Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

**Syntax:** The syntax of an anonymous class expression is like the invocation of a constructor, except that there is a class definition contained in a block of code.

1. // Test can be interface,abstract/concrete class
2. Test t = new Test()
3. {
4. // data members and methods
5. public void test\_method()
6. {
7. ........
8. ........
9. }
10. };

Java anonymous inner class example using class

1. **abstract** **class** Person{
2. **abstract** **void** eat();
3. }
4. **class** TestAnonymousInner{
5. **public** **static** **void** main(String args[]){
6. Person p=**new** Person(){
7. **void** eat(){System.out.println("nice fruits");}
8. };
9. p.eat();
10. }
11. }
12. Output:
13. nice fruits

# Java Adapter Classes

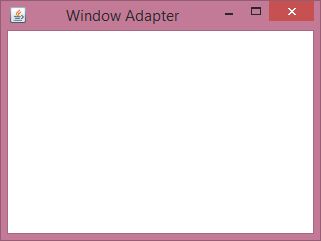
Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

## **Java WindowAdapter Example**

1. **import** java.awt.\*;
2. **import** java.awt.event.\*;
3. **public** **class** AdapterExample{
4. Frame f;
5. AdapterExample(){
6. f=**new** Frame("Window Adapter");
7. f.addWindowListener(**new** WindowAdapter(){
8. **public** **void** windowClosing(WindowEvent e) {
9. f.dispose();
10. }
11. });
13. f.setSize(400,400);
14. f.setLayout(**null**);
15. f.setVisible(**true**);
16. }
17. **public** **static** **void** main(String[] args) {
18. **new** AdapterExample();
19. }
20. }

Output:



An HTML form with a pre-selected checkbox:

<form action="/action\_page.php">  
  <input type="checkbox" name="vehicle" value="Bike"> I have a bike<br>  
  <input type="checkbox" name="vehicle" value="Car" checked> I have a car<br>  
  <input type="submit" value="Submit">  
</form>

Difference between web server and application server

* Web Server is designed to serve HTTP Content. App Server can also serve HTTP Content but is not limited to just HTTP. It can be provided other protocol support such as RMI/RPC
* Web Server is mostly designed to serve static content, though most Web Servers have plugins to support scripting languages like Perl, PHP, ASP, JSP etc. through which these servers can generate dynamic HTTP content.
* Most of the application servers have Web Server as integral part of them, that means App Server can do whatever Web Server is capable of. Additionally App Server have components and features to support Application level services such as Connection Pooling, Object Pooling, Transaction Support, Messaging services etc.
* As web servers are well suited for static content and app servers for dynamic content, most of the production environments have web server acting as reverse proxy to app server. That means while servicing a page request, static contents (such as images/Static HTML) are served by web server that interprets the request. Using some kind of filtering technique (mostly extension of requested resource) web server identifies dynamic content request and transparently forwards to app server.

**Definition:**

JavaFX is an excellent pedagogical tool for learning object-oriented programming.

**JavaFX vs Swing and AWT**

Swing and AWT are replaced by the JavaFX platform for developing rich Internet

applications.

When Java was introduced, the GUI classes were bundled in a library known as the Abstract

Windows Toolkit (AWT). AWT is fine for developing simple graphical user interfaces, but not

for developing comprehensive GUI projects. In addition, AWT is prone to platform-specific

bugs. The AWT user-interface components were replaced by a more robust, versatile, and flexible

library known as Swing components.

**why javaFx**

Swing is designed for developing desktop GUI applications. It is now

replaced by a completely new GUI platform known as JavaFX. JavaFX incorporates modern

GUI technologies to enable you to develop rich Internet applications. A rich Internet application

(RIA) is a Web application designed to deliver the same features and functions normally

associated with deskop applications.

**Scopes of javaFx**

A JavaFX application can run seemlessly on a desktop

and from a Web browser. Additionally, JavaFX provides a multi-touch support for touchenabled

devices such as tablets and smart phones. JavaFX has a built-in 2D, 3D, animation

support, video and audio playback, and runs as a stand-alone application or from a browser.

**Difference Between Applet and Japplet**

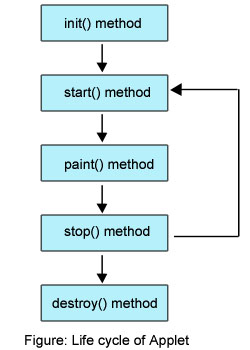
Applet = AWT  
JApplet = Swing Class   
  
If you're looking to use Swing components inside of your applet, JApplet would give you that functionality. Swing has superseded the original AWT, so it would still be a good idea to use JApplet in instead of Applet, simply for keeping up with the current libraries.  
  
The swing components   
provide better look and feel and they are light weight   
components compared to awt components.   
  
Swing components are derived from the javax.swing.\*; package and they are prefixed with 'J'(like JFrame,JButton,JApplet) .

**Difference Between Applet and servlet**

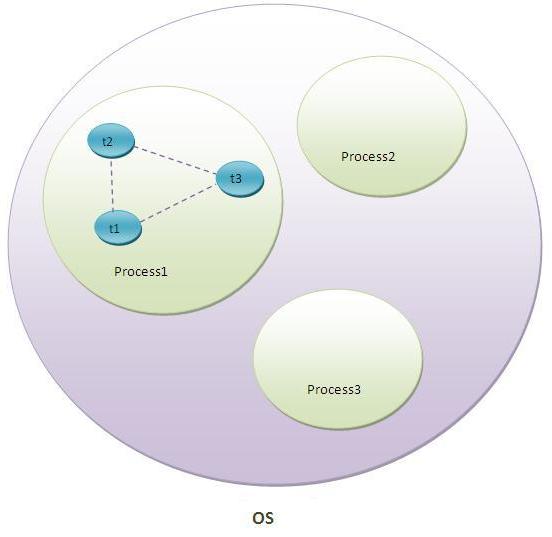
Servlets  
  
Servlets are Java based analog to CGI programs, implemented by means of servlet container associated with an HTTP server. Servlet is a server side component which runs on the web server. The servlet does not have a user interface.  
  
Applets  
  
These are applications which run on your Java compatible web browser. You are using them all the time. For example you might have opened a calculator on Google which is an applet that runs on your web browser. Though that calculator might not be made in Java but it still is an applet.

### Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



**Multithreading in java** is a process of executing **multiple threads** simultaneously. Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and **multithreading**, both are used to achieve multitasking.



**The J2EE Components**

* Enterprise JavaBeans (EJB) ...
* Java Servlets. ...
* JavaServer Pages (JSP) ...
* Java Naming Directory Interface (JNDI) ...
* Java Database Connectivity (JDBC) ...
* Java Message Service (JMS) ...
* Java Transaction API (JTA) ...
* Java API for XML (JAXP)

What is J2EE

**J2EE** is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online. The **J2EE** platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitiered, Web-based applications.

# JSP Tutorial

**JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.

A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

### Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows:

#### **1) Extension to Servlet**

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

#### **2) Easy to maintain**

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

#### **3) Fast Development: No need to recompile and redeploy**

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

#### **4) Less code than Servlet**

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

### Life cycle of a JSP Page

The JSP pages follows these phases:

* Translation of JSP Page
* Compilation of JSP Page
* Classloading (class file is loaded by the classloader)
* Instantiation (Object of the Generated Servlet is created).
* Initialization ( jspInit() method is invoked by the container).
* Reqeust processing ( \_jspService() method is invoked by the container).
* Destroy ( jspDestroy() method is invoked by the container).

**HTTP is a session stateless protocol**

A **stateless protocol** does not require the server to retain **session** information or status about each communications partner for the duration of multiple requests. **HTTP** is a **stateless protocol**, which means that the connection between the browser and the server is lost once the transaction ends. Http protocol is a stateless so we need to maintain state using **session** tracking techniques. Each time user requests to the server, server treats the request as the new request.

**Java Network programming**

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package provides support for the two common network protocols −

* **TCP** − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
* **UDP** − UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications

# RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:



# 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 (Model 2) Architecture**

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



# Java Bean

A Java Bean is a java class that should follow 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 Java Bean?**

|  |
| --- |
| According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance. |

### Simple example of java bean class

1. //Employee.java
3. **package** mypack;
4. **public** **class** Employee **implements** java.io.Serializable{
5. **private** **int** id;
6. **private** String name;
8. **public** Employee(){}
10. **public** **void** setId(**int** id){**this**.id=id;}
12. **public** **int** getId(){**return** id;}
14. **public** **void** setName(String name){**this**.name=name;}
16. **public** String getName(){**return** name;}
18. }

### How to access the java bean class?

|  |
| --- |
| To access the java bean class, we should use getter and setter methods. |

1. **package** mypack;
2. **public** **class** Test{
3. **public** **static** **void** main(String args[]){
5. Employee e=**new** Employee();//object is created
7. e.setName("Arjun");//setting value to the object
9. System.out.println(e.getName());
11. }}

# Session Bean

Session bean encapsulates business logic only, it can be invoked by local, remote and webservice client.

It can be used for calculations, database access etc.

The life cycle of session bean is maintained by the application server (EJB Container).

## **Types of Session Bean**

There are 3 types of session bean.

**1) Stateless Session Bean**: It doesn't maintain state of a client between multiple method calls.

**2) Stateful Session Bean**: It maintains state of a client across multiple requests.

**3) Singleton Session Bean**: One instance per application, it is shared between clients and supports concurrent access.

# Entity Bean in EJB 3.x

Entity bean represents the persistent data stored in the database. It is a server-side component.

In EJB 2.x, there was two types of entity beans: **bean managed persistence** (BMP) and container managed persistence (CMP).

Since EJB 3.x, it is deprecated and replaced by JPA (Java Persistence API) that is covered in the hibernate tutorial.

In hibernate tutorial, there are given hibernate with annotation examples where we are using JPA annotations. The JPA with Hibernate is widely used today.

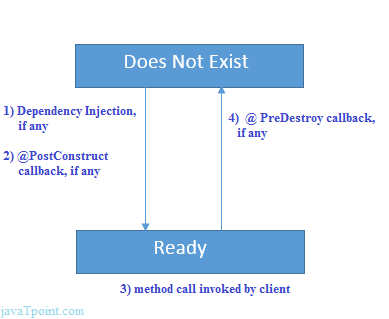
## **Annotations used in Stateless Session Bean**

There are 3 important annotations used in stateless session bean:

1. @Stateless
2. @PostConstruct
3. @PreDestroy

## **Life cycle of Stateless Session Bean**

There is only two states of stateless session bean: does not exist and ready. It is explained by the figure given below.



#### **Create stateless bean component**

To create the stateless bean component, you need to create a remote interface and a bean class.

*File: AdderImplRemote.java*

1. **package** com.javatpoint;
2. **import** javax.ejb.Remote;
4. @Remote
5. **public** **interface** AdderImplRemote {
6. **int** add(**int** a,**int** b);
7. }

*File: AdderImpl.java*

1. **package** com.javatpoint;
2. **import** javax.ejb.Stateless;
4. @Stateless(mappedName="st1")
5. **public** **class** AdderImpl **implements** AdderImplRemote {
6. **public** **int** add(**int** a,**int** b){
7. **return** a+b;
8. }
9. }

# Stateful Session Bean

**Stateful Session bean** *is a business object that represents business logic* like stateless session bean. But, it maintains state (data).

In other words, *conversational state* between multiple method calls is maintained by the container in stateful session bean.

## **Annotations used in Stateful Session Bean**

There are 5 important annotations used in stateful session bean:

1. @Stateful
2. @PostConstruct
3. @PreDestroy
4. @PrePassivate
5. @PostActivate

## **Example of Stateful Session Bean**

To develop stateful session bean application, we are going to use **Eclipse IDE** and **glassfish 3** server.

As described in the previous example, you need to create bean component and bean client for creating session bean application.

#### **1) Create stateful bean component**

Let's create a remote interface and a bean class for developing stateful bean component.

*File: BankRemote.java*

1. **package** com.javatpoint;
2. **import** javax.ejb.Remote;
3. @Remote
4. **public** **interface** BankRemote {
5. **boolean** withdraw(**int** amount);
6. **void** deposit(**int** amount);
7. **int** getBalance();
8. }

*File: Bank.java*

1. **package** com.javatpoint;
2. **import** javax.ejb.Stateful;
3. @Stateful(mappedName = "stateful123")
4. **public** **class** Bank **implements** BankRemote {
5. **private** **int** amount=0;
6. **public** **boolean** withdraw(**int** amount){
7. **if**(amount<=**this**.amount){
8. **this**.amount-=amount;
9. **return** **true**;
10. }**else**{
11. **return** **false**;
12. }
13. }
14. **public** **void** deposit(**int** amount){
15. **this**.amount+=amount;
16. }
17. **public** **int** getBalance(){
18. **return** amount;
19. }
20. }

**Types of Enterprise java Beans**

The following table describes the Enterprise Beans that you can execute in the EJB container provided with Application Server:

**Table 2-3 Classification of Enterprise Bean that can be executed in EJB container**

|  |  |
| --- | --- |
| **Major classification** | **Minor classification** |
| Session Bean | Stateless Session Bean |
| Stateful Session Bean |
| Singleton Session Bean |
| Entity Bean | BMP (Bean Managed Persistence) |
| CMP (Container-Managed Persistence) |
| Message-driven Bean | None |

Server.java

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.OutputStreamWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class Server

{

    private static Socket socket;

    public static void main(String[] args)

    {

        try

        {

            int port = 25000;

            ServerSocket serverSocket = new ServerSocket(port);

            System.out.println("Server Started and listening to the port 25000");

            //Server is running always. This is done using this while(true) loop

            while(true)

            {

                //Reading the message from the client

                socket = serverSocket.accept();

                InputStream is = socket.getInputStream();

                InputStreamReader isr = new InputStreamReader(is);

                BufferedReader br = new BufferedReader(isr);

                String number = br.readLine();

                System.out.println("Message received from client is "+number);

                //Multiplying the number by 2 and forming the return message

                String returnMessage;

                try

                {

                    int numberInIntFormat = Integer.parseInt(number);

                    int returnValue = numberInIntFormat\*2;

                    returnMessage = String.valueOf(returnValue) + "\n";

                }

                catch(NumberFormatException e)

                {

                    //Input was not a number. Sending proper message back to client.

                    returnMessage = "Please send a proper number\n";

                }

                //Sending the response back to the client.

                OutputStream os = socket.getOutputStream();

                OutputStreamWriter osw = new OutputStreamWriter(os);

                BufferedWriter bw = new BufferedWriter(osw);

                bw.write(returnMessage);

                System.out.println("Message sent to the client is "+returnMessage);

                bw.flush();

            }

        }

        catch (Exception e)

        {

            e.printStackTrace();

        }

        finally

        {

            try

            {

                socket.close();

            }

            catch(Exception e){}

        }

    }

}

**Client.java**

public class Client

{

    private static Socket socket;

    public static void main(String args[])

    {

        try

        {

            String host = "localhost";

            int port = 25000;

            InetAddress address = InetAddress.getByName(host);

            socket = new Socket(address, port);

            //Send the message to the server

            OutputStream os = socket.getOutputStream();

            OutputStreamWriter osw = new OutputStreamWriter(os);

            BufferedWriter bw = new BufferedWriter(osw);

            String number = "2";

            String sendMessage = number + "\n";

            bw.write(sendMessage);

            bw.flush();

            System.out.println("Message sent to the server : "+sendMessage);

            //Get the return message from the server

            InputStream is = socket.getInputStream();

            InputStreamReader isr = new InputStreamReader(is);

            BufferedReader br = new BufferedReader(isr);

            String message = br.readLine();

            System.out.println("Message received from the server : " +message);

        }

        catch (Exception exception)

        {

            exception.printStackTrace();

        }

        finally

        {

            //Closing the socket

            try

            {

                socket.close();

            }

            catch(Exception e)

            {

                e.printStackTrace();

            }

        }

    }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 8 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Connection class

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class MyDBClass {

protected Connection connection;

protected PreparedStatement ps;

protected ResultSet rs;

public MyDBClass() {

try {

//Class.forName("oracle.jdbc.driver.OracleDriver");

Class.forName("com.mysql.jdbc.Driver");

//System.out.println("11111111111111");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

public Connection getConnection() {

//String url="jdbc:oracle:thin:@192.168.8.1:1521:liu";

String url="jdbc:mysql://localhost:3306/test";

try {

connection = DriverManager.getConnection(url,"root","ict");

//connection = DriverManager.getConnection(url,"scott","tiger");

//System.out.println(connection);

} catch (SQLException e) {

e.printStackTrace();

}

return connection;

}

}

//Service class

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class ServiceClass extends MyDBClass{

public boolean InsetDB(String name ) {

this.getConnection();

String sql = "INSERT INTO STUDENT(NAME) VALUES(?)";

try

{

ps = connection.prepareStatement(sql);

ps.setString(1, name);

//rs = ps.executeQuery();

ps.executeUpdate();

ps.close();

//rs.close();

connection.close();

return true;

}

catch(SQLException e)

{

e.printStackTrace();

}

return false;

}

}

//First window class

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.stage.Stage;

import javafx.scene.layout.GridPane;

import javafx.scene.text.Text;

import javafx.scene.text.Font;

import javafx.scene.text.FontWeight;

import javafx.scene.text.FontPosture;

import java.util.Scanner;

import java.util.List;

public class FirstWindow extends Application {

GridPane gridPane = new GridPane();

private Text text1 = new Text("Hi there!");

private Label insertLabel = new Label("Enter Name or Id:");

private TextField nameTextFiled = new TextField();

private Button insertButton= new Button("Insert");

private TableView<Student> table = new TableView<Student>();

// private TableView table = new TableView();

private final Label label = new Label("Student List");

@Override // Override the start method in the Application class

public void start(Stage primaryStage) {

// Create a scene and place a button in the scene

gridPane.setHgap(5);

gridPane.setVgap(5);

text1.setFont(Font.font("Courier", FontWeight.BOLD, FontPosture.ITALIC, 15));

gridPane.add(text1, 5,5);

gridPane.add(insertLabel,5,7);

gridPane.add(nameTextFiled, 6,7);

gridPane.add(insertButton, 7, 7);

insertButton.setOnAction(e -> insertDB());

Scene scene = new Scene(gridPane, 550, 250);

primaryStage.setTitle("This the First Window"); // Set the stage title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

}

public void insertDB()

{

// System.out.print("You Have Entered:"+nameTextFiled.getText());

ServiceClass serviceClass = new ServiceClass();

serviceClass.InsetDB(nameTextFiled.getText());

List<Student> students = serviceClass.ReadFromDB();

if (students!=null){

for(int i=0; i<students.size(); i++){

Student std = students.get(i);

System.out.println(std.getId()+" "+std.getName());

}

}

}

public static void main(String[] args) {

launch(args);

}

}