC, Repository, Singleton, Factory design patterns and some important terms, including Web server and Web Container. This unit will also provide you with the skill and process to package Java Application project to JAR/WAR and deploy your application in Tomcat.

The next Unit 2 and 3 of this block will give you detailed discussions on Servlet programming and session management in web applications. Unit 4 of this block will provide you with a basic understanding of Java Server Pages (JSP) components that make an entire JSP page, Java Bean, Custom tag, and many other concepts used in JSP. After studying this block, you will be able to use design patterns and build your java application using data retrieval and data storage in Servlets/JSP programming and their deployment in Tomcat.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- ... differentiate between web server and web container,
- ... know about design patterns and different types of design patterns,

- ... explain where and why singleton pattern is used,
- ... explain factory method design pattern and how to implement it,
- ... know the difference between jar and war files and packaging of java application to JAR/WAR file, and
- ... deploy java application as war file.

1.2 WEB SERVER AND WEB CONTAINER

You are well aware about web applications and you have opened many web applications on the internet. Assume that you have filled a form on this and received a response from them. This communication between you and web application are fulfilled using HTTP request and HTTP response with HTTP Protocol and method (You can study much more about these in the next Unit of this Block). When you write your programme in a programming language such as Servlet and JSP (this will be covered in detail in Unit 2, 3 and Unit 4 of this Block), you will be using some other terms like Web Server and Web Container in addition to HTTP request and response. This section describes you about the Web Server and Web Container.

Web Server is a server software that handles HTTP requests and responses to deliver web pages or web content to clients (i.e. web browser) using HTTP protocol. Web browser communicates with web server using the Hypertext Transfer Protocol (HTTP). Hypertext Transfer Protocol (HTTP) is specially meant to communicate between Client and Server using Web (or Internet). To successfully execute web application, the number of server side technologies (such as JSP, Servlets and PHP) and their libraries are installed on the web server. Without these libraries, a web server cannot execute those server technologies based applications. In other words, we may say that the web server creates an execution infrastructure for the server technologies. An example of web server is Apache HTTP Server.

Web Container is a web server component that handles Servlets, Java Server Pages (JSP) files, and other Web-tier components. Web container is also called a Servlet Container or Servlet Engine. It is the responsibility of the Web container to map a URL to a particular servlet. Also, Web container ensures that the mapped URL requester has the correct access rights. It means that it provides the run time environment to web applications. The most common web containers are Glassfish, Eclipse, JBOSS, Apache Tomcat, WebSphere and Web Logic.

For deployment and running the JSPs/Servlets we need a compatible web server with a servlet container.

You can know more about the Java 2 Enterprise Edition (J2EE) and its uses in java programming in the next section.

1.3 INTRODUCTION TO J2EE

In the previous section, you have learned about the two most important terms web server and web container which are required for Java programming. Through this section you know about the Java 2 Enterprise Edition (J2EE).

You are well aware about core java or Java Standard Edition (Java SE). Java SE provides all core functionalities of java programming language. Java technology is used as a programming language as well as a platform. A Java platform provides a specific environment in which Java applications run. The Java Enterprise Edition platform resides on top of the Java Standard Edition platform.

The J2EE platform is a set of services, application programming interfaces (APIs) and protocols. J2EE is used to develop and deploy multi-tier web-based enterprise applications using a series of protocols and application programming interfaces (APIs). J2EE contains several APIs such as Java Servlets, Java Server Pages (JSP), Enterprise Java Beans (EJB), Java Database Connectivity (JDBC), Java Message Service (JMS), Java Naming and Directory Interface (JNDI) and so on.

The J2EE application model divides applications into three basic parts like components, containers and connectors. The application developers works on the components part, whereas system vendors are responsible for implementing containers and connectors. Containers act as a mediator between clients and components by providing services like transaction support and resource pooling. Connectors provide bidirectional communication between J2EE components and enterprise systems. Below in figure 1, you can depict the functioning of J2EE model:

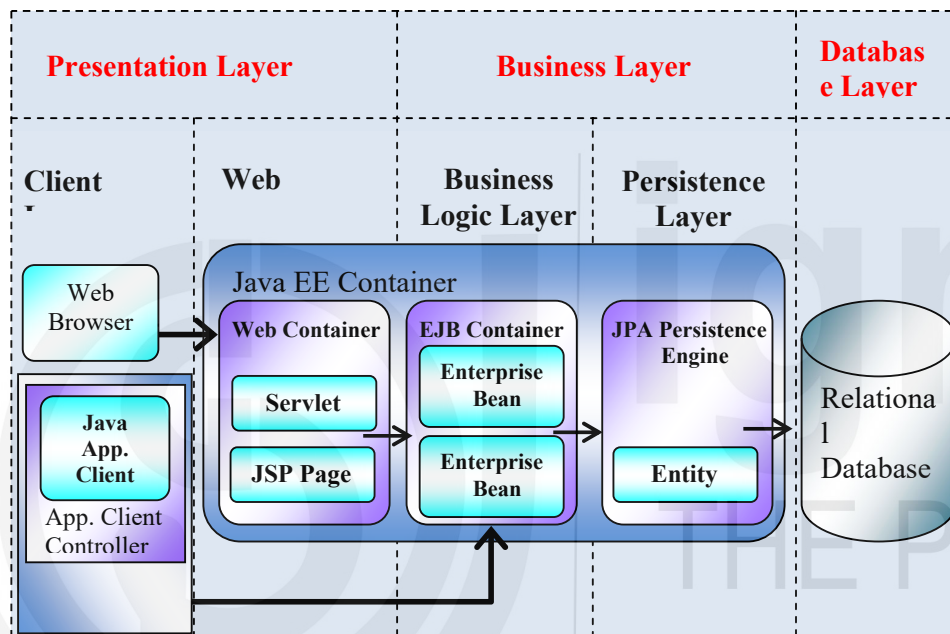


Figure 1: J2EE Architecture

A J2EE application contains four components or tiers: Presentation, Application, Business, and Resource adapter components. The presentation component is the client side component that is visible to the client and runs on the client's server. The Application component is web side layer that runs on the J2EE server. The business component is the business layer which includes server-side business logic such as JavaBeans, and it is also run on the J2EE server. The resource adaptor component comprises an enterprise information system.

When you develop J2EE application, you may find J2EE clients such as a web client, an application client, wireless clients or Java Web Start-enabled clients. For running J2EE application, you need a J2EE container which is a server platform, Java component can be run on this container using APIs provided through the Web container and EJB container. The EJB container is a server platform used for controlling the execution of Enterprise Bean. Also, the EJB container job is to provide local and remote access to enterprise beans.

For creating complex applications, you can use J2EE frameworks like Struts, Spring and Hibernate, which will you read in block-2 of this course.

☛ Check Your Progress 1

1. What is a web server, and how does it differ from a web container? Also, name any four web containers.

.....

.....

.....

.....

2. What is J2EE? What are the components of J2EE applications? What technologies are included in the J2EE platform?

.....

.....

.....

.....

3. Define the term module in J2EE. What are the four specific modules used in J2EE applications?

.....

.....

.....

.....

1.4 DESIGN PATTERNS

The previous section has given you an introductory lesson on the Java 2 Enterprise Edition (J2EE). This section gives a detailed description of the Design Patterns and their types and uses in Java.

Design patterns are the best solutions that are tested, verified developed prototypes that can speed up your development process. When you may face problems during software development, the design patterns gives you explanations for those problems. Experienced developers have developed these explanations to provide the finest solutions to the problems. It is also beneficial for un-experienced software developers to learn software design in a simpler manner. The concept of design pattern has been initiated by four authors collectively known as GOF (Gang of Four), and they have published a book (Design Patterns - Elements of Reusable Object-Oriented Software) on this concept. Design Patterns provides a general reusable solution to commonly occurring problems. It is not a complete design that can be directly mapped into your

code and solve your purpose. It is just like a template or explanation for how to solve your problems. It offers a common platform for all software developers.

There are 23 classic design patterns and they are categorized into three groups: Creational, Structural and Behavioral. The Creational design patterns deal with the creation of an object. Structural design patterns deal with the class structure, and the behavioral design patterns define the interaction between objects.

Creational design patterns

The Creational design patterns provide a way to deal with the creation of an object and define 5 design patterns such as Abstract Factory, Builder, Factory, Prototype and Singleton pattern from which two like Singleton and Factory design pattern are to be discussed below. The **Abstract Factory** design pattern allows to create families of related objects. This pattern permits us to create a Factory for factory classes. The **Builder** design pattern provides a valuable solution to many object creation problems in object-oriented programming. The objective of this pattern is to separate the creation of a complex object from its representation. The **Prototype** design pattern allows us to produce new objects by cloning an existing object using a clone() method. This pattern is useful when a particular resource is costly to create or when the abstract factory pattern is not preferred.

Structural design patterns

The Structural design patterns are related to the creation of class and object structure. These patterns define seven design patterns: Adapter, Bridge, Composite, Decorator, Façade, flyweight, and Proxy. The **Adapter** design pattern provides a way to work as a link between two unrelated interfaces. The object that links these incompatible or unrelated interfaces is called an Adapter. So, the adapter design pattern permits the interface of an existing class to be used as another interface. The adapter pattern is generally used to make available the existing classes for others without changing their source code. The **Composite** design pattern defines a pool of objects that are to be treated as a single object. The **Decorator** design pattern is used to change the functionality of an object at runtime. The **Façade** design pattern is used to generating wrapper interfaces on top of existing interfaces, and it hides the complications of the system and arranges a simpler interface to the client application. The **Flyweight** design pattern moderates the cost of generating and controlling a large number of related objects. The **Proxy** design pattern is used as a procurator or placeholder for another object to regulate the access and reduce cost as well as reduce complexity.

Behavioral design patterns

The third design patterns category is Behavioral, which defines 11 design patterns: Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor. They are specifically concerned with the interaction between objects. As the name recommends, the **chain of responsibility** design pattern builds a chain of receiver objects for a request. This pattern is used where the request sender is disassociated from its receiver based on the type of request. The **Command** design pattern is used to create objects containing information about an action, such as method name, owner of the method, and Parameter values necessary to be produced later. As the name suggests, the **Interpreter** design pattern provides an interpreter to deal with language grammar and it is used to interpret sentences in a language. The **Iterator** design pattern provides a way to access the collection object without revealing its underlying representation.

This pattern is frequently used in Java and .Net programming languages. **Mediator** design pattern provides a communication medium or a mediator class that handles all the communications between different classes. The capability of **Memento** design pattern is used to restore an object to its earlier state. The **Observer** design pattern is beneficial when an object is changed; then, you can get notified about the state of the object through this pattern. The **State** design pattern permits an object to modify its behaviour when its inner state is modified. The **Strategy** pattern or policy design pattern allows selecting an algorithm during runtime. The **Template Method** design pattern provides a way to create a superclass template method and permit its child classes to provide concrete behaviour. The **Visitor** design pattern gives a way to separate an operation from the object structure on which it operates. The basic purpose of this pattern is to define a new operation without modifying the original classes.

Besides these design patterns some more design patterns are also available. The MVC, Repository, Singleton and Factory design patterns are discussed below:

1.4.1 MVC Design Pattern

Model View Controller (MVC) design pattern is an architectural pattern for web applications. It can be used across many frameworks with many programming languages such as Java, PHP, Python, Ruby, etc. This design pattern is extensively used to design enterprise web applications and mobile applications. The MVC design pattern comprises three layers such as data, presentation and control information. This pattern requires each of these layers to act independently.

The MVC design pattern described three layers such as Model, View and Controller. In MVC pattern, M (Model) denotes only the pure application data and does not contain any logic for representing data to a user, whereas V (View) is responsible for presenting data to the user. The C (Controller) comes between the view and the model layer and it is responsible for accepting a request from the user, performs interactions on the data model objects, and sends it back to the view layer.

The UML diagram of MVC design pattern is depicted in figure 2. The following example demonstrates to you how MVC design pattern will work. The example contains total of four java files, from which three java files for MVC layers and one java file containing the main() method. For defining the MVC pattern, the first java file is created as 'UniversityModel', which acts as a model, 'univView' as a view that can display university details and 'UnivController' file is responsible for storing the data in univ object and updateView() method updates the data in a 'univView' class method. The 'MVCEExample' file will use 'UnivController' file to illustrate the MVC pattern.

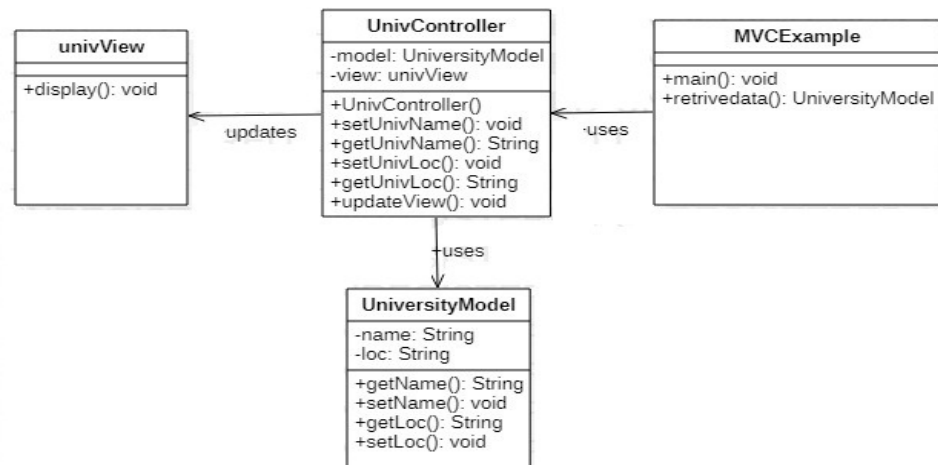


Figure 2: UML class diagram of MVC design pattern is

All files of MVC pattern example containing java source code are listed below.

1. You can create a UniversityModel.java file using the following code:

```
// UniversityModel.java
public class UniversityModel
{
    private String name;
    private String loc;

    public String getName()
    {return name; }

    public void setName(String name)
    {this.name = name; }

    public String getLoc()
    {return loc; }

    public void setLoc(String loc)
    {this.loc = loc; }
}
```

2. You can create a univView.java file using the following code:

```
// univView.java
public class univView
{
    public void display(String univName, String univLoc)
    {
        System.out.println("University Details: ");
        System.out.println("Name: " + univName);
        System.out.println("Location: " + univLoc);
    }
}
```

3. You can create a UnivController.java file using the following code:

```
// UnivController.java
public class UnivController
{
    private UniversityModel model;
    private univView view;

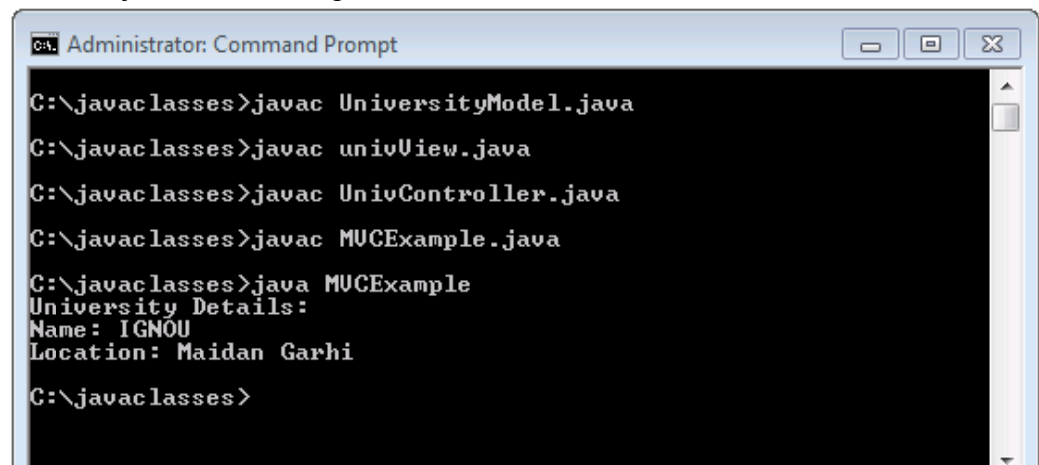
    public UnivController(UniversityModel model, univView view)
    {
        this.model = model;
        this.view = view;
    }
    public void setUnivName(String name)
    {
        model.setName(name);
    }
    public String getUnivName()
    {
        return model.getName();
    }
}
```

```
    }  
    public void setUnivLoc(String loc)  
    {  
        model.setLoc(loc);  
    }  
    public String getLoc()  
    {  
        return model.getLoc();  
    }  
    public void updateView()  
    {  
        view.display(model.getName(), model.getLoc());  
    }  
}
```

4. Now, create a MVCEXample.java file using the following code

```
//MVCEXample.java  
public class MVCEXample  
{  
    public static void main(String[] args)  
    {  
        UniversityModel model = retrivedata();  
  
        //Create a view : to write student details on console  
        univView view = new univView();  
        UnivController controller = new UnivController(model, view);  
        controller.updateView();  
    }  
    private static UniversityModel retrivedata()  
    {  
        UniversityModel univ = new UniversityModel();  
        univ.setName("IGNOU");  
        univ.setLoc("Maidan Garhi");  
        return univ;  
    }  
}
```

5. Now, compile all the files using javac command and run MVCEXample file with java as shown in figure-3:



```
C:\javaclasses>javac UniversityModel.java  
C:\javaclasses>javac univView.java  
C:\javaclasses>javac UnivController.java  
C:\javaclasses>javac MVCEXample.java  
C:\javaclasses>java MVCEXample  
University Details:  
Name: IGNOU  
Location: Maidan Garhi  
C:\javaclasses>
```

Figure 2: Output Screen for MVC pattern Example

1.4.2 Repository Design Pattern

In plain words, repository means something is deposited in storage. The repository meaning is same as in terms of design pattern, it is related to the data. The repository design pattern is used in data-centric applications. The Repository pattern is used to isolate the business logic layer and the data source layers in your application. The repository layer comes between the domain and data mapping layers. As shown in figure 4, the repository resides in the mid of the client business logic and data source layer. At the data source layer, a database can be kept/used.

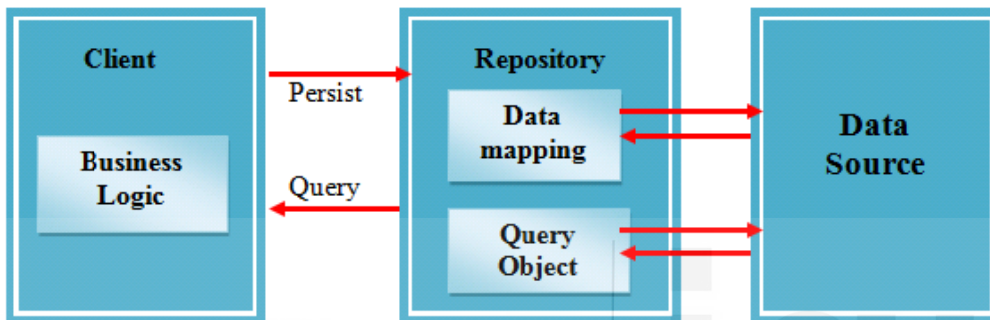


Figure 4: Block diagram for Repository Design Pattern

Suppose client business logic wishes a query from data source then it first goes to repository layer thereafter repository will send the query to the database, and then repository maps the data to the business entity. Similarly, when client business logic wants to save data in the database, data will not directly go to the database but will go through the repository layer.

While using repository pattern, your application's business logic layer need not have

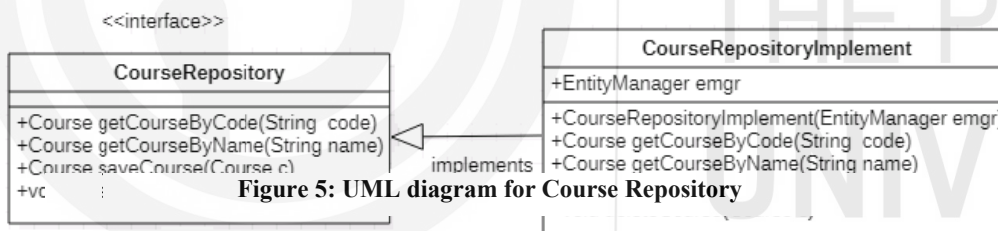


Figure 5: UML diagram for Course Repository

any knowledge on data persistence, it means the repository layer provides a way to hide the internal implementation of how data persistence happens. This way it empowers you to replace your data source without changing your business logic. This pattern is useful when you have several entities, and complex queries are applied on those entities.

This pattern is one of the most popular Java persistence patterns. This pattern is used with JPA and other frameworks. A JPA (Java Persistence API) is a java specification and defined in `javax.persistence` package. JPA is used to access and manage persist data between Java object and relational database. JPA can act as a link between object-oriented domain models and relational database systems. Various ORM (Object Relational Mapping) tools such as Hibernate, Spring are used by the JPA for implementing data persistence.

Assume that you want to develop a course repository for a university database application. For this, you need persistent data storage for courses where you may add and remove courses and search for them. For using repository design pattern, you can create an interface that defines read and write operations for a specific entity and this

interface is implemented by data store related classes. The UML diagram for such repository is as follows:

In the UML diagram, you have seen that there are four methods in 'CourseRepository' interface that you can use to find a course by code and name; save and delete course entity. You can create a course repository interface like the following code and then write a class for implementing 'CourseRepository' interface. If you are using any framework such as Spring Data JPA and Apache DeltaSpike Data then you can only define the repository interfaces, and these frameworks can generate standard repository implementations for you. Also, you can write your own implementation as per the requirements of the implementation complexity of your problem. For running this pattern, you need database connectivity with J2EE framework. The repository design pattern is stronger to you when you will read J2EE frameworks in the next Block-2 of this course.

```
public interface CourseRepository
{
    Course getCourseByCode(String code);
    Course getCourseByName(String name);
    Course saveCourse(Course c);
    void deleteCourse(Course c);
}
```

1.4.3 Singleton Design Pattern

The singleton pattern is a simplest software design pattern amongst the 23 well-known "Gang of Four" design patterns and it comes under the creational design pattern category, which deals with the creation of an object. As the name suggests, Singleton design pattern is used to create only one instance of a class, and all other classes of the application can use this instance. This type of pattern is mostly used in database connection and multi-threaded applications. Singleton pattern is used in logging, caching, thread pools, configuration settings etc. You can use Singleton pattern in other design patterns like Abstract Factory, Builder, Facade, Prototype etc., while developing your own solutions to a specific problem.

The UML class diagram of Singleton pattern is as under:

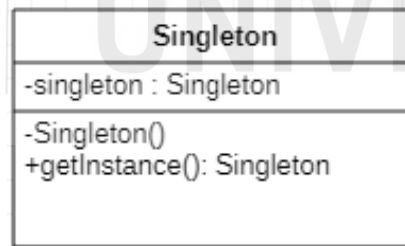


Figure 3: UML class diagram of Singleton pattern

An implementation of this pattern must ensure that only one instance of the singleton class will exist and that instance provides global access to others. While implementing singleton pattern, constructors of the class must be declared as private, and method is defined as public static that returns the instance of the class. The instance is typically stored as a private static variable. The static variable is initialized at some point before the static method is first called.

Java core libraries provide many classes which are written using singleton pattern. A few of them are listed below:

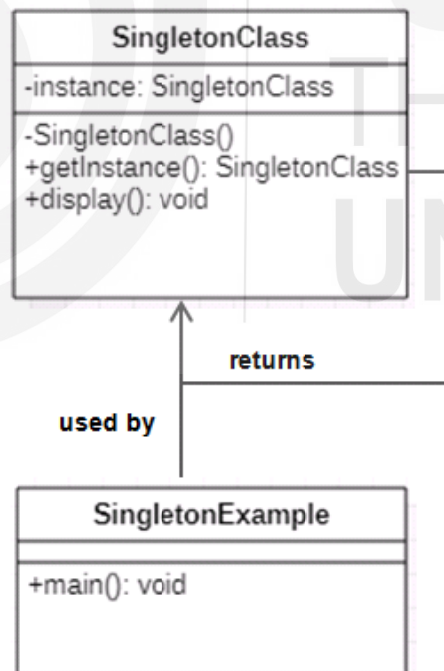
... Java.lang.Runtime with getRuntime() method

... Java.awt.Toolkit with getDefaultToolkit()
... Java.awt.Desktop with getDesktop()

A sample implementation of Singleton design pattern in Java is as follows:

```
public final class Singleton
{
    //A singleton class should have public visibility
    //static instance of class globally accessible
    private static final Singleton instance = new Singleton();
    private Singleton()
    {
        /* private constructor, so that class cannot be instantiated from outside this class
        */
    }
    public static Singleton getInstance()
    {
        /*declaration of the method as public static, so that instance can be used by
        other classes */
        return instance;
    }
}
```

The following example illustrates to you how the Singleton design pattern works. For this example, UML class diagram is as shown in figure 7:



This Singleton Design Pattern example contains two classes. One is 'SingletonClass', which behaves as Singleton class. This class have private constructor and provides a static method to get its static instance to other classes. The second class is 'SingletonExample', which use an instance of the 'SingletonClass' class. The implementation of this example is underneath.

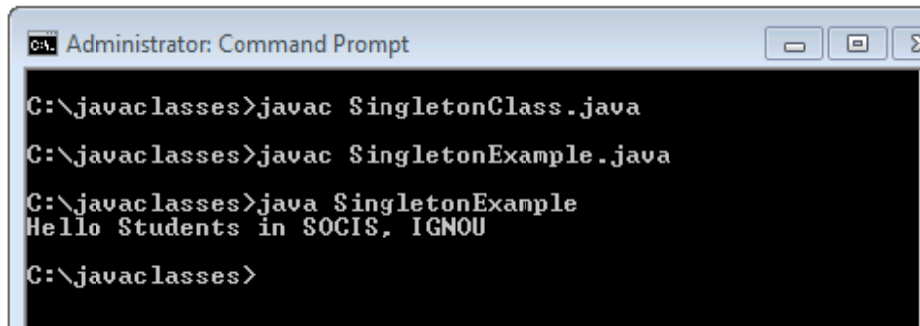
... First, you can create a Singleton Class as the name of 'SingletonClass' using the following source code:

```
// SingletonClass.java
public class SingletonClass
{
    //create an object of SingletonClass
    private static SingletonClass instance = new SingletonClass();
    private SingletonClass(){}
    public static SingletonClass getInstance()
    {
        return instance;
    }
    public void display()
    {
        System.out.println("Hello Students in SOCIS, IGNOU");
    }
}
```

... Create another class called 'SingletonExample' to get the only instance from the singleton class. The code of this class is listed below:

```
// SingletonExample.java
public class SingletonExample
{
    public static void main(String[] args)
    {
        //Get the instance created by Singleton class
        SingletonClass instance = SingletonClass.getInstance();
        instance.display();    //display data
    }
}
```

... Now compile the both classes and run the 'SingleExample'. The output of the this example is displayed in following figure-8:



```
C:\javaclasses>javac SingletonClass.java
C:\javaclasses>javac SingletonExample.java
C:\javaclasses>java SingletonExample
Hello Students in SOCIS, IGNOU
C:\javaclasses>
```

Figure 4: Output Screen for Singleton example

1.4.4 Factory Design Pattern

In the previous section, the Singleton design pattern has been explained. The Singleton design pattern belongs to the Creational pattern category. As you know, the

creational pattern is related to the creation of the object. In this section, you will learn one more design pattern named Factory, which also comes in the same category.

A Factory Design Pattern or Factory Method Pattern is one of the most used design pattern in java. It is a creational design pattern which deals with the creation of an object. As per the GOF, this pattern defines an interface or abstract class for creating an object, but subclasses are responsible for creating the instance of the class. The advantage of Factory Pattern is that it allows the sub-classes to choose the type of objects to create. In other words, this pattern is used to create an object where object creation logic is hidden to the client and refers to the newly created object using a common interface. This design pattern is also known as 'Virtual Constructor'.

To implement the Factory Design Pattern, first, you define a factory method inside an interface or abstract class and let the subclass use the above factory method and decide which object to create.

Factory Design Pattern implementation

The following example demonstrates to you how to work Factory Design Pattern. Assume that IGNOU wants to send announcement details of new courses to users

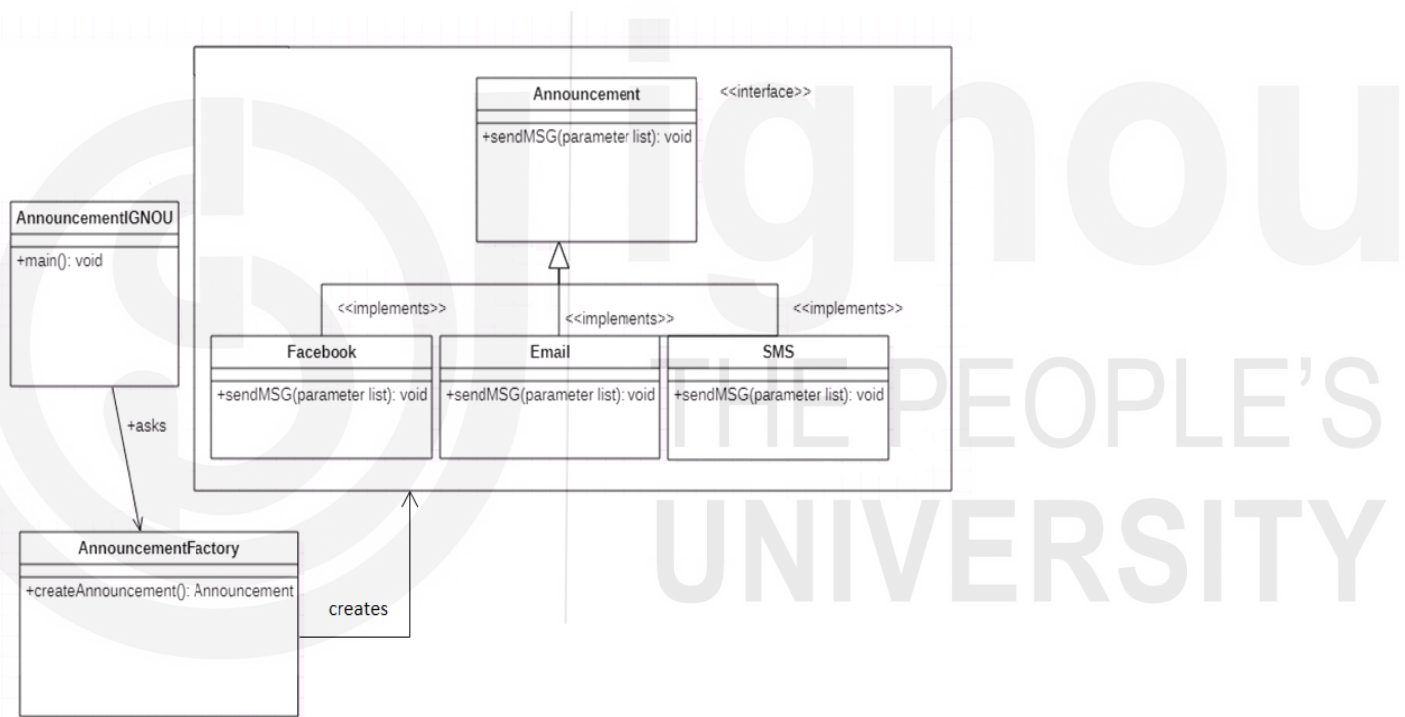


Figure 5: UML diagram for Factory Design Pattern

through Facebook, Email and SMS. Let's implement this example with the help of the Factory design pattern. For implementing this, first, you can design a UML class diagram, and then you can transform this class diagram into source code implementation. UML class diagram for this example using the Factory design pattern is shown in figure 9.

For implementing the above class diagram, you can create an interface as **Announcement** and three concrete classes such as **Facebook.java**, **Email.java** and **SMS.java**. These three classes can implement 'Announcement' interface. Besides these, you can also create two more classes. You can create a factory class 'AnnouncementFactory.java' to get an **Announcement** object. Creation of 'AnnouncementIGNOU.java' class is to use factory class and get an object of a concrete class.

Now you can follow the steps for the creation and running the example of Factory design pattern:

... First, you can create Announcement interface.

```
public interface Announcement
{
    void sendMSG();
}
```

... Now, you can create all three classes implementing the same interface as per the following codes:

```
//Facebook.java
public class Facebook implements Announcement
{
    public void sendMSG()
    {
        System.out.println("Sending announcement through Facebook");
    }
}
```

```
//Email.java
public class Email implements Announcement
{
    public void sendMSG()
    {
        System.out.println("Sending announcement as an e-mail");
    }
}
```

```
//SMS.java
public class SMS implements Announcement
{
    public void sendMSG()
    {
        System.out.println("Sending announcement as an SMS");
    }
}
```

... Now, you can create a Factory Class 'AnnouncementFactory.java' as code listed below:

```
// AnnouncementFactory.java
public class AnnouncementFactory
{
    public Announcement createAnnouncement(String media)
    {
        if (media == null || media.isEmpty())
            return null;
        if ("Facebook".equals(media))
        {
            return new Facebook();
        }
        else if ("EMAIL".equals(media))
        {

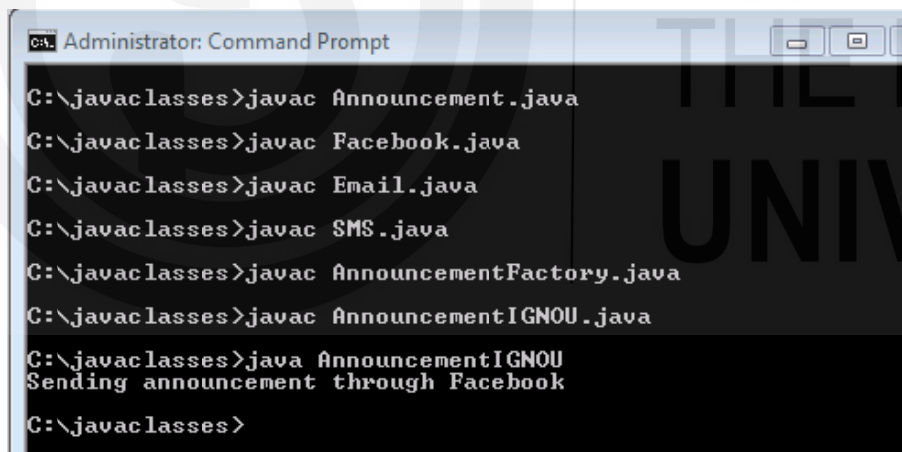
```

```
        return new Email();
    }
    else if ("SMS".equals(media))
    {
        return new SMS();
    }
    return null;
}
}
```

... You can create 'AnnouncementIGNOU.java'. Using this file, you can use 'AnnouncementFactory' class to create and get an object of concrete class. Here you are passing some information as "Facebook".

```
// AnnouncementIGNOU.java
public class AnnouncementIGNOU
{
    public static void main(String[] args)
    {
        AnnouncementFactory anFactory = new AnnouncementFactory()
        Announcement announcement =
        anFactory.createAnnouncement("Facebook");
        announcement.sendMessage();
    }
}
```

... Now, you can compile all the java files and run 'AnnouncementIGNOU' as per the following figure 10. This figure is also shown output at the command prompt.



```
C:\javaclasses>javac Announcement.java
C:\javaclasses>javac Facebook.java
C:\javaclasses>javac Email.java
C:\javaclasses>javac SMS.java
C:\javaclasses>javac AnnouncementFactory.java
C:\javaclasses>javac AnnouncementIGNOU.java
C:\javaclasses>java AnnouncementIGNOU
Sending announcement through Facebook
C:\javaclasses>
```

Figure 6: Compiling and output screen for Factory pattern example

Up to now, you have learned about the design patterns, including MVC, Repository, Singleton and Factory, and how to implement these patterns. In the next section, we will learn about the building and deployment of java/J2EE applications

☛ Check Your Progress 2

1. What do you mean by Design Patterns? Can you name some of the design patterns used in JDK core libraries?

.....

.....

.....

.....

2. What is Singleton pattern? Name one singleton class in Java. How can you create Singleton class in java? Can we create a clone of a singleton object? If yes, then how to prevent cloning of a singleton object?

.....

.....

.....

.....

3. Explain the benefits of Factory pattern.

.....

.....

.....

.....

1.5 BUILDING JAVA APPLICATION JAR AND WAR AND DEPLOYMENT IN TOMCAT

This section will help you to understand about building java applications to JAR and WAR files and deployment in Tomcat. The following sections described to you how to create WAR file, how to deploy WAR file and how to extract WAR file. You will find more about the installation process of Apache's Tomcat and running and deployment of your Servlets and JSP programs in section 2.7 of Unit 2 of this Block-1. Before an understanding of this description, you will know about the JAR and WAR packaging in Java.

JAR Packaging

The JAR (Java Archive) is a package file format. The file extension of JAR files is .jar and may contain libraries, resources, and metadata files. Basically, it is a zipped file containing the compressed versions of .class files, compiled Java libraries and applications. You can create a JAR file using the jar command like the following. You

can also create JAR files using IDEs. You can learn more about jar files in ‘Reference Section’ of this Unit.

```
jar cf name-jar-file input-file(s)
```

Where c option is used to create a JAR file, f option specifies the outputs which go to a file. You can define any filename for a JAR file. Using the input-file(s) argument, you can specify a space-separated list of one or more files that you wishes to include in your JAR file.

WAR Packaging

WAR (Web Archive) is used to package web applications. You can deploy on any Servlet/JSP container. It may contain JSP, Servlet, XML, images, HTML pages, CSS files and other resources. WAR package combines all the files into a single unit. It takes a smaller amount of time while transferring file from client to server. The extension of WAR files is .war needs a server to execute a WAR file.

The following sections describe to you how to create, deploy and extract WAR file.

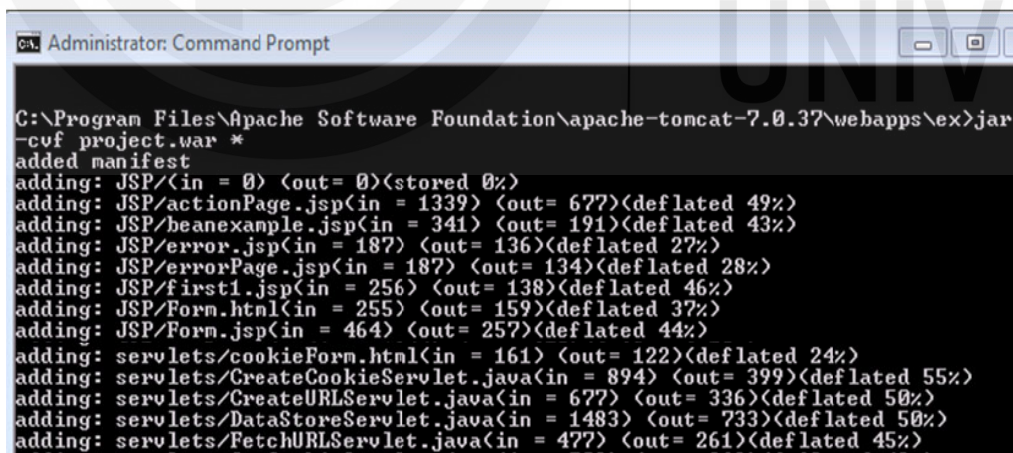
Create WAR file

You can create war file using jar tool of JDK or IDEs. You can use -c option of jar to create the war file. For creating war file, you can go inside the project application directory (outside the WEB-INF folder), then write command like the following:

```
jar -cvf projectname.war *
```

Where -c option is used to create file, -v to generate the verbose output and -f to specify the archive file name. The * (asterisk) symbol indicates that all the files of this directory (including sub directory).

The following figure-11 is displayed creation process of WAR file:



```
C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\webapps\ex>jar
-cvf project.war *
added manifest
adding: JSP/(in = 0) (out= 0)(stored 0%)
adding: JSP/actionPage.jsp(in = 1339) (out= 677)(deflated 49%)
adding: JSP/beanexample.jsp(in = 341) (out= 191)(deflated 43%)
adding: JSP/error.jsp(in = 187) (out= 136)(deflated 27%)
adding: JSP/errorPage.jsp(in = 187) (out= 134)(deflated 28%)
adding: JSP/first1.jsp(in = 256) (out= 138)(deflated 46%)
adding: JSP/Form.html(in = 255) (out= 159)(deflated 37%)
adding: JSP/Form.jsp(in = 464) (out= 257)(deflated 44%)
adding: servlets/cookieForm.html(in = 161) (out= 122)(deflated 24%)
adding: servlets/CreateCookieServlet.java(in = 894) (out= 399)(deflated 55%)
adding: servlets/CreateURLServlet.java(in = 677) (out= 336)(deflated 50%)
adding: servlets/DataStoreServlet.java(in = 1483) (out= 733)(deflated 50%)
adding: servlets/FetchURLServlet.java(in = 477) (out= 261)(deflated 45%)
```

Figure 7: Command prompt showing creation process of WAR file

Deploy the WAR file

You can deploy war file by placing the war file in a specific folder of the server. If

Figure 7: UML class diagram for Singleton Design Pattern Example

you are using the tomcat server and want to deploy this file manually, go to the 'webapps' directory of apache tomcat and paste the war file. The server will extract the war file internally. Now, you are able to access the web project through a browser.

... Suppose, you are deploying project.war file. First go to tomcat webapps folder and paste it.

... Go to tomcat->bin folder and start tomcat by clicking startup.bat file

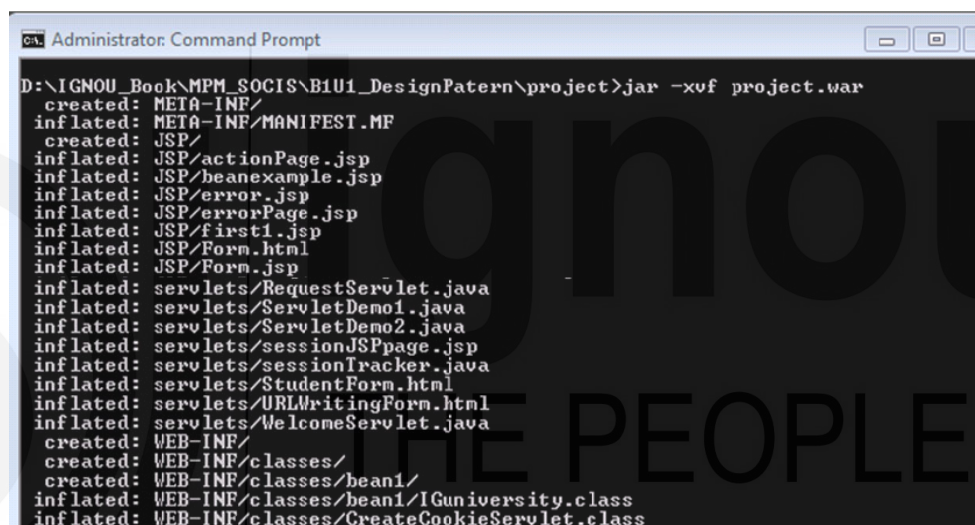
... Now, you can access your application from the browser. Open the browser and write in the address bar as localhost:port/projectname eg. localhost:8080/project

Extract war file manually

If you wish to extract the war file manually, then you need to use -x switch of jar tool of JDK. The following command is used to extract the war file.

```
jar -xvf projectname.war
```

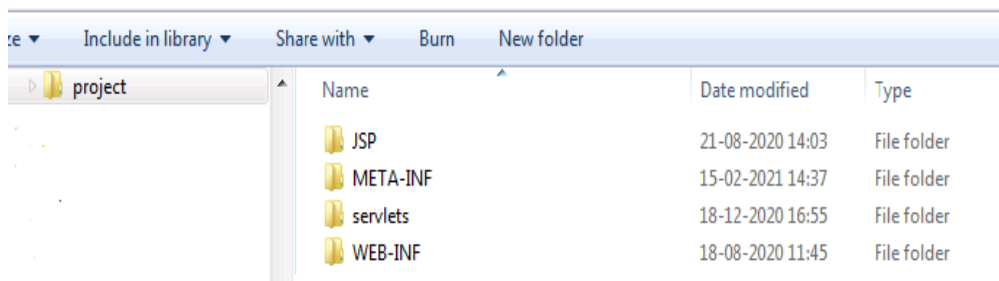
The following figure-12 shows you extraction process of WAR file:



```
D:\IGNOU_Book\MPM_SOCIS\B1U1_DesignPattern\project>jar -xvf project.war
created: META-INF/
inflated: META-INF/MANIFEST.MF
created: JSP/
inflated: JSP/actionPage.jsp
inflated: JSP/beanexample.jsp
inflated: JSP/error.jsp
inflated: JSP/errorPage.jsp
inflated: JSP/first1.jsp
inflated: JSP/Form.html
inflated: JSP/Form.jsp
inflated: servlets/RequestServlet.java
inflated: servlets/ServletDemo1.java
inflated: servlets/ServletDemo2.java
inflated: servlets/sessionJSPpage.jsp
inflated: servlets/sessionTracker.java
inflated: servlets/StudentForm.html
inflated: servlets/URLWritingForm.html
inflated: servlets/WelcomeServlet.java
created: WEB-INF/
created: WEB-INF/classes/
created: WEB-INF/classes/bean1/
inflated: WEB-INF/classes/bean1/IGUniversity.class
inflated: WEB-INF/classes/CreateCookieServlet.class
```

Figure 12: Screen showing extraction process of WAR file

The figure-13 shows you the file structure after extraction of WAR file. Now, you have learned about how to create, deploy and extract WAR files. The next unit will tell you how to create servlet and deploy in Tomcat.



Name	Date modified	Type
JSP	21-08-2020 14:03	File folder
META-INF	15-02-2021 14:37	File folder
servlets	18-12-2020 16:55	File folder
WEB-INF	18-08-2020 11:45	File folder

Figure 13: File structure after extraction process of WAR file

1. What is the difference between JAR and WAR files?

.....

.....

.....

.....

2. Describe the process of creation, deployment and extraction of WAR files.

.....

.....

.....

.....

1.6 SUMMARY

This unit focussed on the J2EE, Architecture and Design Patterns. You have learned about the J2EE and its architecture. By the services of Java 2 Enterprise Edition, you can develop large scale, component based, distributed, multi-tier applications. J2EE provides many APIs that can be used to build applications. This unit also covered 23 well-known design patterns which are categorized into three groups such as Creational, Structural and Behavioral. The design pattern has given you prior developed descriptions or template as solutions to commonly occurring problems in programming language. Design patterns are the best solutions that are tested, verified prototypes that can speed up your software development process. You have learned about the MVC, Singleton, Factory and Repository design patterns with examples. The MVC pattern described three layers such as Model, View and Controller. These three layers have separate functions like Model represent data, View presents data to user and Controller accepts the request from the user performs interactions on the data model objects and send it back to the view layer. The repository design pattern is used in data-centric applications. The repository design pattern provides a way to hide the internal implementation of how data persistence happens, and it empowers you to replace your data source without changing your business logic.

The singleton pattern deals with the creation of an object and is used to create only one instance of a class and all other classes of the application can use this instance. A Factory Design Pattern or Factory Method Pattern deals with creating an object and defines an interface or abstract class for creating an object, but subclasses are responsible for creating the instance of the class. At the end of this unit, you have learned about how java application can be built as a war file and their deployment in tomcat.

1.7 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

Check your Progress 1

- 1) **Web Server** is server software that handles HTTP requests and responses to deliver web pages or web content to clients (i.e. web browser) using HTTP protocol. Web server creates an execution infrastructure for the server technologies. An example of web server is Apache HTTP Server. A compatible web server with a servlet container is always necessary for deployment and running the JSPs/Servlets.

Web Container is the web server component that handles Servlets, Java Server Pages (JSP) files, and other Web-tier components. Web container is also called as Servlet Container or Servlet Engine. It is the responsibility of Web container to map a URL to a particular servlet and ensure that the URL requester has the correct access rights. It means that it provides the run time environment to web applications.

The most common web containers are Glassfish, Eclipse, JBOSS, Apache Tomcat, Websphere and Web Logic.

- 2) J2EE is used to develop and deploy multi-tier web-based enterprise applications using a series of protocols and application programming interfaces (APIs). J2EE contains many APIs such as Java Servlets, Java Server Pages (JSP), Enterprise JavaBeans (EJB), Java Database Connectivity (JDBC), Java Message Service (JMS), Java Naming and Directory Interface (JNDI) and so on. The J2EE platform consist of set of services, application programming interfaces (APIs), and protocols.

A J2EE application comprises four components or tiers such as Presentation, Application, Business and Resource adapter components. The presentation component is the client side component which is visible to the client and runs on the client's server. The Application component is a web side layer that runs on the J2EE server. The business component is the business layer which includes server-side business logic such as JavaBeans, and it also runs on the J2EE server. The resource adaptor component includes enterprise information system.

J2EE contains several API technologies such as Java Servlets, Java Server Pages (JSP), Enterprise Java Beans (EJB), Java Database Connectivity (JDBC), Java Message Service (JMS), Java Transaction API (JTA), Java Naming and Directory Interface (JNDI), JDBC data access API and so on. The J2EE platform is a set of services, application programming interfaces (APIs) and protocols.

- 3) A module in J2EE is a group of one or more components of the same container type and one component deployment descriptor of the same type. There are four different modules used in J2EE: application client module, web, EJB, and resource adapter module. The Application Client module is bundled as a JAR file which contains class files and client deployment descriptor (web.xml) file. The WEB module is bundled as a JAR file that comprises class files (servlets), web deployment descriptor file, JSP pages, images, and HTML files. The Enterprise Java Beans (EJB) module is wrapped as a JAR file, collections of class files (ejb) and EJB deployment descriptor. The fourth module is Resource Adapter which is packaged as a JAR file consisting of classes, a resource adapter deployment descriptor, Java interfaces and native libraries.

Check your Progress 2

- 1) Design patterns are the best solutions that are tested, verified developed prototypes that can speed up your development process. When you may face problems during software development, the design patterns give you explanations for those problems. Experienced developers have developed these explanations to offer the finest solutions to the problems. It is also beneficial for inexperienced software developers to learn software design in a simpler manner.

The name of the design patterns used in JDK core libraries are Decorator pattern (BufferedInputStream can decorate other streams such as FilterInputStream), Singleton pattern (which is used by Runtime, Calendar classes), Factory pattern (used by Wrapper class like Integer.valueOf) and Observer pattern (used by event handling frameworks like swing).

- 2) The singleton pattern is a simplest software design pattern amongst the 23 well-known "Gang of Four" design patterns. It comes under the creational design pattern category which deals with the creation of an object. Singleton design pattern is used to create only one instance of a class and all other classes of the application can use this instance. This type of pattern is mostly used in database connection and multi-threaded applications.

The name of singleton class in JDK is java.lang.Runtime. The implementation of Singleton design pattern is as under:

```
public final class Singleton
{
    //A singleton class should have public visibility
    //static instance of class globally accessible
    private static final Singleton instance = new Singleton();
    private Singleton()
    {
        /* private constructor, so that class cannot be instantiated from outside
        this class */
    }
    public static Singleton getInstance()
    {
        /*declaration of method as public static, so that instance can be used
        by other classes */
        return instance;
    }
}
```

You can create a clone of a singleton object. Java provide clone() method of Object class for cloning. This method is protected method. In java, by default, every class extends Object class, the object of any class, including Singleton class can call clone() method. If you want to create a clone of a singleton object, then class must be implemented by java.lang.Cloneable interface.

You can use throw exception within the body of clone() method to prevent cloning of a singleton object.

- 3) A Factory Design Pattern or Factory Method Pattern is mostly used design pattern in java. It is a creational design pattern which deals with the creation of an object. As per the GOF, this pattern defines an interface or abstract class for creating an object, but subclasses are responsible for creating the class's instance. In other words, this pattern is used to create objects where object creation logic is hidden to the client and refers to newly created objects using

a common interface. This design pattern is also known as ‘Virtual Constructor’. The advantage of Factory Pattern is that it allows the subclasses to choose the type of objects they create.

Check Your Progress 3

- 1) The JAR (Java Archive) is a package file format. The file extension of JAR files is .jar and may contain libraries, resources and metadata files. Basically, it is a zipped file containing the compressed versions of .class files, compiled Java libraries and applications. WAR (Web Archive) is used to package web applications. You can deploy any Servlet/JSP container. It may contain JSP, Servlet, XML, images, HTML pages, CSS files and other resources. It combines all the files into a single unit and takes a smaller amount of time while transferring file from client to server. The extension of WAR files is .war and needs a server to execute a WAR file.
- 2) You can create war file using jar tool of JDK. You can use -c switch of jar to create the war file. For creating war file, you can go inside the project application directory (outside the WEB-INF folder) then write command as `jar -cvf projectname.war *` on the command prompt. For the deployment of WAR file, you can place war file in specific folder of server. If you are using the tomcat server and want to deploy the war file manually, go to the ‘webapps’ directory of apache tomcat and paste the war file. The server will extract the war file internally. Now, you are able to access the web project through the browser. If you wish to extract the war file manually then you need to use -x switch of jar tool of JDK.

1.8 REFERENCES/FURTHER READING

- ... Kathy Sierra, Bryan Basham, and Bert Bates, “Head First Servlets and JSP”, O’Reilly Media, Inc., 2008.
 - ... Budi Kurniawan, “Java for the Web with Servlets, JSP, and EJB: A Developer’s Guide to J2EE Solutions: A Developer’s Guide to Scalable Solutions”, Techmedia, 2002.
 - ... <https://www.oracle.com/java/technologies/appmodel.html>
 - ... <https://docs.oracle.com/javase/8/docs/technotes/guides/javaws/>
 - ... https://docs.oracle.com/cd/B10018_07/migrate.902/a95110/overview.htm
 - ... Design Patterns: Elements of Reusable Object-Oriented Software – Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
 - ... <https://docs.oracle.com/javase/tutorial/deployment/jar/build.html>
 - ... <https://introcs.cs.princeton.edu/java/85application/jar/jar.html>
-