Unit I – Java Database Connectivity (JDBC)

- JDBC Overview
- Connection Class
- Meta Data function
- SQLException
- SQLWarning
- Statement
- Resultset
- Other JDBC Classes

JDBC Overview

- JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.
- JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
- JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
    - Making a connection to a database.
    - Creating SQL or MySQL statements.
    - Executing SQL or MySQL queries in the database.
    - Viewing & Modifying the resulting records.

Applications of JDBC

- Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as,
    - Java Applications
    - Java Applets
    - Java Servlets
    - Java ServerPages (JSPs)
    - Enterprise JavaBeans (EJBs).
- All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.
- JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

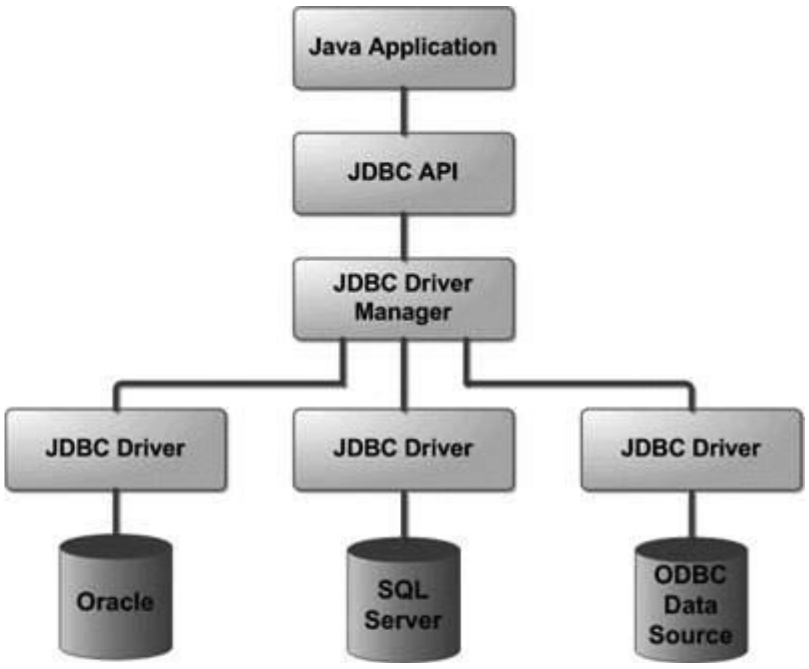
The JDBC 4.0 Packages

- The java.sql and javax.sql are the primary packages for JDBC 4.0.
- It offers the main classes for interacting with your data sources.
- The new features in these packages include changes in the following are as,
    - Automatic database driver loading.
    - Exception handling improvements.
    - Enhanced BLOB/CLOB functionality.

- Connection and statement interface enhancements.
- National character set support.
- SQL ROWID access.
- SQL 2003 XML data type support.

JDBC Architecture

- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –
  - **JDBC API:** This provides the application-to-JDBC Manager connection.
  - **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.
- Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application



JDBC Components

The JDBC API provides the following interfaces and classes –

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

JDBC Driver

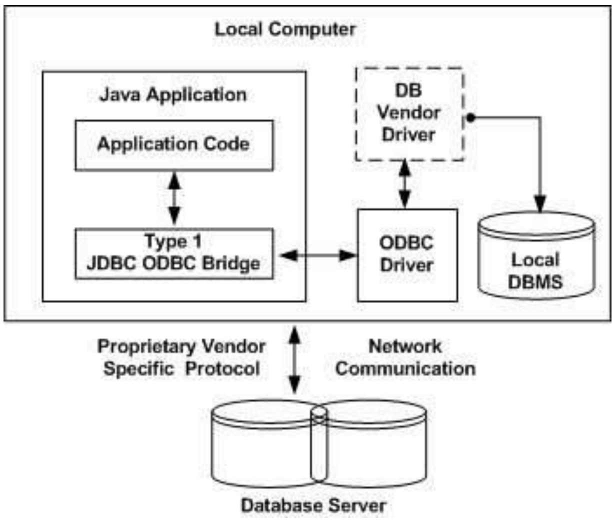
- JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.
- For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.
- The *Java.sql* package that ships with JDK, contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers. Third party vendors implement the *java.sql.Driver* interface in their database driver.

JDBC Drivers Types

- JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates.
- Sun has divided the implementation types into four categories:
  - Type 1: JDBC-ODBC Bridge Driver
  - Type 2: JDBC-Native API
  - Type 3: JDBC-Net pure Java
  - Type 4: 100% Pure Java

Type 1: JDBC-ODBC Bridge Driver

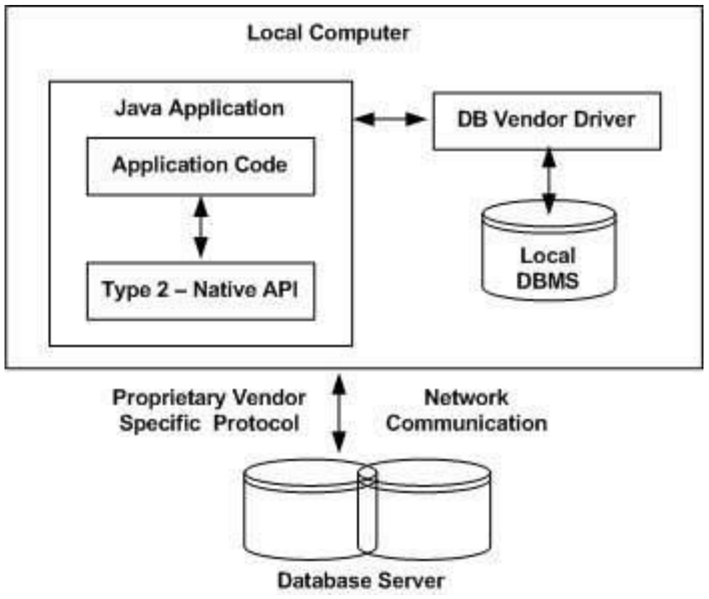
- In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine.
- Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.



- When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.
- The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

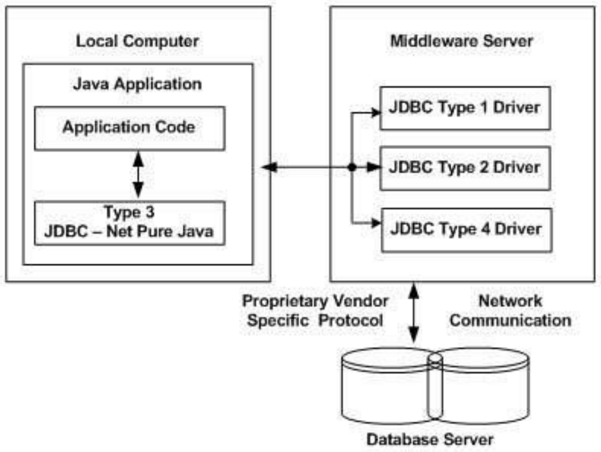
- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database.
- These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.
- If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.



- The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java

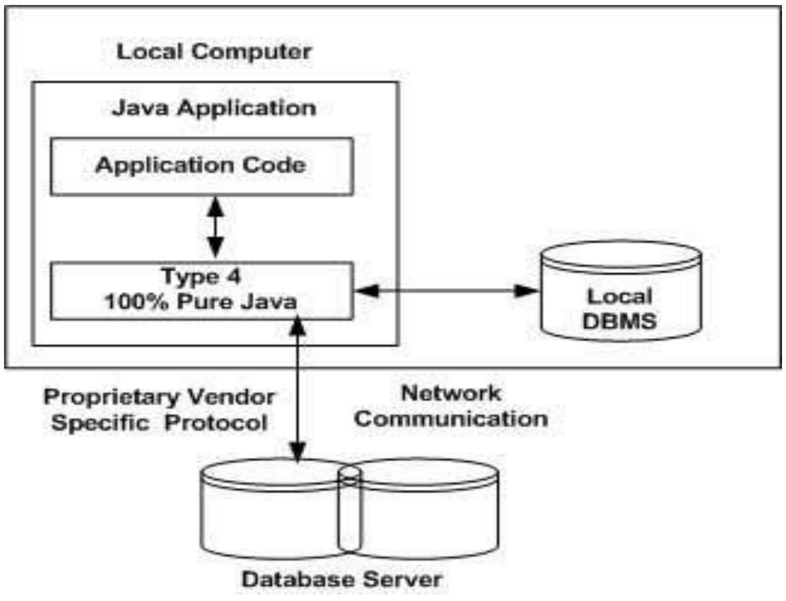
- In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server.



- The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.
- You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.
- Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java

- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



- MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Creating JDBC Application

There are following six steps involved in building a JDBC application:

- **Import the packages:** Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** will suffice.
- **Register the JDBC driver:** Requires that you initialize a driver so you can open a communication channel with the database.
- **Open a connection:** Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database.
- **Execute a query:** Requires using an object of type Statement for building and submitting an SQL statement to the database.

- **Extract data from result set:** Requires that you use the appropriate *ResultSet.getXXX()* method to retrieve the data from the result set.
- **Clean up the environment:** Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

JDBC Connection Class

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps:

- **Import JDBC Packages:** Add **import** statements to your Java program to import required classes in your Java code.
- **Register JDBC Driver:** This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.
- **Database URL Formulation:** This is to create a properly formatted address that points to the database to which you wish to connect.
- **Create Connection Object:** Finally, code a call to the *DriverManager* object's *getConnection( )* method to establish actual database connection.

Import JDBC Packages

- The **Import** statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.
- To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following *imports* to your source code

```
import java.sql.* ;  // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger support
```

Register JDBC Driver

- You must register the driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into the memory, so it can be utilized as an implementation of the JDBC interfaces.
- You need to do this registration only once in your program. You can register a driver in one of two ways.

1. Approach I - Class.forName()

- The most common approach to register a driver is to use Java's **Class.forName()** method, to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.
- The following example uses Class.forName( ) to register the Oracle driver –

```
try {
  Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
  System.out.println("Error: unable to load driver class!");
  System.exit(1);
}
```

- You can use **getInstance()** method to work around noncompliant JVMs, but then you'll have to code for two extra Exceptions as follows −

```
try {
  Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}
catch(ClassNotFoundException ex) {
  System.out.println("Error: unable to load driver class!");
  System.exit(1);
catch(IllegalAccessException ex) {
  System.out.println("Error: access problem while loading!");
  System.exit(2);
catch(InstantiationException ex) {
  System.out.println("Error: unable to instantiate driver!");
  System.exit(3);
}
```

2. Approach II - DriverManager.registerDriver()

- The second approach you can use to register a driver, is to use the static **DriverManager.registerDriver()** method.
- You should use the *registerDriver()* method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.
- The following example uses registerDriver() to register the Oracle driver

```
try {
  Driver myDriver = new oracle.jdbc.driver.OracleDriver();
  DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
  System.out.println("Error: unable to load driver class!");
  System.exit(1);
}
```

Database URL Formulation

- After loaded the driver, you can establish a connection using the **DriverManager.getConnection()** method.
- The following are list the three overloaded DriverManager.getConnection() methods:
    - getConnection(String url)
    - getConnection(String url, Properties prop)
    - getConnection(String url, String user, String password)
- Here each form requires a database **URL**. A database URL is an address that points to your database.
- Formulating a database URL is where most of the problems associated with establishing a connection occurs.
- Following table lists down the popular JDBC driver names and database URL.

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql:**//hostname/ databaseName

ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

- All the highlighted part in URL format is static and you need to change only the remaining part as per your database setup.

Create Connection Object

- There are three forms of **DriverManager.getConnection()** method to create a connection object.

1. ***Using a Database URL with a username and password***

- The most commonly used form of getConnection() requires you to pass a database URL, a *username*, and a *password*:
- Assuming you are using Oracle's **thin** driver, you'll specify a host:port:databaseName value for the database portion of the URL.
- If you have a host at TCP/IP address 192.0.0.1 with a host name of amrood, and your Oracle listener is configured to listen on port 1521, and your database name is EMP, then complete database URL would be:

```
jdbc:oracle:thin:@amrood:1521:EMP
```

- Now call getConnection() method with appropriate username and password to get a **Connection** object as follows:

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

2. ***Using Only a Database URL***

- A second form of the DriverManager.getConnection( ) method requires only a database URL:

```
DriverManager.getConnection(String url);
```

- However, in this case, the database URL includes the username and password and has the following general form

```
jdbc:oracle:driver:username/password@database
```

- So, the above connection can be created as follows:

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

3. ***Using a Database URL and a Properties Object***

- A third form of the DriverManager.getConnection( ) method requires a database URL and a Properties object –

```
DriverManager.getConnection(String url, Properties info);
```

- A Properties object holds a set of keyword-value pairs. It is used to pass driver properties to the driver during a call to the getConnection() method.
- To make the same connection made by the previous examples, use the following code –

```
import java.util.*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);
```

Closing JDBC Connections

- At the end of your JDBC program, it is required explicitly to close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.
- Relying on the garbage collection, especially in database programming, is a very poor programming practice. You should make a habit of always closing the connection with the close() method associated with connection object.
- To ensure that a connection is closed, you could provide a 'finally' block in your code. A *finally* block always executes, regardless of an exception occurs or not.
- To close the above opened connection, you should call close() method as follows:

```
conn.close();
```

- Explicitly closing a connection conserves DBMS resources, which will make your database administrator happy.

Metadata

- Generally, Data about data is known as metadata.
- JDBC has two types of Meta data:
  - DatabaseMetaData
  - ResultsetMetaData

DatabaseMetaData

- The DatabaseMetaData interface provides methods to get information about the database you have connected with like, database name, database driver version, maximum column length etc...
- Following are some methods of DatabaseMetaData class.

Method	Description
getDriverName()	Retrieves the name of the current JDBC driver
getDriverVersion()	Retrieves the version of the current JDBC

	driver
getUserName()	Retrieves the user name.
getDatabaseProductName()	Retrieves the name of the current database.
getDatabaseProductVersion()	Retrieves the version of the current database.
getNumericFunctions()	Retrieves the list of the numeric functions available with this database.
getStringFunctions()	Retrieves the list of the numeric functions available with this database.
getSystemFunctions()	Retrieves the list of the system functions available with this database.
getTimeDateFunctions()	Retrieves the list of the time and date functions available with this database.
getURL()	Retrieves the URL for the current database.
supportsSavepoints()	Verifies weather the current database supports save points
supportsStoredProcedures()	Verifies weather the current database supports stored procedures.
supportsTransactions()	Verifies weather the current database supports transactions.

ResultsetMetaData

- The ResultSetMetaData provides information about the obtained ResultSet object like, the number of columns, names of the columns, datatypes of the columns, name of the table etc…
- Following are some methods of ResultSetMetaData class.

Method	**Description**
getColumnCount()	Retrieves the number of columns in the current ResultSet object.
getColumnLabel()	Retrieves the suggested name of the column for use.
getColumnName()	Retrieves the name of the column.
getTableName()	Retrieves the name of the table.

SQLException

- Exception handling allows you to handle exceptional conditions such as program-defined errors in a controlled fashion.
- When an exception condition occurs, an exception is thrown. The term thrown means that current program execution stops, and the control is redirected to the nearest applicable catch clause. If no applicable catch clause exists, then the program's execution ends.
- JDBC Exception handling is very similar to the Java Exception handling but for JDBC, the most common exception you'll deal with is **java.sql.SQLException.**

SQLException Methods

- An SQLException can occur both in the driver and the database. When such an exception occurs, an object of type SQLException will be passed to the catch clause.
- The passed SQLException object has the following methods available for retrieving additional information about the exception:

Method	**Description**
getErrorCode( )	Gets the error number associated with the

	exception.
getMessage( )	Gets the JDBC driver's error message for an error, handled by the driver or gets the Oracle error number and message for a database error.
getSQLState( )	Gets the XOPEN SQLstate string. For a JDBC driver error, no useful information is returned from this method. For a database error, the five-digit XOPEN SQLstate code is returned. This method can return null.
getNextException( )	Gets the next Exception object in the exception chain.
printStackTrace( )	Prints the current exception, or throwable, and it's backtrace to a standard error stream.
printStackTrace(PrintStream s)	Prints this throwable and its backtrace to the print stream you specify.
printStackTrace(PrintWriter w)	Prints this throwable and it's backtrace to the print writer you specify.

- By utilizing the information available from the Exception object, you can catch an exception and continue your program appropriately. Here is the general form of a try block –

```
try {
  // Your risky code goes between these curly braces!!!
}
catch(Exception ex) {
  // Your exception handling code goes between these
  // curly braces, similar to the exception clause
  // in a PL/SQL block.
}
finally {
  // Your must-always-be-executed code goes between these
  // curly braces. Like closing database connection.
}
```

SQLWarning

- SQLWarning is a subclass of SQLException that holds database access warnings.
- Warnings do not stop the execution of a specific application, as exceptions do.
- A warning may be retrieved on the Connection object, the Statement object, PreparedStatement and CallableStatement objects, or on the ResultSet using **getWarnings** method.
  - SQLWarning warning = stmt.getWarnings();

Example source code

```
while (warning != null)
{
   System.out.println("Message: " + warning.getMessage());
   System.out.println("SQLState: " + warning.getSQLState());
   System.out.println("Vendor error code: " + warning.getErrorCode());
   warning = warning.getNextWarning();
}
```

JDBC Statements

- Once a connection is obtained we can interact with the database.
- The JDBC *Statement, CallableStatement,* and *PreparedStatement* interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.
- They also define methods that help bridge data type differences between Java and SQL data types used in a database.
- The following table provides a summary of each interface's purpose to decide on the interface to use.

Interfaces	**Recommended Use**
Statement	Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Statement Objects

Creating Statement Object

Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement( ) method, as in the following example:

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

- **boolean execute (String SQL)**: Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate (String SQL)**: Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery (String SQL)**: Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

Closing Statement Object

Just as you close a Connection object to save database resources, for the same reason you should also close the Statement object.
A simple call to the close() method will do the job. If you close the Connection object first, it will close the Statement object as well. However, you should always explicitly close the Statement object to ensure proper cleanup.

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  stmt.close();
}
```

The PreparedStatement Objects

The *PreparedStatement* interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.
This statement gives you the flexibility of supplying arguments dynamically.

Creating PreparedStatement Object

```
PreparedStatement pstmt = null;
try {
  String SQL = "Update Employees SET age = ? WHERE id = ?";
  pstmt = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

- All parameters in JDBC are represented by the **?** symbol, which is known as the parameter marker. You must supply values for every parameter before executing the SQL statement.
- The **setXXX()** methods bind values to the parameters, where **XXX** represents the Java data type of the value you wish to bind to the input parameter. If you forget to supply the values, you will receive an SQLException.
- Each parameter marker is referred by its ordinal position. The first marker represents position 1, the next position 2, and so forth. This method differs from that of Java array indices, which starts at 0.

- All of the **Statement object's** methods for interacting with the database (a) execute(), (b) executeQuery(), and (c) executeUpdate() also work with the PreparedStatement object. However, the methods are modified to use SQL statements that can input the parameters.

Closing PreparedStatement Object

- Just as you close a Statement object, for the same reason you should also close the PreparedStatement object.
- A simple call to the close() method will do the job. If you close the Connection object first, it will close the PreparedStatement object as well. However, you should always explicitly close the PreparedStatement object to ensure proper cleanup.

```
PreparedStatement pstmt = null;
try {
  String SQL = "Update Employees SET age = ? WHERE id = ?";
  pstmt = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  pstmt.close();
}
```

The CallableStatement Objects

- Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.

Creating CallableStatement Object

Suppose, you need to execute the following Oracle stored procedure:

```
CREATE OR REPLACE PROCEDURE getEmpName
  (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END;
```

Note: Above stored procedure has been written for Oracle, but we are working with MySQL database so, let us write same stored procedure for MySQL as follows to create it in EMP database

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
  (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
```

```
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

- Three types of parameters exist: IN, OUT, and INOUT.
- The PreparedStatement object only uses the IN parameter.
- The CallableStatement object can use all the three. The following are the definitions:

Parameter	**Description**
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from theOUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

- The following code snippet shows how to employ the **Connection.prepareCall()** method to instantiate a **CallableStatement** object based on the preceding stored procedure –

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

- The String variable SQL, represents the stored procedure, with parameter placeholders.
- Using the CallableStatement objects is much like using the PreparedStatement objects. You must bind values to all the parameters before executing the statement, or you will receive an SQLException.
- If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.
- When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type, to the data type that the stored procedure is expected to return.
- Once you call your stored procedure, you retrieve the value from the OUT parameter with the appropriate getXXX() method. This method casts the retrieved value of SQL type to a Java data type.

Closing CallableStatement Object

- Just as you close other Statement object, for the same reason you should also close the CallableStatement object.
- A simple call to the close() method will do the job. If you close the Connection object first, it will close the CallableStatement object as well. However, you should always explicitly close the CallableStatement object to ensure proper cleanup.

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  cstmt.close();
}
```

JDBC Resutset

- The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The *java.sql.ResultSet* interface represents the result set of a database query.
- A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.
- The methods of the ResultSet interface can be broken down into three categories −
  - **Navigational methods:** Used to move the cursor around.
  - **Get methods:** Used to view the data in the columns of the current row being pointed by the cursor.
  - **Update methods:** Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.
- The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding Statement that generates the ResultSet is created.
- JDBC provides the following connection methods to create statements with desired ResultSet −
  - **createStatement(int RSType, int RSConcurrency);**
  - **prepareStatement(String SQL, int RSType, int RSConcurrency);**
  - **prepareCall(String sql, int RSType, int RSConcurrency);**

- The first argument indicates the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.

Types of ResultSet

- The possible RSType are given below. If you do not specify any ResultSet type, you will automatically get one that is TYPE_FORWARD_ONLY.

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

Concurrency of ResultSet

- The possible RSConcurrency are given below. If you do not specify any Concurrency type, you will automatically get one that is CONCUR_READ_ONLY.

Concurrency	Description
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

- To create a forward-only, read only ResultSet object –

```
try {
  Statement stmt = conn.createStatement(
                ResultSet.TYPE_FORWARD_ONLY,
                ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) {
  ....
}
finally {
  ....
}
```

Navigating a Result Set

- There are several methods in the ResultSet interface that involve moving the cursor, including

S. No	Methods & Description
1	**public void beforeFirst() throws SQLException:** Moves the cursor just before the first row.
2	**public void afterLast() throws SQLException** Moves the cursor just after the last row.
3	**public boolean first() throws SQLException** Moves the cursor to the first row.
4	**public void last() throws SQLException** Moves the cursor to the last row.
5	**public boolean absolute(int row) throws SQLException** Moves the cursor to the specified row.
6	**public boolean relative(int row) throws SQLException**

	Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
7	**public boolean previous() throws SQLException** Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
8	**public boolean next() throws SQLException** Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
9	**public int getRow() throws SQLException** Returns the row number that the cursor is pointing to.
10	**public void moveToInsertRow() throws SQLException** Moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered.
11	**public void moveToCurrentRow() throws SQLException** Moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing

Viewing a Result Set

- The ResultSet interface contains dozens of methods for getting the data of the current row.
- There is a get method for each of the possible data types, and each get method has two versions –
    - One that takes in a column name.
    - One that takes in a column index.
- For example, if the column you are interested in viewing contains an int, you need to use one of the getInt() methods of ResultSet

S. No	Methods & Description
1	**public int getInt(String columnName) throws SQLException** Returns the int in the current row in the column named columnName.
2	**public int getInt(int columnIndex) throws SQLException** Returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

- Similarly, there are get methods in the ResultSet interface for each of the eight Java primitive types, as well as common types such as java.lang.String, java.lang.Object, and java.net.URL.
- There are also methods for getting SQL data types java.sql.Date, java.sql.Time, java.sql.TimeStamp, java.sql.Clob, and java.sql.Blob. Check the documentation for more information about using these SQL data types.

Updating a Result Set

- The ResultSet interface contains a collection of update methods for updating the data of a result set.
- As with the get methods, there are two update methods for each data type –
    - One that takes in a column name.
    - One that takes in a column index.
- For example, to update a String column of the current row of a result set, you would use one of the following updateString() methods:

S. No	Methods & Description
1	**public void updateString(int columnIndex, String s) throws SQLException** Changes the String in the specified column to the value of s.
2	**public void updateString(String columnName, String s) throws SQLException** Similar to the previous method, except that the column is specified by its name instead of its index.

- There are update methods for the eight primitive data types, as well as String, Object, URL, and the SQL data types in the java.sql package.
- Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods.

S. No	Methods & Description
1	**public void updateRow()** Updates the current row by updating the corresponding row in the database.
2	**public void deleteRow()** Deletes the current row from the database
3	**public void refreshRow()** Refreshes the data in the result set to reflect any recent changes in the database.
4	**public void cancelRowUpdates()** Cancels any updates made on the current row.
5	**public void insertRow()** Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

Other JDBC Classes
Date
Time
TimeStamp

Unit II – NETWORKING

- ➢ InetAddress
- ➢ TCP/ IP client sockets
- ➢ TCP/ IP server sockets
- ➢ URL
- ➢ URL Connection
- ➢ Datagrams
- ➢ Client/ Server application using RMI.

Networking Basics

- In Java Networking is a concept of connecting two or more computing devices together so that we can share resources.
- Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. Sharing resources
2. Centralize software management

Java Networking Terminology

- The widely used java networking terminologies are given below:
    1. IP Address
    2. Protocol
    3. Port Number
    4. MAC Address
    5. Connection-oriented and connection-less protocol
    6. Socket

1) IP Address

- IP address is a unique number assigned to a node of a network e.g. 192.168.0.1.
- It is composed of octets that range from 0 to 255.
- It is a logical address that can be changed.

2) Protocol

- A protocol is a set of rules basically that is followed for communication.
- For example: TCP, FTP, Telnet, SMTP, POP etc.

3) Port Number

- The port number is used to uniquely identify different applications.
- It acts as a communication endpoint between applications.
- The port number is associated with the IP address for communication between two applications.

4) MAC* (Media Access Control) *Address

- MAC Address is a unique identifier of NIC (Network Interface Controller).
- A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and Connection-less protocol

- In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.
- But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

- A socket is an endpoint between two way communication.

java.net package

- The java.net package provides many classes to deal with networking applications in Java.
- A list of these classes is given below:

○ Authenticator	CacheRequest	CacheResponse
○ ContentHandler	CookieHandler	CookieManager
○ DatagramPacket	DatagramSocket	DatagramSocketImpl
○ InterfaceAddress	JarURLConnection	MulticastSocket
○ InetSocketAddress	InetAddress	Inet4Address
○ Inet6Address	IDN	HttpURLConnection
○ HttpCookie	NetPermission	NetworkInterface
○ PasswordAuthentication	Proxy	ProxySelector
○ ResponseCache	SecureCacheResponse	ServerSocket
○ Socket	SocketAddress	SocketImpl
○ SocketPermission	URI	StandardSocketOptions
○ URL	URLClassLoader	URLConnection
○ URLDecoder	URLEncoder	URLStreamHandler

InetAddress class

- **InetAddress** class represents an IP address.
- The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com, etc.
- An IP address is represented by 32-bit or 128-bit unsigned number.
- An instance of InetAddress represents the IP address with its corresponding host name.
- There are two types of address types:
    - Unicast
    - Multicast.
- The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.
- Moreover, InetAddress has a cache mechanism to store successful and unsuccessful host name resolutions.

Methods of InetAddress class

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	it returns the instance of InetAdddress containing local host name and address.
public String getHostName()	it returns the host name of the IP address.
public String getHostAddress()	it returns the IP address in string format.

Example of InetAddress class

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");
System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Host Name: www.javatpoint.com
IP Address: 206.51.231.148
```

Socket Programming

- Java Socket programming is used for communication between the applications running on different JRE.
- Java Socket programming can be connection-oriented or connection-less.
- Socket and ServerSocket classes are used for connection-oriented socket programming
- DatagramSocket and DatagramPacket classes are used for connection-less socket programming.
- The client in socket programming must know two information:

    1. IP Address of Server
    2. Port number

- Here, we are going to make one-way client and server communication.
- In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket.
- The Socket class is used to communicate client and server. Through this class, we can read and write message.
- The ServerSocket class is used at server-side.
- The accept() method of ServerSocket class blocks the console until the client is connected.
- After the successful connection of client, it returns the instance of Socket at server-side.

TCP/IP Client Socket
Socket class

- A socket is simply an endpoint for communications between the machines.
- The Socket class can be used to create a socket.

Methods:

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

TCP/IP Server Socket
ServerSocket class

- The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Methods:

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Creating Server:

- To create the server application, we need to create the instance of ServerSocket class.
- Here, we are using 6666 port number for the communication between the client and server.
- You may also choose any other port number.
- The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

```
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection and waits for the client
```

Creating Client:

- To create the client application, we need to create the instance of Socket class.
- Here, we need to pass the IP address or hostname of the Server and a port number.
- Here, we are using "localhost" because our server is running on same system.

```
Socket s=new Socket("localhost",6666);
```

URL

- The **Java URL** class represents an URL.
- URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web.
- For example: https://www.javatpoint.com/java-tutorial

- A URL contains four information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

Constructors of URL class

- **URL(String spec)**
  - ✓ Creates an instance of a URL from the String representation.
- **URL(String protocol, String host, int port, String file)**
  - ✓ Creates an instance of a URL from the given protocol, host, port number, and file.
- **URL(String protocol, String host, int port, String file, URLStreamHandler handler)**
  - ✓ Creates an instance of a URL from the given protocol, host, port number, file, and handler.
- **URL(String protocol, String host, String file)**

- ✓ Creates an instance of a URL from the given protocol name, host name, and file name.

- **URL(URL context, String spec)**
  - ✓ Creates an instance of a URL by parsing the given spec within a specified context.
- **URL(URL context, String spec, URLStreamHandler handler)**
  - ✓ Creates an instance of a URL by parsing the given spec with the specified handler within a given context.

Methods of URL class

- The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	**Description**
public String getProtocol()	It returns the protocol of the URL.
public String getHost()	It returns the host name of the URL.
public String getPort()	It returns the Port Number of the URL.
public String getFile()	It returns the file name of the URL.
public String getAuthority()	It returns the authority of the URL.
public String toString()	It returns the string representation of the URL.
public String getQuery()	It returns the query string of the URL.
public String getDefaultPort()	It returns the default port of the URL.
public URLConnection openConnection()	It returns the instance of URLConnection i.e. associated with this URL.
public boolean equals(Object obj)	It compares the URL with the given object.
public Object getContent()	It returns the content of the URL.
public String getRef()	It returns the anchor or reference of the URL.
public URI toURI()	It returns a URI of the URL.

Example of URL class

```
//URLDemo.java
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{  URL url=new URL("http://www.javatpoint.com/java-tutorial");
System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial
```

URLConnection class

- The **Java URLConnection** class represents a communication link between the URL and the application.
- This class can be used to read and write data to the specified resource referred by the URL.

Object of URLConnection class

- The openConnection() method of URL class returns the object of URLConnection class.

  Syntax:**public** URLConnection openConnection()**throws** IOException{}

- The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method.
- The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of URLConnection class

```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon=url.openConnection();
InputStream stream=urlcon.getInputStream();
int i;
while((i=stream.read())!=-1){
System.out.print((char)i);
}
}catch(Exception e){System.out.println(e);}
}
}
```

HttpURLConnection class

- The **Java HttpURLConnection** class is http specific URLConnection. It works for HTTP protocol only.
- By the help of HttpURLConnection class, you can information of any HTTP URL such as header information, status code, response code etc.
- The java.net.HttpURLConnection is subclass of URLConnection class.

Object of HttpURLConnection class

- The openConnection() method of URL class returns the object of URLConnection class.
  Syntax:**public** URLConnection openConnection()**throws** IOException{}

- You can typecast it to HttpURLConnection type as given below.

```
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
```

HttpURLConnecton Example

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
for(int i=1;i<=8;i++){
```

```
System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
}
huc.disconnect();
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Date = Wed, 10 Dec 2014 19:31:14 GMT
Set-Cookie = JSESSIONID=D70B87DBB832820CACA5998C90939D48; Path=/
Content-Type = text/html
Cache-Control = max-age=2592000
Expires = Fri, 09 Jan 2015 19:31:14 GMT
Vary = Accept-Encoding,User-Agent
Connection = close
Transfer-Encoding = chunked
```

Datagrams

- Datagrams are bundles of information passed between machines.

- Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

DatagramSocket class

- **Java DatagramSocket** class represents a connection-less socket for sending and receiving datagram packets.
- A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

DatagramPacket class

- **Java DatagramPacket** is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed

Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

```
//DSender.java
import java.net.*;
public class DSender{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Welcome java";
    InetAddress ip = InetAddress.getByName("127.0.0.1");

    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}
```

Example of Receiving DatagramPacket by DatagramSocket

```
//DReceiver.java
import java.net.*;
public class DReceiver{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

Client Server Application using 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.

Stub

- The stub is an object, acts as a gateway for the client side.
- All the outgoing requests are routed through it.
- It resides at the client side and represents the remote object.
- When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

- The skeleton is an object, acts as a gateway for the server side object.
- All the incoming requests are routed through it.
- When the skeleton receives the incoming request, it does the following tasks:
    1. It reads the parameter for the remote method
    2. It invokes the method on the actual remote object, and
    3. It writes and transmits (marshals) the result to the caller.

-

- In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.
- If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application has all these features, so it is called the distributed application.

RMI Example

The following 6 steps given are to write the RMI program:

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

- In this example, we have followed all the 6 steps to create and run the rmi application.
- The client application need only two files, remote interface and client application.
- In the rmi application, both client and server interacts with the remote interface.
- The client application invokes methods on the proxy object, RMI sends the request to the remote JVM.
- The return value is sent back to the proxy object and then to the client application.

1) Create the remote interface

- For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface.
- Here, we are creating a remote interface that extends the Remote interface.
- There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the remote interface

- Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to
  - Either extend the UnicastRemoteObject class, (or)
  - Use the exportObject() method of the UnicastRemoteObject class
- In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) Create the stub and skeleton objects using the rmic tool

- Next step is to create stub and skeleton objects using the rmi compiler.
- The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

- Now start the registry service by using the rmiregistry tool.
- If you don't specify the port number, it uses a default port number.
- In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

- Now rmi services need to be hosted in a server process.
- The Naming class provides methods to get and store the remote object.
- The Naming class provides 5 methods.

public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It returns the reference of the remote object.
public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It binds the remote object with the given name.
public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;	It destroys the remote object which is bound with the given name.
public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;	It binds the remote object to the new name.
public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;	It returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

- At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object.
-

- In this example, we are running the server and client applications, in the same machine so we are using localhost.
- If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

For running **this** rmi example,

1. Compile all the java files: javac *.java
2. Create stub and skeleton object by rmic tool: rmic AdderRemote
3. Start rmi registry in one command prompt: rmiregistry 5000
4. Start the server in another command prompt: java MyServer
5. Start the client application in another command prompt: java MyClient
6. Display the Output

Unit III – Java Beans

- Bean Development Kit
- Jar Files
- Introspection
- Design Pattern for properties, events and methods
- Constrained Properties
- Persistence
- Customizers

Introduction to Java Beans

- Software components are self-contained software units developed according to the motto "Developed them once, run and reused them everywhere" Or in other words, reusability is the main concern behind the component model.
- A software component is a reusable object that can be plugged into any target software application. You can develop software components using various programming languages, such as C, C++, Java, and Visual Basic.
- A "Bean" is a reusable software component model based on sun's java bean specification that can be manipulated visually in a builder tool.
- The term software component model describe how to create and use reusable software components to build an application
- Builder tool is nothing but an application development tool which lets you both to create new beans or use existing beans to create an application.
- To enrich the software systems by adopting component technology JAVA came up with the concept called Java Beans.
- Java provides the facility of creating some user defined components by means of Bean programming.
- We create simple components using java beans. We can directly embed these beans into the software.

Advantages of Java Beans

- The java beans possess the property of "Write once and run anywhere".
- Beans can work in different local platforms.
- Beans have the capability of capturing the events sent by other objects and vice versa enabling object communication.
- The properties, events and methods of the bean can be controlled by the application developer.(ex. Add new properties)
- Beans can be configured with the help of auxiliary software during design time.(no hassle at runtime)
- The configuration setting can be made persistent.(reused)
- Configuration setting of a bean can be saved in persistent storage and restored later.

What can we do/create by using Java Bean

There is no restriction on the capability of a Bean.

- It may perform a simple function, such as checking the spelling of a documet, or a complex function, such as forecasting the performance of a stock portfolio. A Bean may be visible to an end user. One example of this is a button on a graphical user interface.

- Software to generate a pie chart from a set of data points is an example of a Bean that can execute locally.
- Bean that provides real-time price information from a stock or commodities exchange.

Definition of a builder tool

- Builder tools allow a developer to work with JavaBeans in a convenient way.
- By examining a Java Bean by a process known as Introspection, a builder tool exposes the discovered features of the Java Bean for visual manipulation.
- A builder tool maintains a list of all JavaBeans available.
- It allows you to compose the Bean into applets, application, servlets and composite components (e.g. a JFrame), customize its behavior and appearance by modifying its properties and connect other components to the event of the Bean or vice versa.

Some Examples of Application Builder tools

TOOL	***VENDOR***	***DESCRIPTION Java***
Java Workshop2.0	Sun Micro Systems., Inc.,	Complete IDE that support applet, application and bean Development
Visual age for java	IBM	Bean Oriented visual development toolset.
Jbuilder	Borland Inc.	Suit of bean oriented java development tool
Beans Development Kit	Sun Micro Systems., Inc.,	Supports only Beans development

Java Beans Basic rules

- A Java Bean should:
  - be public
  - implement the serializable interface
  - have a no-arg constructor
  - be derived from javax.swing.JComponent or java.awt.Component, if it is visual
- The classes and interfaces defined in the java.beans package enable you to create JavaBeans. The Java Bean components can exist in one of the following three phases of development
  - Construction phase Build phase Execution phase
  - It supports the standard component architecture features of,
  - Properties
  - Events
  - Methods
  - Persistence
- In addition Java Beans provides support for
  - Introspection (Allows Automatic Analysis of a java beans)
  - Customization (To make it easy to configure a java beans component)

Services of Java Bean Components

- **Builder support:** Enables you to create and group multiple JavaBeans in an application**.**
- **Layout:** Allows multiple JavaBeans to be arranged in a development environment.
- **Interface publishing:** Enables multiple JavaBeans in an application to communicate with each other.
- **Event handling:** Refers to firing and handling of events associated with a JavaBean.
- **Persistence**: Enables you to save the last state of JavaBean

Features of a Java Bean

- Support for "introspection" so that a builder tool can analyze how a bean works.
- Support for "customization" to allow the customisation of the appearance and behaviour of a bean.
- Support for "events" as a simple communication metaphor than can be used to connect up beans.
- Support for "properties", both for customization and for programmatic use.
- Support for "persistence", so that a bean can save and restore its customized state.

Bean Development Kit

Is a development environment to create, configure, and test Java Beans. The features of BDK environment are:

- Provides a GUI to create, configure, and test JavaBeans.
- Enables you to modify JavaBean properties and link multiple JavaBeans in an application using BDK.
- Provides a set of sample JavaBeans.
- Enables you to associate pre-defined events with sample JavaBeans.

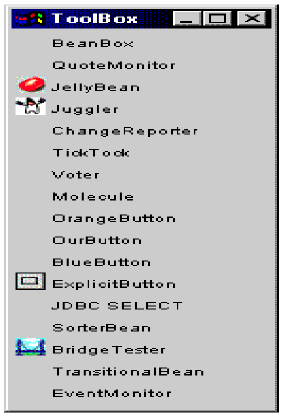
Identifying BDK Components

- Execute the run.bat file of BDK to start the BDK development environment.

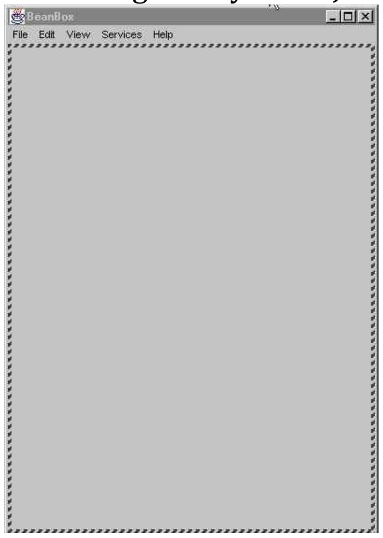
Components of BDK development environment

1. ToolBox 2. BeanBox 3. Properties 4. Method Trace

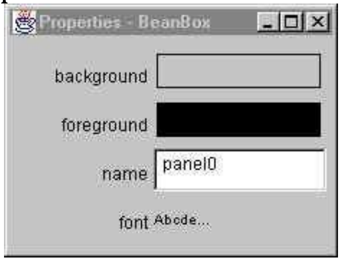
1.Tool Box window: Lists the sample JavaBeans of BDK.



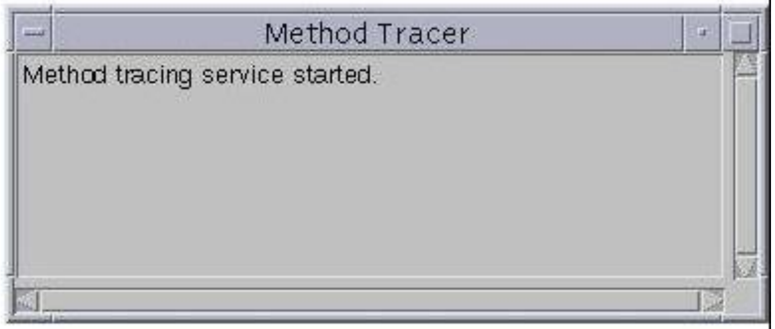
Bean Box window: Is a workspace for creating the layout of Java Bean application.



Properties window: Displays all the exposed properties of a JavaBean. You can modify JavaBean properties in the properties window.



Method Tracer window: Displays the debugging messages and method calls for a JavaBean application.



Method tracing service started.

Steps to Develop a User-Defined Java Bean

1. Create a directory for the new bean
2. Create the java bean source file(s)
3. Compile the source file(s)
4. Create a manifest file
5. Generate a JAR file
6. Start BDK
7. Load Jar file
8. Test.

1. Create a directory for the new bean

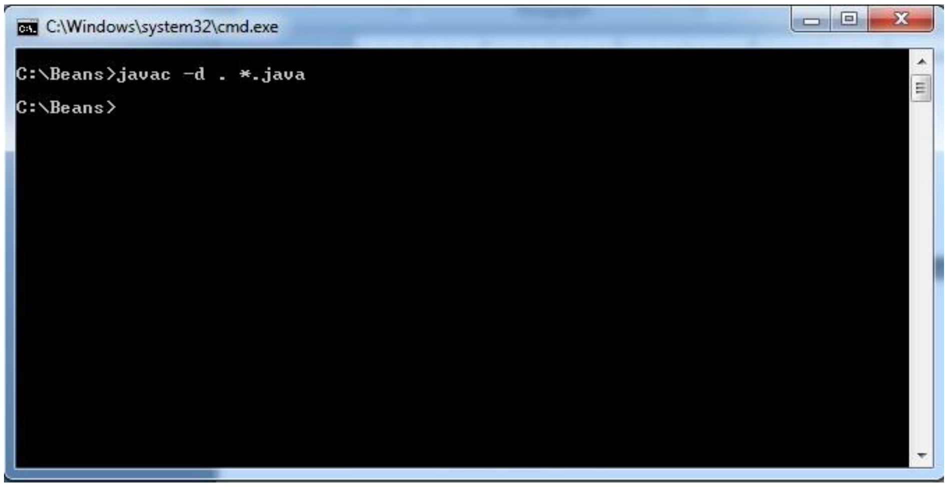
- Create a directory/folder like C:\Beans

2. Create bean source file

```
        package
        com.cmrcet.yellaswamy.beans;
        import java.awt.*;
        public class MyBean extends Canvas
        {
        public MyBean()
        {
        setSize(70,50);
        setBackground(Color.
        green);
        }
}
```

3. Compile the source file(s)

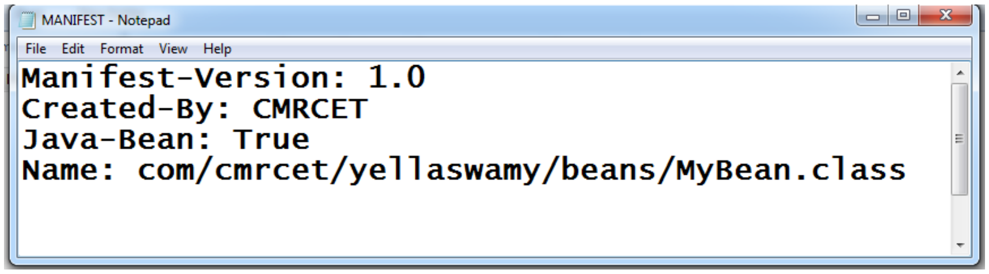
- C:\Beans >javac –d . *.java



```
C:\Beans>javac -d . *.java
C:\Beans>
```

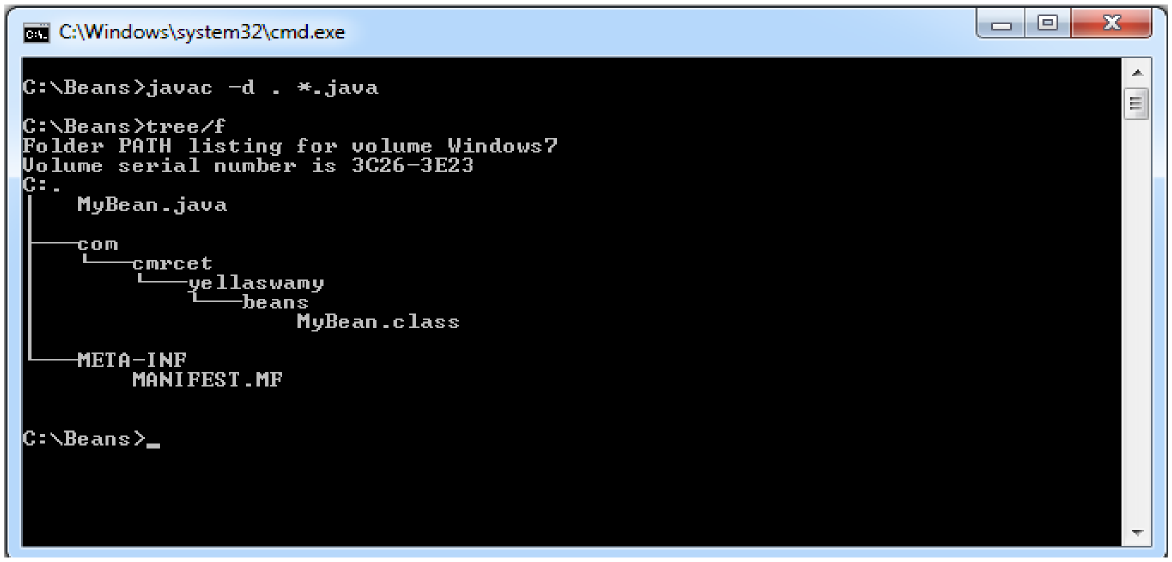
4. Create a manifest file Manifest File

- The manifest file for a JavaBean application contains a list of all the class files that make up a JavaBean.
- The entry in the manifest file enables the target application to recognize the JavaBean classes for an application.
- For example, the entry for the MyBean JavaBean in the manifest file is as shown:



```
Manifest-Version: 1.0
Created-By: CMRCET
Java-Bean: True
Name: com/cmrcet/yellaswamy/beans/MyBean.class
```

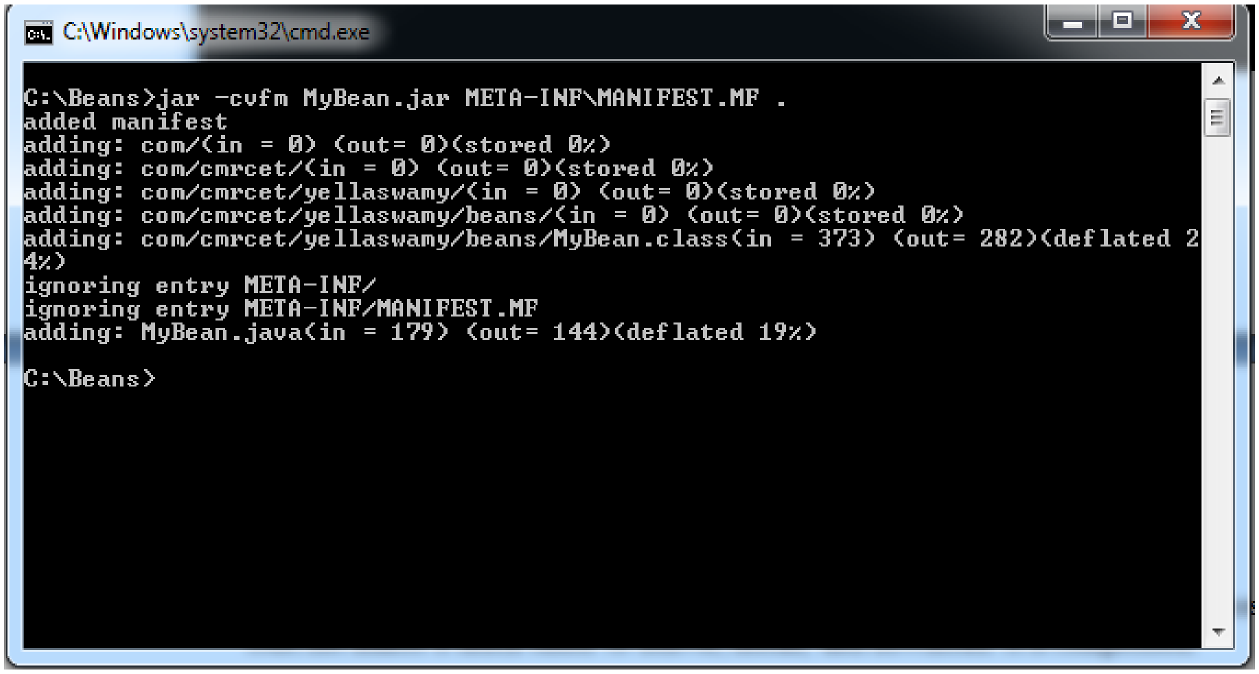
- Write that 2 lines code in the notepad and save that file as MANIFEST.MF in META-INF directory
- The rules to create a manifest file are:
  - ✓ Press the Enter key after typing each line in the manifest file.
  - ✓ Leave a space after the colon.
  - ✓ Type a hyphen between Java and Bean.
  - ✓ No blank line between the Name and the Java-Bean entry.



```
C:\Beans>javac -d . *.java

C:\Beans>tree/f
Folder PATH listing for volume Windows7
Volume serial number is 3C26-3E23
C:.
│   MyBean.java
│
├───com
│   └───cmrcet
│       └───yellaswamy
│           └───beans
│                   MyBean.class
│
└───META-INF
        MANIFEST.MF


C:\Beans>_
```

5. Generate a JAR file

- ○ Syntax for creating jar file using manifest file
- ○ C:\Benas>jar –cvfm MyBean.jar META-INF\MANIFEST.MF.



```
C:\Beans>jar -cvfm MyBean.jar META-INF\MANIFEST.MF .
added manifest
adding: com/(in = 0) (out= 0)(stored 0%)
adding: com/cmrcet/(in = 0) (out= 0)(stored 0%)
adding: com/cmrcet/yellaswamy/(in = 0) (out= 0)(stored 0%)
adding: com/cmrcet/yellaswamy/beans/(in = 0) (out= 0)(stored 0%)
adding: com/cmrcet/yellaswamy/beans/MyBean.class(in = 373) (out= 282)(deflated 2
4%)
ignoring entry META-INF/
ignoring entry META-INF/MANIFEST.MF
adding: MyBean.java(in = 179) (out= 144)(deflated 19%)

C:\Beans>
```

***JAR file* (Java Archive)**

- JAR file allows you to efficiently deploy a set of classes and their associated resources.
- JAR file makes it much easier to deliver, install, and download. It is compressed.
- The files of a JavaBean application are compressed and grouped as JAR files to reduce the size and the download time of the files.

- The syntax to create a JAR file from the command prompt is:

  ✓ jar <options><file_names>

- The file_names is a list of files for a JavaBean application that are stored in the JAR file.

The various options that you can specify while creating a JAR file are:

✓ c: Indicates the new JAR file is created.

✓ f: Indicates that the first file in the file_names list is the name of the JAR file.

✓ m: Indicates that the second file in the file_names list is the name of the manifest file.

✓ t: Indicates that all the files and resources in the JAR file are to be displayed in a tabular format.

✓ v: Indicates that the JAR file should generate a verbose output.

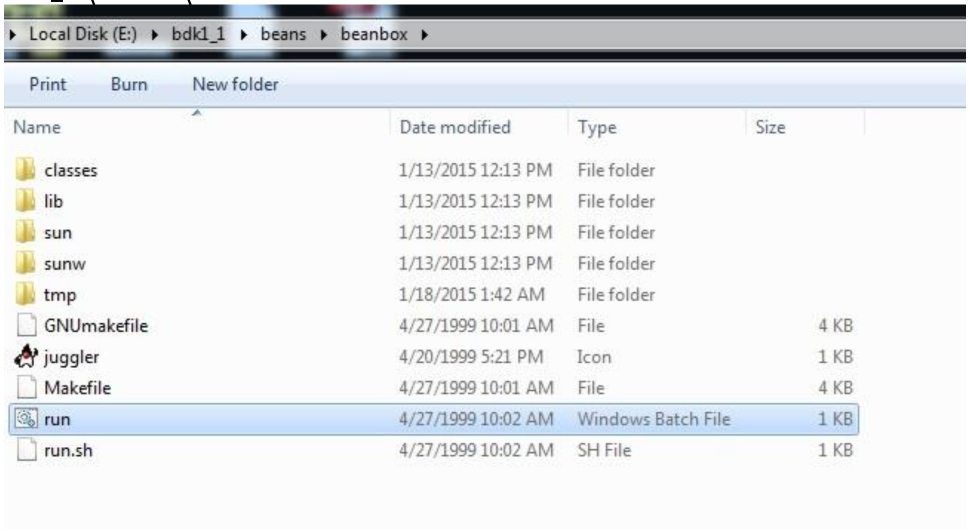
✓ x: Indicates that the files and resources of a JAR file are to be extracted.

✓ o: Indicates that the JAR file should not be compressed.

✓ m: Indicates that the manifest file is not created.

6. Start BDK

- Go to C:\bdk1_1\beans\beanbox Click on run.bat file.



Local Disk (E:) ▸ bdk1_1 ▸ beans ▸ beanbox ▸

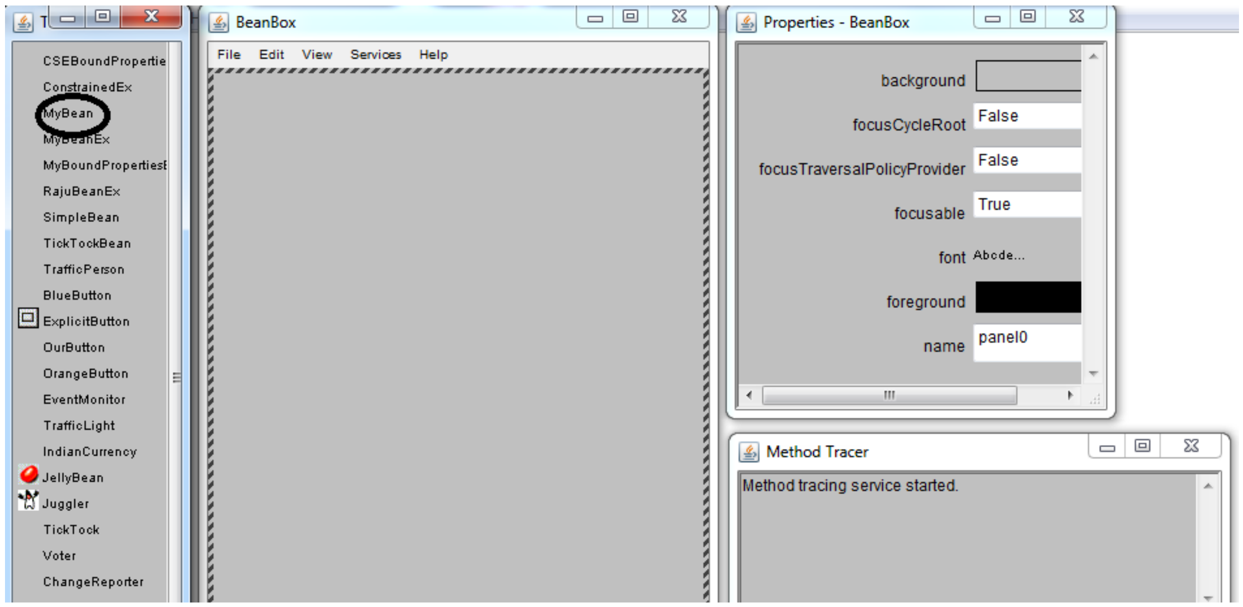
Print Burn New folder

Name	Date modified	Type	Size
classes	1/13/2015 12:13 PM	File folder	
lib	1/13/2015 12:13 PM	File folder	
sun	1/13/2015 12:13 PM	File folder	
sunw	1/13/2015 12:13 PM	File folder	
tmp	1/18/2015 1:42 AM	File folder	
GNUmakefile	4/27/1999 10:01 AM	File	4 KB
juggler	4/20/1999 5:21 PM	Icon	1 KB
Makefile	4/27/1999 10:01 AM	File	4 KB
run	4/27/1999 10:02 AM	Windows Batch File	1 KB
run.sh	4/27/1999 10:02 AM	SH File	1 KB

- When we click on run.bat file the BDK software automatically started.

7. Load Jar file Go to

- Beanbox->File->Load jar
- Here we have to select our created jar file when we click on ok.
- our bean(userdefined) MyBean appear in the ToolBox.

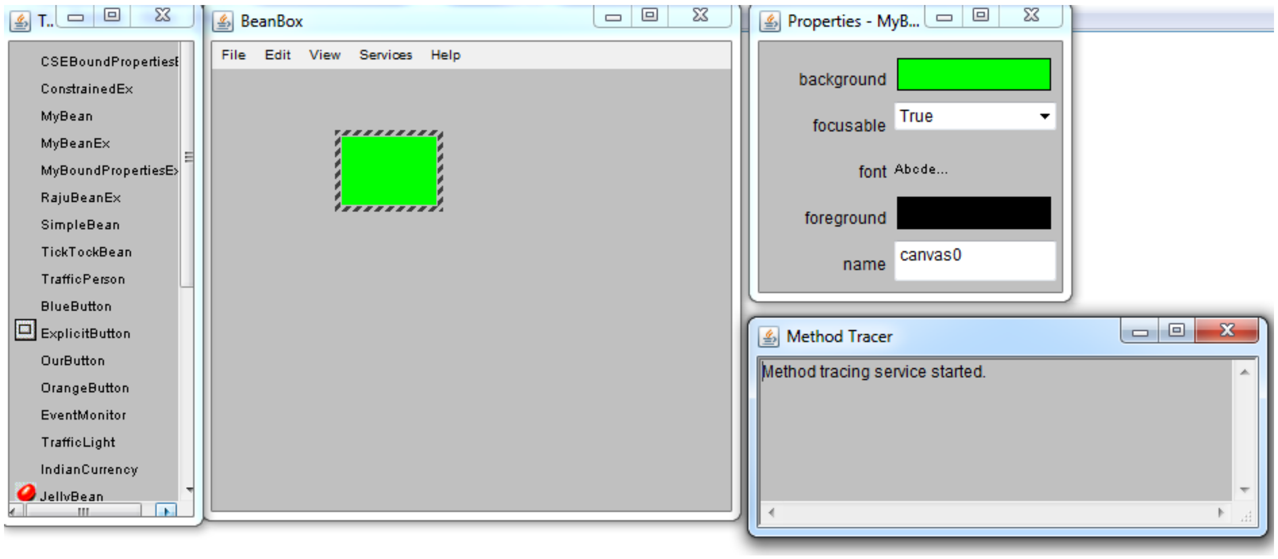


background
focusCycleRoot False
focusTraversalPolicyProvider False
focusable True
font Abcde...
foreground
name panel0

Method tracing service started.

8. Test our created user defined bean

- Select the MyBean from the ToolBox when we select that bean one + simple appear then drag that Bean in to the Beanbox.
- If you want to apply events for that bean, now we apply the events for that Bean.



background
focusable True
font Abcde...
foreground
name canvas0

Method tracing service started.

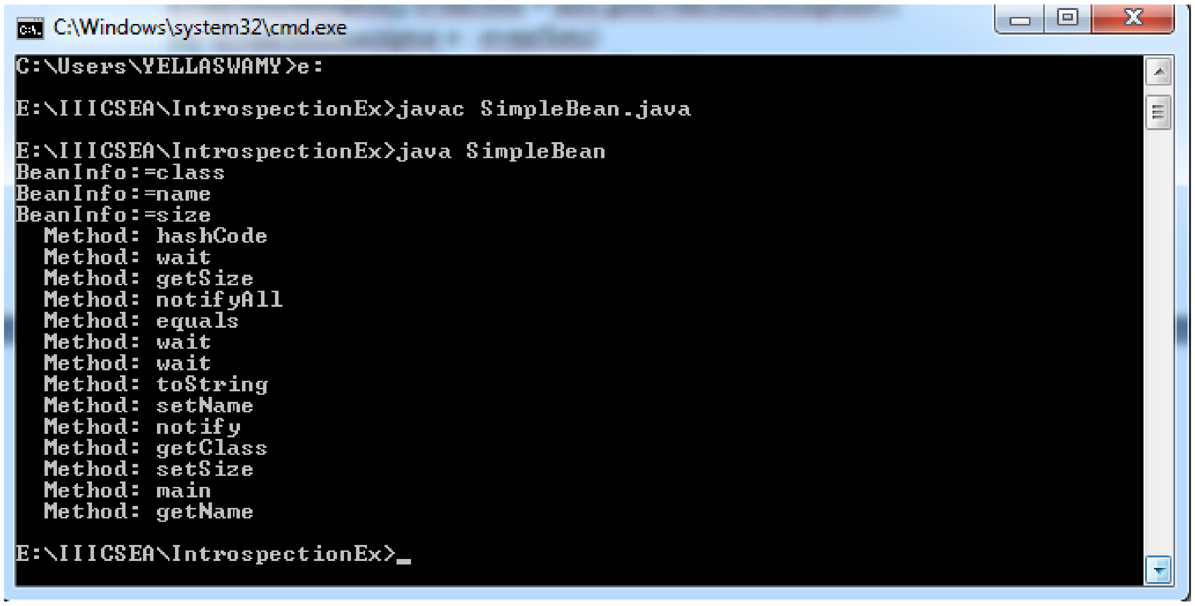
Introspection

- Introspection can be defined as the technique of obtaining information about bean properties, events and methods.
- Basically introspection means analysis of bean capabilities.
- Introspection is the automatic process by which a builder tool finds out which properties, methods, and events a bean supports.
- Introspection describes how methods, properties, and events are discovered in the beans that you write.
- This process controls the publishing and discovery of bean operations and properties without introspection, the JavaBeans technology could not operate.
- Basically introspection means analysis of bean capabilities.
- There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by builder tool.
- The first method, simple naming conventions are used. These allow the introspection mechanisms to infer information about a Bean.
- In the second way, an additional class is provided that explicitly supplies this information.

SimpleBean.java

```
//Introspection
import java.beans.BeanInfo;
import java.beans.IntrospectionException;
import java.beans.Introspector;
import java.beans.*;
import java.beans.PropertyDescriptor;
public class SimpleBean
{
private String
name="CMRCET"; private
int Size;
public String getName()
{
return this.name;
}
public int getSize()
{
return this.Size;
}
public void setSize(int size)
{
this.Size=size;
}
public void setName(String name)
{
this.name=name;
}
public static void main(String args[])throws IntrospectionException
{
BeanInfo info=Introspector.getBeanInfo(SimpleBean.class);
for(PropertyDescriptor pd:info.getPropertyDescriptors())
{
System.out.println("BeanInfo:="+pd.getName());
}

MethodDescriptor[] methods = info.getMethodDescriptors();

for (MethodDescriptor m : methods)
        System.out.println(" Method: " + m.getName());
        EventSetDescriptor[] eventSets = info.getEventSetDescriptors();
        for (EventSetDescriptor e : eventSets)
        System.out.println(" Event: " + e.getName());
} }
```

Output:



C:\Windows\system32\cmd.exe

```
C:\Users\YELLASWAMY>e:

E:\IIICSEA\IntrospectionEx>javac SimpleBean.java

E:\IIICSEA\IntrospectionEx>java SimpleBean
BeanInfo:=class
BeanInfo:=name
BeanInfo:=size
  Method: hashCode
  Method: wait
  Method: getSize
  Method: notifyAll
  Method: equals
  Method: wait
  Method: wait
  Method: toString
  Method: setName
  Method: notify
  Method: getClass
  Method: setSize
  Method: main
  Method: getName

E:\IIICSEA\IntrospectionEx>_
```

Design patterns for Java Bean Properties

- A property is a subset of a Bean's state. A bean property is a named attribute of a bean that can affect its behavior or appearance.
- Examples of bean properties include color, label, font, font size, and display size.
- Properties are the private data members of the Java Bean classes.
- Properties are used to accept input from an end user in order to customize a Java Bean.
- Properties can retrieve and specify the values of various attributes, which determine the behavior of a Java Bean.

Types of JavaBeans Properties

- Simple properties
- Boolean properties
- Indexed properties

- Bound Properties
- Constrained Properties

Simple Properties

- Simple properties refer to the private variables of a JavaBean that can have only a single value. Simple properties are retrieved and specified using the get and set methods respectively.

  ✓ A read/write property has both of these methods to access its values. The get method used to read the value of the property .The set method that sets the value of the property.

  ✓ The setXXX() and getXXX() methods are the heart of the java beans properties mechanism. This is also called getters and setters. These accessor methods are used to set the property .

  ✓ The syntax of get method is,

    - public return_type get<PropertyName>()
    - public T getN();
    - public void setN(T arg) N is the name of the property and T is its type

Example
```
public double getDepth()
 {
return depth;
 }
```

- Read only property has only a get method.

  ✓ The syntax of set method is

    - public void set<PropertyName>(data_type value)

Example
```
public void setDepth(double d)
{
 Depth=d;
}
```

- Write only property has only a set method.

Boolean Properties

- A Boolean property is a property which is used to represent the values True or False.
- Have either of the two values, TRUE or FALSE. It can identified by the following methods:

Syntax

- Let N be the name of the property and T be the type of the value then

- publicbooleanisN();
  public void setN(boolean
  parameter); public Boolean
  getN(); publicboolean
  is<PropertyName>()
  publicboolean
  get<PropertyName>()

- First or second pattern can be used to retrieve the value of a Boolean.
  - ✓ public void set<PropertyName>(boolean value)
- For getting the values isN() and getN() methods are used and for setting the Boolean values setN() method is used.

Example

```
publicboolean
dotted=false;
publicbooleanisDotted()
{ return dotted; }
public void setDotted(boolean
dotted) { this.dotted=dotted; }
```

Indexed Properties

- Indexed Properties are consists of multiple values. If a simple property can hold an array of value they are no longer called simple but instead indexed properties.
- The method's signature has to be adapted accordingly. An indexed property may expose set/get methods to read/write one element in the array (so-called 'index getter/setter') and/or so-called 'array getter/setter' which read/write the entire array.
- Indexed Properties enable you to set or retrieve the values from an array of property values.
- Indexed Properties are retrieved using the following get methods:

  - Syntax: publicint[] get<PropertyName>()

Example

```
private double data[];
public double getData(int index)
   {
return data[index];
      }
```

- Syntax:public property_datatype get<PropertyName>(int index)

Example

```
public void setData(intindex,double value)
{
```

Data[index]=value;
}
Indexed Properties are specified using the following set methods:

- Syntax: public void set<PropertyName>(int index, property_datatype value)

Example
public double[] getData()
{
return data;
}

- Syntax : public void set<PropertyName>(property_datatype[] property_array)

Example
public void setData(double[] values)
{
}

- The properties window of BDK does not handle indexed properties.
- Hence the output cannot be displayed here**.**

Bound Properties

- A bean that has a bound property generates an event when the property is changed.
- Bound Properties are the properties of a JavaBean that inform its listeners about changes in its values.
- Bound Properties are implemented using the PropertyChangeSupport class and its methods. Bound Properties are always registered with an external event listener.
- The event is of type PropertyChangeEvent and is sent to objects that previously registered an interest in receiving such notifications bean with bound property
- Event source Bean implementing listener -- event target.
- In order to provide this notification service a JavaBean needs to have the following two methods:
    - public void addPropertyChangeListener(PropertyChangeListener p)
      {
      changes.addPropertyChangeListener(p);
      }
    - public void removePropertyChangeListener(PropertyChangeListener p)
      {
      changes.removePropertyChangeListener(p);
      }
- PropertyChangeListener is an interface declared in the java.beans package.
- Observers which want to be notified of property changes have to implement this interface, which consists of only one method:
    - public interface PropertyChangeListener extends EventListener
      {
      public void propertyChange(PropertyChangeEvent e);
      }

Constrained Properties

- It generates an event when an attempt is made to change it value Constrained Properties are implemented using the PropertyChangeEvent class.
- The event is sent to objects that previously registered an interest in receiving an such notification Those other objects have the ability to veto the proposed change This allows a bean to operate differently according to the runtime environment
- A bean property for which a change to the property results in validation by another bean. The other bean may reject the change if it is not appropriate.
- Constrained Properties are the properties that are protected from being changed by other JavaBeans.
- Constrained Properties are registered with an external event listener that has the ability to either accept or reject the change in the value of a constrained property.
- Constrained Properties can be retrieved using the get method.
- The prototype of the get method is:
  - **Syntax: public string get<ConstrainedPropertyName>()**
- The prototype of the set method is:
  - **Syntax: public string set<ConstrainedPropertyName>(String str) throws PropertyVetoException**
- There are three parts to constrained property Implementations:
  - A source bean containing one or more constrained properties
  - A PropertyChangeEvent object containing the property name,and its old and new values.This is the same class used for bound properties.
  - Listeners objects that implements the VetoableChangeListener interface.
  - This object accepts or rejects proposed changes to a constrained property in the source Bean.
  - A bean Contaiing Constrained properties must allow VetoableChangeListener objects to register and unregister their interest in receiving notification that a property change is proposed.
  - Fire property change events at those interested listeners when a property change is proposed.The event should be fired before the actual property change takes place. This gives each listener a chance to veto the proposed change. The PropertyChangeEvent is fired by a call to each listeners vetoableChange() method.
  - If a listener vetoes, then make sure that any other listeners can revert to the old value.

Steps to implement Constrained Property

1. Add import Statement to support Property Change Events
2. Instantiate a VetoableChangeSupport object
3. Add code to Fire,When a PropertyChangeEvent when the property is changed.
4. Implement VetoableChangeSupport methods to add or remove listeners
5. Build the jar file and Install it in the Bean Box
6. Test this bean in BeanBox same as boundproperties

```
// ConstrainedEx.java
package com.yellaswamy.constrainedexample;

//step1
import java.awt.*;
import java.beans.*;
public class ConstrainedEx extends Canvas
{
//Step2

String price="100";
VetoableChangeSupport vcs=new
VetoableChangeSupport(this); private PropertyChangeSupport
pcs=new PropertyChangeSupport(this);

//constructor
public ConstrainedEx()
{
setBackground(Color.red);
setForeground(Color.blue);
}

public void setString(String newprice)
{
//Step3

String
oldprice=price;
price=newprice
;
pcs.firePropertyChange("price","oldprice","newprice");
}

public String getString()
{
return price;
}

public Dimension getMinimumSize()
{
return new Dimension(100,100);
}

//step4
public void addVetoableChangeListener(VetoableChangeListener vcl)
{
vcs.addVetoableChangeListener(vcl);
}
public void removeVetoableChangeListener(VetoableChangeListener vcl)
{
vcs.removeVetoableChangeListener(vcl);
}
}
```

Design Patterns for Events

Handling Events in JavaBeans

- Enables Beans to communicate and connect together. Beans generate events and these events can be sent to other objects. Event means any activity that interrupts the current ongoing activity.
- Example: mouse clicks, pressing key…
- User-defined JavaBeans interact with the help of user-defined events, which are also called custom events. You can use the Java event delegation model to handle these custom events.
- The components of the event delegation model are:

✓ **Event Source:** Generates the event and informs all the event listeners that are registered with it
✓ **Event Listener:** Receives the notification, when an event source generates an event
✓ **Event Object:** Represents the various types of events that can be generated by the event sources.

Creating Custom Events

- The classes and interfaces that you need to define to create the custom JavaBean events are:
  - An event class to define a custom JavaBean event.
  - An event listener interface for the custom JavaBean event.
  - An event handler to process the custom JavaBean event.
- A target Java application that implements the custom event.

Creating the Event Class

- The event class that defines the custom event extends the EventObject class of the java.util package.

Example

```
public class NumberEvent extends EventObject
{
publicint number1,number2;
publicNumberEvent(Object o,int number1,int number2)
{
 super(o); this.number1=number1;
 this.number2=number2;
} }
```

- Beans can generate events and send them to other objects.

Creating Event Listeners

- When the event source triggers an event, it sends a notification to the event listener interface.
- The event listener interface implements the java.util.EventListener interface.

Syntax

```
public void addTListener(TListenereventListener);
public void addTListener(TListenereventListener)throws TooManyListeners;
public void removeTListener(TListenereventListener);
```

- The target application that uses the custom event implements the custom listener.

Example

```
public interface NumberEnteredListener extends EventListener
{
public void arithmeticPerformed(NumberEventmec);
 }
```

Creating Event Handler

- Custom event handlers should define the following methods:

  - **addXXListener():** Registers listeners of a JavaBean event.
  - **fireXX():** Notifies the listeners of the occurrence of a JavaBean event.
  - **removeXXListener():** Removes a listener from the list of registered listeners of a JavaBean.

The code snippet to define an event handler for the custom event NumberEvent

```
public class NumberBean extends JPanel implements ActionListener
 {
publicNumberBean()
   {}
NumberEnteredListenermel;
public void addNumberListener(NumberEnteredListenermel)
  {
this.mel = mel;
  }
 }
```

Persistence

- Persistence means an ability to save properties and events of our beans to non-volatile storage and retrieve later.
- It has the ability to save a bean to storage and retrieve it at a later time Configuration settings are saved It is implemented by Java serialization.
- If a bean inherits directly or indirectly from Component class it is automatically Serializable.

1. Enables developers to customize Beans in an application builder, and then retrieve those Beans, with customized features intact, for future use,perhaps in another environment.
2. Java Beans supports two forms of persistence:
    a. Automatic persistence
    b. External persistence

a. Automatic Persistence: Automatic persistence are java's built-in serialization mechanism to save and restore the state of a bean.

b. External Persistence: External persistence, on the other hand, gives you the option of supplying your own custom classes to control precisely how a bean state is stored and retrieved.

1. Juggle Bean.
2. Building an applet
3. Your own bean

Customizers

- The Properties window of the BDK allows a developer to modify the several properties of the Bean.
- Property sheet may not be the best user interface for a complex component.
- It can provide step-by-step wizard guide to use component.
- It can provide a GUI frame with image which visually tells what is changed such as radio button, check box, ...
- It can customize the appearance and behavior of the properties Online documentation can also be provided.
- A Bean developer has great flexibility to develop a customizer that can differentiate his or her product in the marketplace.
- To make it easy to configure a java beans component enables a developer to use an application builder tool to customize the appearance and behavior of a bean.

Unit IV – Java Servlets

- Life Cycle of Servlet
- Generic Servlet
- HTTP Servlet
- Reading Initialization Parameters
- Reading Servlet Parameters
- Cookies
- Session Tracking

Servlets

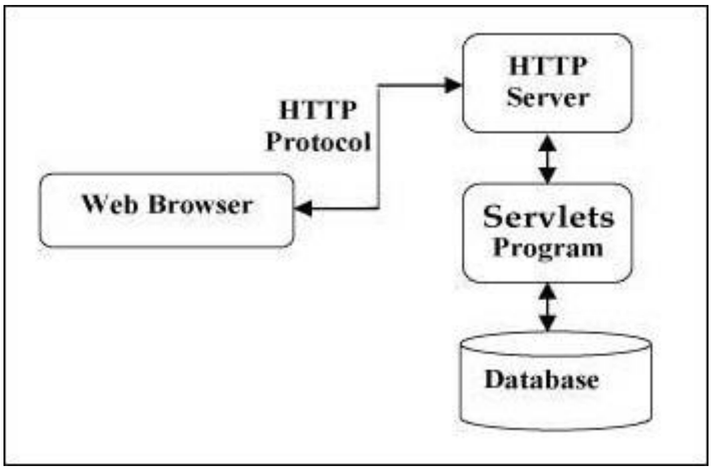
- Servlets provide a component-based, platform-independent method for building Webbased applications, without the performance limitations of CGI programs.
- Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.
- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Requirements of Servlets

- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.
  - ✓ Performance is significantly better.
  - ✓ Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
  - ✓ Servlets are platform-independent because they are written in Java.
  - ✓ Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
  - ✓ The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture

- The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks

Servlets perform the following major tasks –

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages

- Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.
- Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.
- These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.
- Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Servlet Environment

- A development environment is where you would develop your Servlet, test them and finally run them.
- Like any other Java program, you need to compile a servlet by using the Java compiler **javac** and after compilation the servlet application, it would be deployed in a configured environment to test and run..
- This development environment setup involves the following steps:

Setting up Java Development Kit

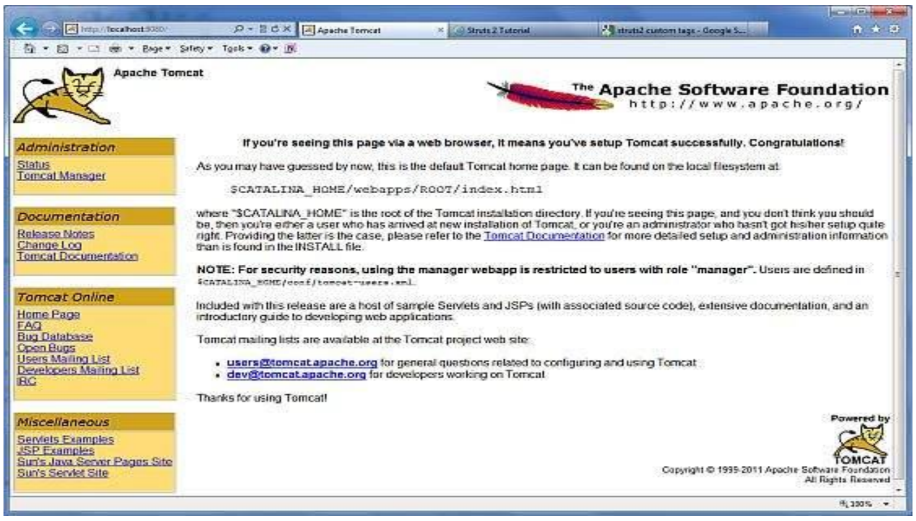
- This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up PATH environment variable appropriately.
- We can download SDK from Oracle's Java site – Java SE Downloads.
- Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.
- If you are running Windows and installed the SDK in C:\jdk1.8.0_65, you would put the following line in your C:\autoexec.bat file.
    - set PATH = C:\jdk1.8.0_65\bin;%PATH%
    - set JAVA_HOME = C:\jdk1.8.0_65

- Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.
- On Unix (Solaris, Linux, etc.), if the SDK is installed in /usr/local/jdk1.8.0_65 and you use the C shell, you would put the following into your .cshrc file.
    - setenv PATH /usr/local/jdk1.8.0_65/bin:$PATH
    - setenv JAVA_HOME /usr/local/jdk1.8.0_65
- Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java.

Setting up Web Server – Tomcat

- A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them.
- Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server.
- The following are the steps to setup Tomcat on your machine:
    - Download latest version of Tomcat from https://tomcat.apache.org/.
    - Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-8.0.28 on windows, or /usr/local/apache-tomcat-8.0.289 on Linux/Unix and create CATALINA_HOME environment variable pointing to these locations.

- Tomcat can be started by executing the following commands on windows machine
    - %CATALINA_HOME%\bin\startup.bat (or)
    - C:\apache-tomcat-8.0.28\bin\startup.bat
- Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine
    - $CATALINA_HOME/bin/startup.sh (or)
    - /usr/local/apache-tomcat-8.0.28/bin/startup.sh
- After startup, the default web applications included with Tomcat will be available by visiting **http://localhost:8080/**. If everything is fine then it should display following result



Apache Tomcat

The Apache Software Foundation
http://www.apache.org/

Administration
Status
Tomcat Manager

Documentation
Release Notes
Change Log
Tomcat Documentation

Tomcat Online
Home Page
FAQ
Bug Database
Open Bugs
Users Mailing List
Developers Mailing List
IRC

Miscellaneous
Servlets Examples
JSP Examples
Sun's Java Server Pages Site
Sun's Servlet Site

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager". Users are defined in $CATALINA_HOME/conf/tomcat-users.xml.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- users@tomcat.apache.org for general questions related to configuring and using Tomcat
- dev@tomcat.apache.org for developers working on Tomcat

Thanks for using Tomcat!

Powered by TOMCAT

Copyright © 1999-2011 Apache Software Foundation
All Rights Reserved

- Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site – http://tomcat.apache.org
- Tomcat can be stopped by executing the following commands on windows machine
    - C:\apache-tomcat-8.0.28\bin\shutdown
- Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine
    - /usr/local/apache-tomcat-8.0.28/bin/shutdown.sh

Setting Up the CLASSPATH

- Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.
- If you are running Windows, you need to put the following lines in your C:\autoexec.bat file.
    - set CATALINA = C:\apache-tomcat-8.0.28
    - setCLASSPATH = %CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%
- Alternatively, on Windows NT/2000/XP, you could go to My Computer –> Properties –> Advanced –> Environment Variables. Then, you would update the CLASSPATH value and press the OK button.
- On Unix (Solaris, Linux, etc.), if you are using the C shell, you would put the following lines into your .cshrc file.
    - setenv CATALINA = /usr/local/apache-tomcat-8.0.28
    - setenv CLASSPATH $CATALINA/common/lib/servlet-api.jar:$CLASSPATH
- Assuming that your development directory is C:\ServletDevel (Windows) or /usr/ServletDevel (Unix) then you would need to add these directories as well in CLASSPATH in similar way as you have added above.

Life cycle of Servlet

- A servlet life cycle can be defined as the entire process from its creation till the destruction.
- The following are the paths followed by a servlet.
    - ✓ The servlet is initialized by calling the **init()** method.
    - ✓ The servlet calls **service()** method to process a client's request.
    - ✓ The servlet is terminated by calling the **destroy()** method.
    - ✓ Finally, servlet is garbage collected by the garbage collector of the JVM.

The init() Method

- The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards.
- So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate.
- The init() method simply creates or loads some data that will be used throughout the life of the servlet.
- The init method definition is as follows:

```
public void init() throws ServletException {
  // Initialization code...
}
```

The service() Method

- The service() method is the main method to perform the actual task.
- The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- The service methd definition is as follows:

```
public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException {
}
```

- The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.
- The doGet() and doPost() are most frequently used methods with in each service request.

The doGet() Method

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  // Servlet code
}
```

The doPost() Method

- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  // Servlet code
}
```

The destroy() Method

- The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection.
- The destroy method definition is as follows:

```
public void destroy() {
  // Finalization code...}
```

Example of Servlet Programming

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet
{
  private String message;
  public void init() throws ServletException {
    // Do required initialization
    message = "Hello World";
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
      // Set response content type
    response.setContentType("text/html");

    // Actual logic goes here.
    PrintWriter out = response.getWriter();
    out.println("<h1>" + message + "</h1>");
  }

  public void destroy()
 {
    // do nothing.
  }
}
```

Servlet API

- The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.
- The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Interfaces in javax.servlet package

- There are many interfaces in javax.servlet package. They are as follows:

• Servlet	ServletRequest	ServletResponse
• RequestDispatcher	ServletConfig	ServletContext
• SingleThreadModel	Filter	FilterConfig
• ServletRequestListener	ServletRequestAttributeListener	
• ServletContextListener	ServletContextAttributeListener	

Classes in javax.servlet package

- There are many classes in javax.servlet package. They are as follows:

• GenericServlet	ServletInputStream	ServletOutputStream
• ServletRequestWrapper	ServletResponseWrapper	ServletRequestEvent
• ServletContextEvent	ServletException	UnavailableException
• ServletRequestAttributeEvent	ServletContextAttributeEvent	

Interfaces in javax.servlet.http package

- There are many interfaces in javax.servlet.http package. They are as follows:

• HttpServletRequest	HttpServletResponse
• HttpSession	HttpSessionListener
• HttpSessionAttributeListener	HttpSessionBindingListener
• HttpSessionContext	HttpSessionActivationListener

Classes in javax.servlet.http package

- There are many classes in javax.servlet.http package. They are as follows:

• HttpServlet	Cookie	HttpUtils
• HttpServletRequestWrapper	HttpServletResponseWrapper	
• HttpSessionEvent	HttpSessionBindingEvent	

GenericServlet class

- **GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces.
- It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

- There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

File: First.java

```
import java.io.*;
import javax.servlet.*;
public class First extends GenericServlet
{
public void service(ServletRequest req,ServletResponse res)throws IOException,ServletException
{
res.setContentType("text/html");
PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");
}
}
```

HttpServlet class

- The HttpServlet class extends the GenericServlet class and implements Serializable interface.
- It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

- There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Reading Initialization parameters

- The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET Method

- The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** (question mark) symbol as follows
    - ✓ http://www.test.com/hello?key1 = value1&key2 = value2
- The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.
- This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

POST Method

- A generally more reliable method of passing information to a backend program is the POST method.
- This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a, ? (question mark) in the URL it sends it as a separate message.

- This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

Reading Servlet Parameters

- Servlets handles form data parsing automatically using the following methods depending on the situation.
  - ✓ **getParameter()** – You call request.getParameter() method to get the value of a form parameter.
  - ✓ **getParameterValues()** – Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
  - ✓ **getParameterNames()** – Call this method if you want a complete list of all parameters in the current request.

GET Method Example using URL

- A simple URL which will pass two values to HelloForm program using GET method.
  - ▪ http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI

Sample Source code for Reading Parameters

- Following is the generic example which uses **getParameterNames()** method of HttpServletRequest to read all the available form parameters.
- This method returns an Enumeration that contains the parameter names in an unspecified order.
- Once we have an Enumeration, we can loop down the Enumeration in standard way by, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Reading All Form Parameters";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor = \"#f0f0f0\">\n" +
      "<h1 align = \"center\">" + title + "</h1>\n" +
      "<table width = \"100%\" border = \"1\" align = \"center\">\n" +
      "<tr bgcolor = \"#949494\">\n" +
        "<th>Param Name</th>"
        "<th>Param Value(s)</th>\n"+
      "</tr>\n"
    );

    Enumeration paramNames = request.getParameterNames();

    while(paramNames.hasMoreElements()) {
      String paramName = (String)paramNames.nextElement();
      out.print("<tr><td>" + paramName + "</td>\n<td>");
      String[] paramValues = request.getParameterValues(paramName);

      // Read single valued data
      if (paramValues.length == 1) {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0)
          out.println("<i>No Value</i>");
          else
          out.println(paramValue);
      } else {
        // Read multiple valued data
        out.println("<ul>");

        for(int i = 0; i < paramValues.length; i++) {
          out.println("<li>" + paramValues[i]);
        }
        out.println("</ul>");
      }
    }
    out.println("</tr>\n</table>\n</body></html>");
  }

  // Method to handle POST method request.
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    doGet(request, response);
  }
}
```

Now, try the above servlet with the following form –

```
<html>
  <body>
    <form action = "ReadParams" method = "POST" target = "_blank">
      <input type = "checkbox" name = "maths" checked = "checked" /> Maths
      <input type = "checkbox" name = "physics"  /> Physics
      <input type = "checkbox" name = "chemistry" checked = "checked" /> Chem
      <input type = "submit" value = "Select Subject" />
    </form>
  </body>
</html>
```

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.

Types of Cookie

- There are 2 types of cookies in servlets:

    1. Non-persistent cookie
    2. Persistent cookie

Non-persistent cookie

- It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

- It is **valid for multiple session**.
- It is not removed each time when user closes the browser.
- It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

- **javax.servlet.http.Cookie** class provides the functionality of using cookies.
- It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	**Description**
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Methods of Cookie class

- There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Methods required for using Cookies

- For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

Creation of Cookie

- The simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

Deletion of Cookie

- The simple code to delete cookie.
- It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

Getting Cookies

- The simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++)
{
out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of
cookie
}
```

Example of Servlet Cookies

```
index.html
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

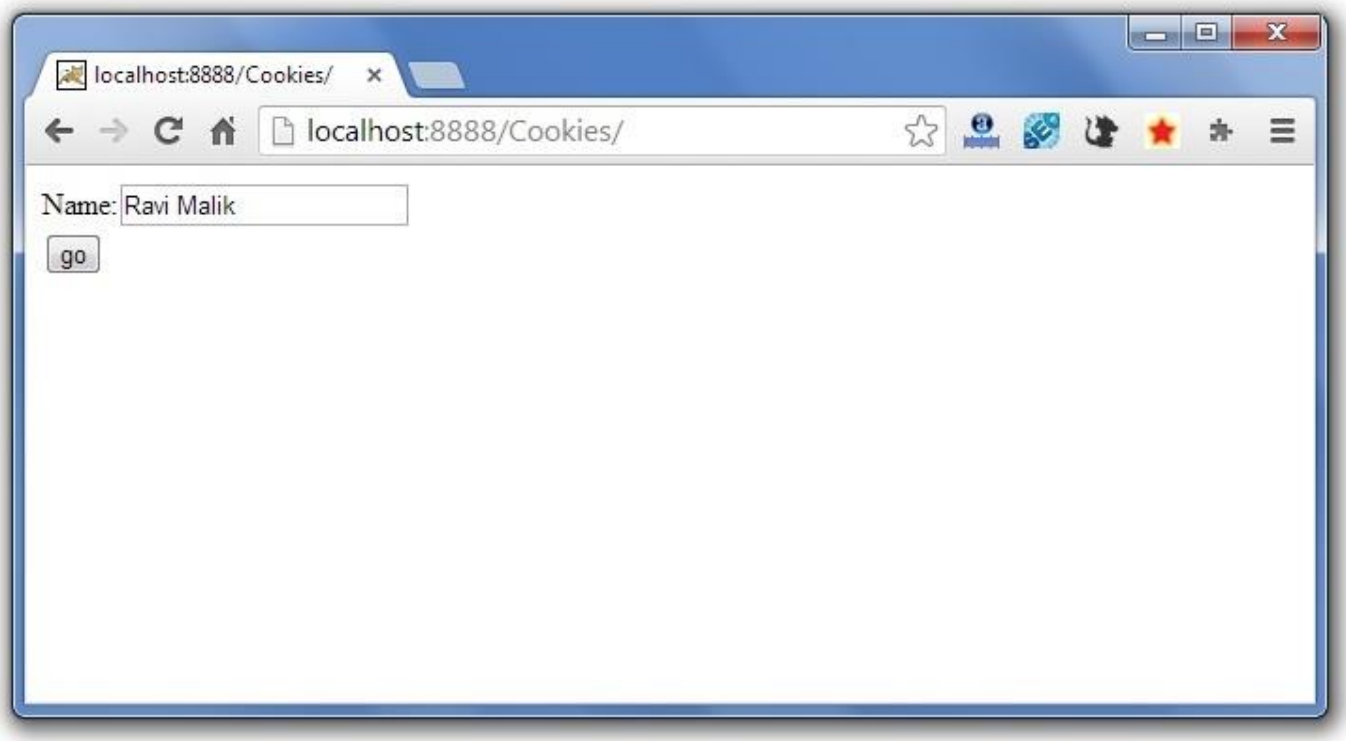
```
FirstServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

  String n=request.getParameter("userName");
  out.print("Welcome "+n);

  Cookie ck=new Cookie("uname",n);//creating cookie object
  response.addCookie(ck);//adding cookie in the response

  //creating submit button
  out.print("<form action='servlet2'>");
  out.print("<input type='submit' value='go'>");
  out.print("</form>");

  out.close();
 }catch(Exception e){System.out.println(e);}
 }
}
```

```
SecondServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class SecondServlet extends HttpServlet {
 public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{
   response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  Cookie ck[]=request.getCookies();
  out.print("Hello "+ck[0].getValue());

  out.close();

     }catch(Exception e){System.out.println(e);}
  } }
```

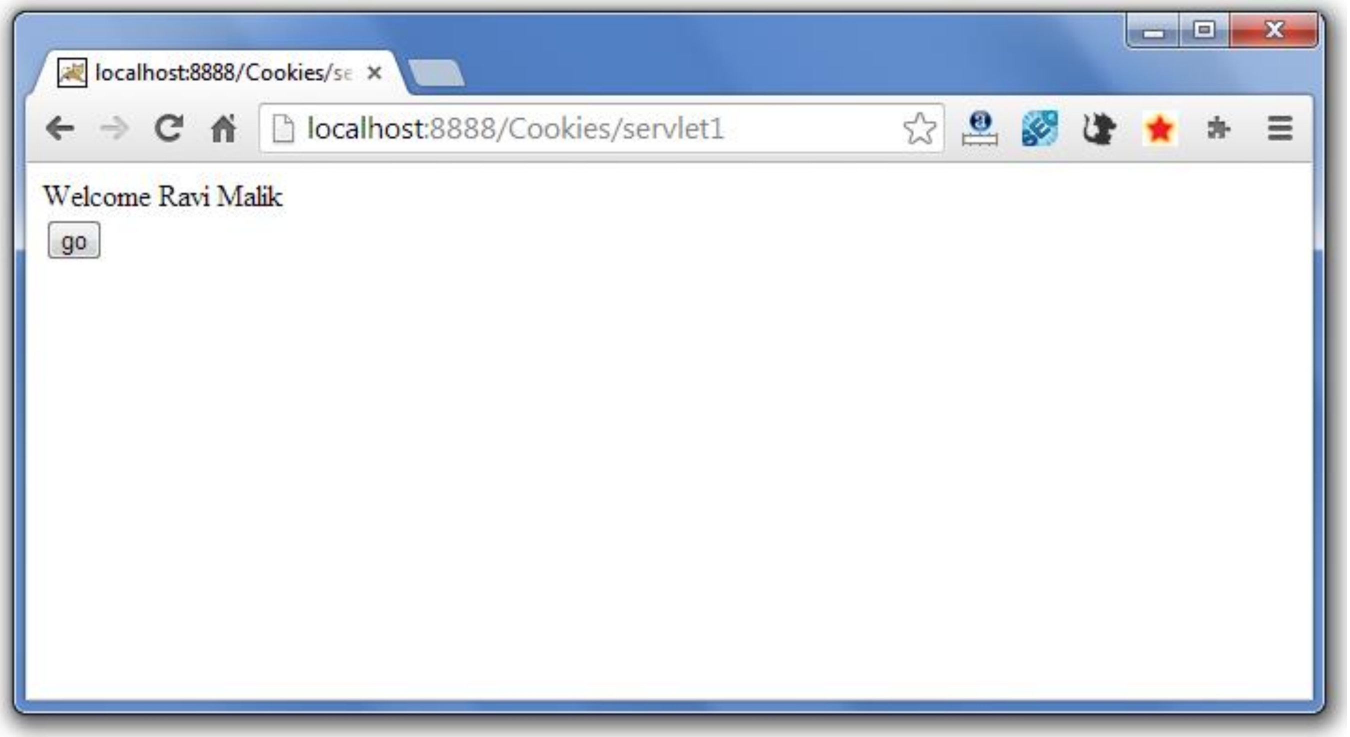
web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

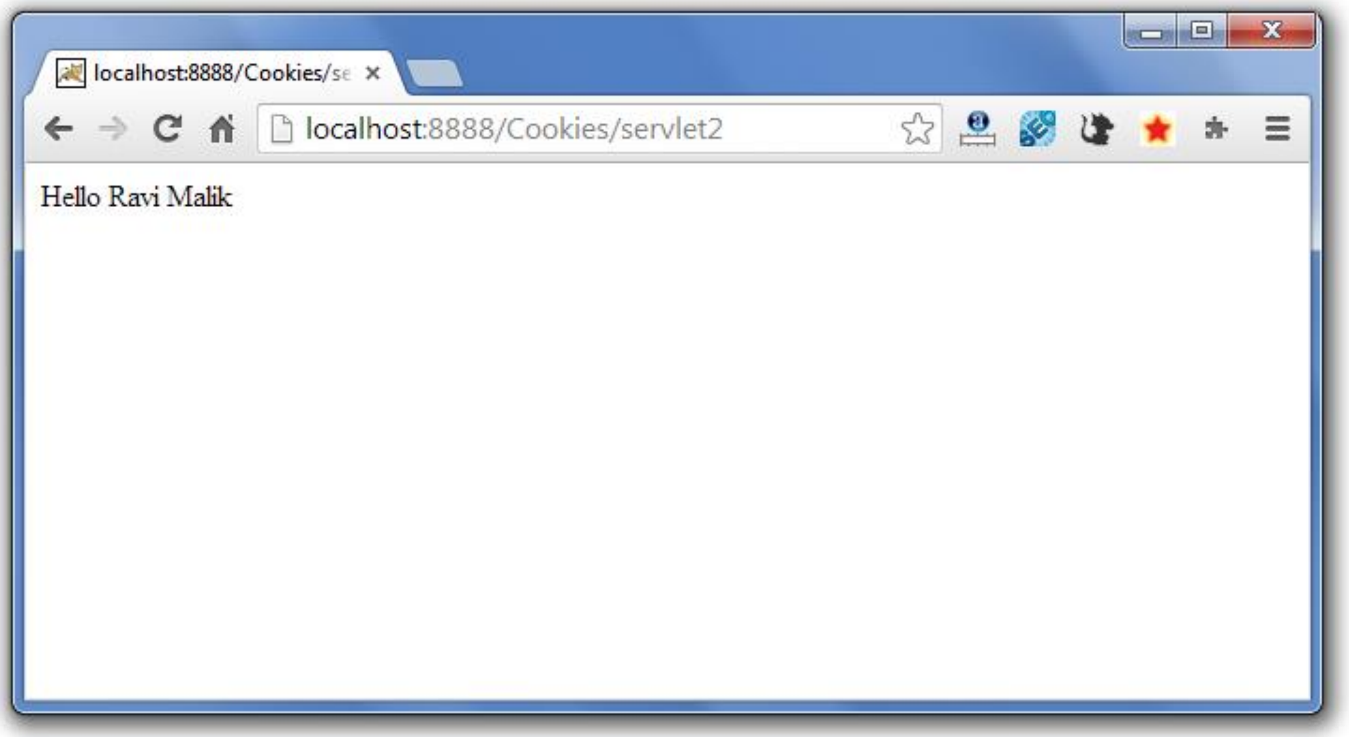
Output



Name: Ravi Malik



```
localhost:8888/Cookies/servlet1

Welcome Ravi Malik
```



```
localhost:8888/Cookies/servlet2

Hello Ravi Malik
```

Session tracking

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- 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. So we need to maintain the state of the user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new request.

Session Tracking Techniques

- There are four techniques used in Session tracking:

    1. Cookies
    2. Hidden Form Field
    3. URL Rewriting
    4. HttpSession

Unit V – Java Applets

- JApplet
- Button
- Combo
- Trees
- Tables
- Panes
- AWT Classes
- Working with Graphics
- Working with Color
- Working with Font

Java Swing (JApplet)

- Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

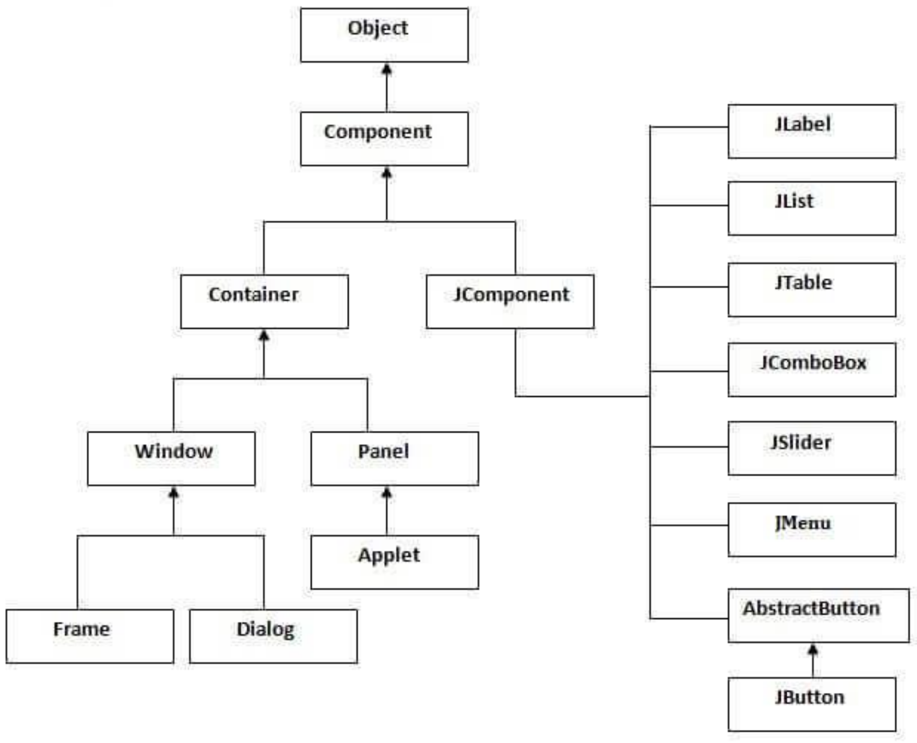
Difference between AWT and Swing

- There are many differences between java awt and swing that are given below:

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Hierarchy of Java Swing classes

- The hierarchy of java swing API is given below:



Methods of Component class

- The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

- There are two ways to create a frame:
    - By creating the object of Frame class (association)
    - By extending Frame class (inheritance)

JButton

- The JButton class is used to create a labeled button that has platform independent implementation.
- The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

- The Declaration of javax.swing.JButton class.
    - ✓ **public class** JButton **extends** AbstractButton **implements** Accessible

Constructors of JButton

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

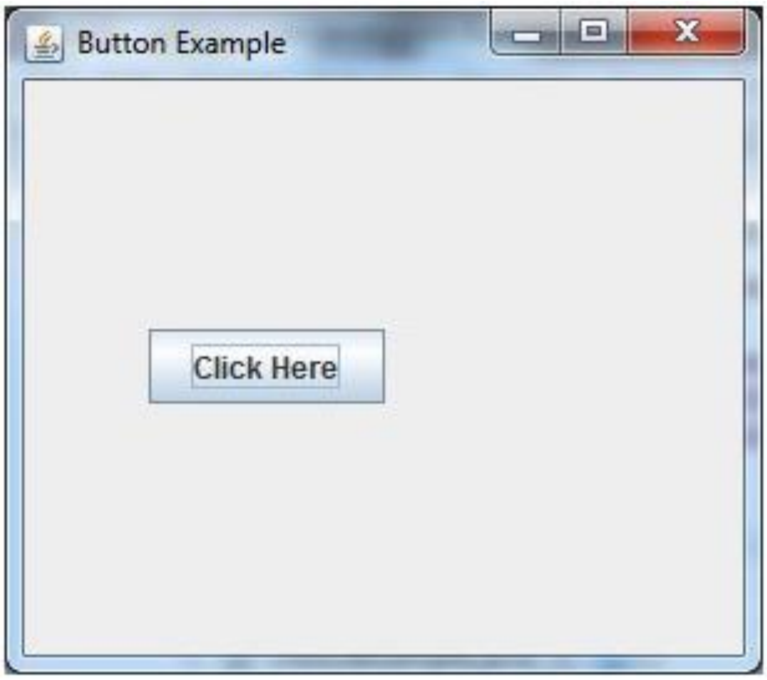
Methods of JButton class

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

JButton ExampleProgram

```
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
   JFrame f=new JFrame("Button Example");
   JButton b=new JButton("Click Here");
   b.setBounds(50,100,95,30);
   f.add(b);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

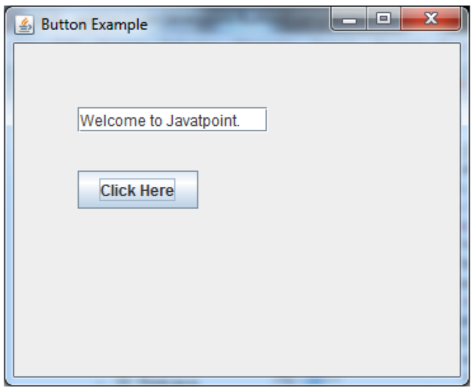
Output:



JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample
{
public static void main(String[] args) {
   JFrame f=new JFrame("Button Example");
   final JTextField tf=new JTextField();
   tf.setBounds(50,50, 150,20);
   JButton b=new JButton("Click Here");
   b.setBounds(50,100,95,30);
   b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
         tf.setText("Welcome to Javatpoint.");
      }
   });
   f.add(b);f.add(tf);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

Output:



JLabel

- The object of JLabel class is a component for placing text in a container.
- It is used to display a single line of read only text.
- The text can be changed by an application but a user cannot edit it directly.
- It inherits JComponent class.

JLabel class declaration

The declaration of javax.swing.JLabel class.

✓ **public class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

Constructors of JLabel:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

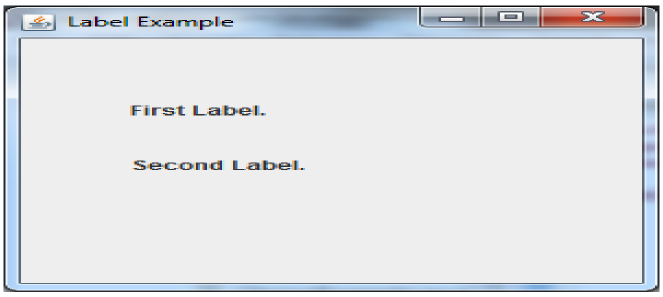
Methods f JLabel class:

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

JLabel Example

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
  {
  JFrame f= new JFrame("Label Example");
  JLabel l1,l2;
  l1=new JLabel("First Label.");
  l1.setBounds(50,50, 100,30);
  l2=new JLabel("Second Label.");
  l2.setBounds(50,100, 100,30);
  f.add(l1); f.add(l2);
  f.setSize(300,300);
  f.setLayout(null);
  f.setVisible(true);
  } }}
```

Output:



JTextField

- The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

- The declaration for javax.swing.JTextField class.
    - ✓ **public class** JTextField **extends** JTextComponent **implements** SwingConstants

Constructors of JTextField:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

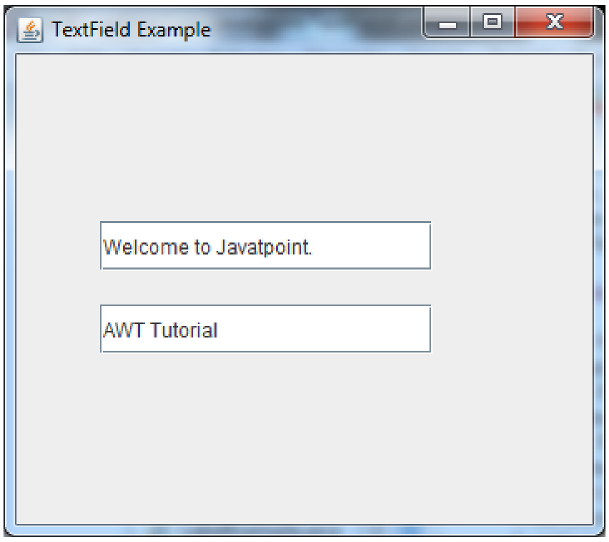
Methods of JTextField:

Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
   {
   JFrame f= new JFrame("TextField Example");
   JTextField t1,t2;
   t1=new JTextField("Welcome to Javatpoint.");
   t1.setBounds(50,100, 200,30);
   t2=new JTextField("AWT Tutorial");
   t2.setBounds(50,150, 200,30);
   f.add(t1); f.add(t2);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);  } }
```

Output:



```
Welcome to Javatpoint.
AWT Tutorial
```

JCheckBox

- The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false).
- Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

JCheckBox class declaration

- The declaration for javax.swing.JCheckBox class.
  - ✓ **public class** JCheckBox **extends** JToggleButton **implements** Accessible

Constructors of JCheckBox:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

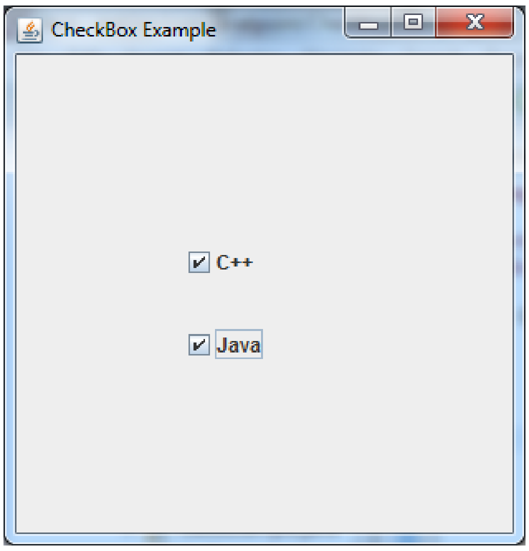
Methods of JCheckBox:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

Java JCheckBox Example

```
import javax.swing.*;
public class CheckBoxExample
{
    CheckBoxExample(){
      JFrame f= new JFrame("CheckBox Example");
      JCheckBox checkBox1 = new JCheckBox("C++");
      checkBox1.setBounds(100,100, 50,50);
      JCheckBox checkBox2 = new JCheckBox("Java", true);
      checkBox2.setBounds(100,150, 50,50);
      f.add(checkBox1);
      f.add(checkBox2);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
   {
   new CheckBoxExample();
   }}
```

Output:



JRadioButton

- The JRadioButton class is used to create a radio button.
- It is used to choose one option from multiple options. It is widely used in exam systems or quiz.
- It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

- The declaration for javax.swing.JRadioButton class.
    - ✓ **public class** JRadioButton **extends** JToggleButton **implements** Accessible

Constructors of JRadioButton:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

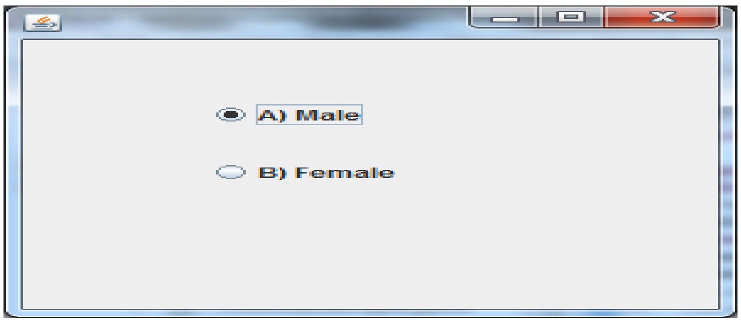
Methods of JRadioButton:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JRadioButton Example

```
import javax.swing.*;
public class RadioButtonExample {
JFrame f;
RadioButtonExample(){
f=new JFrame();
JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) Female");
r1.setBounds(75,50,100,30);
r2.setBounds(75,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
   new RadioButtonExample();
}   }
```

Output:



JComboBox

- The object of Choice class is used to show popup menu of choices.
- Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

- The declaration for javax.swing.JComboBox class.

  ✓ **public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListData Listener, ActionListener, Accessible

Constructors of JComboBox:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

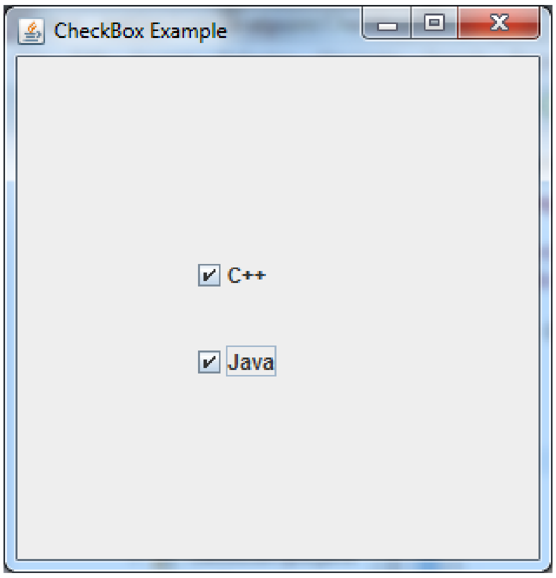
Methods of JComboBox:

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Java JComboBox Example

```
import javax.swing.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
   f=new JFrame("ComboBox Example");
   String country[]={"India","Aus","U.S.A","England","Newzealand"};
   JComboBox cb=new JComboBox(country);
   cb.setBounds(50, 50,90,20);
   f.add(cb);
   f.setLayout(null);
   f.setSize(400,500);
   f.setVisible(true);
}
public static void main(String[] args) {
   new ComboBoxExample();
}
}
```

Output:



JTree

- The JTree class is used to display the tree structured data or hierarchical data.
- JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

JTree class declaration

- The declaration for javax.swing.JTree class.

  ✓ **public class** JTree **extends** JComponent **implements** Scrollable, Accessible

Constructors of JTree:

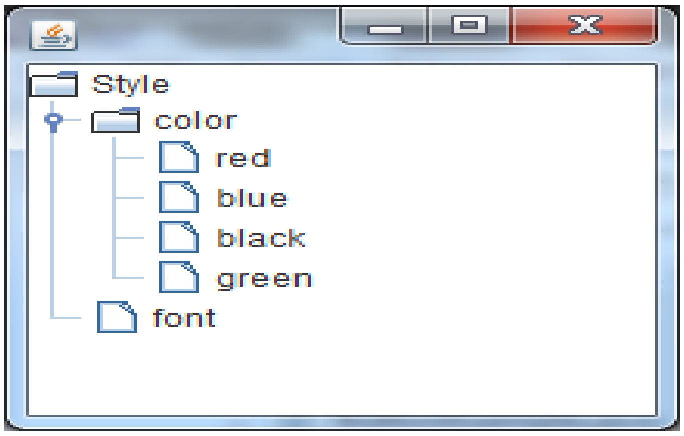
Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

Java JTree Example

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample {
JFrame f;
TreeExample(){
   f=new JFrame();
   DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
   DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
   DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
   style.add(color);
```

```
    style.add(font);
    DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
    DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
    DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");
    DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");
    color.add(red); color.add(blue); color.add(black); color.add(green);
    JTree jt=new JTree(style);
    f.add(jt);
    f.setSize(200,200);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TreeExample();
}}
```

Output:



JTable

- The JTable class is used to display data in tabular form.
- It is composed of rows and columns.

JTable class declaration

- The declaration for javax.swing.JTable class.
  - ✓ **public class** JTree **extends** JComponent **implements** Scrollable, Accessible

Constructors of JTable class:

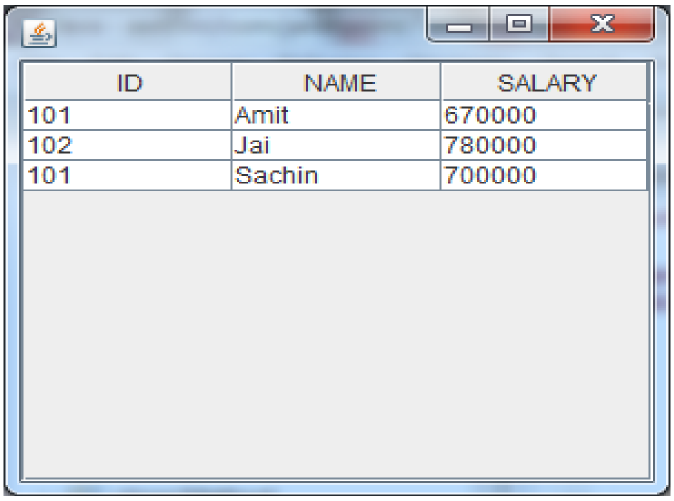
Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
```

```
    f=new JFrame();
    String data[][]={ {"101","Amit","670000"},
                      {"102","Jai","780000"},
                      {"101","Sachin","700000"}};
    String column[]={"ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TableExample();
}
}
```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

JScrollPane

- A JscrollPane is used to make scrollable view of a component.
- When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Constructors of JScrollPane:

Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

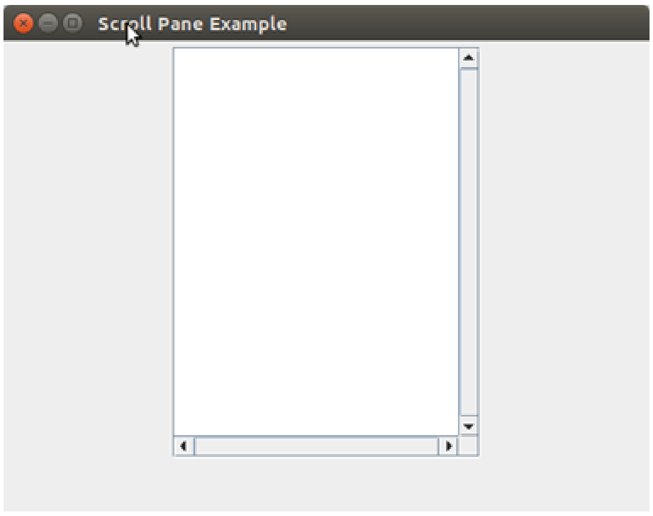
Methods of JScrollPane:

Modifier	Method	Description
Void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
Void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
Void	setCorner(String, Component)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
Component	getCorner(String)	
Void	setViewportView(Component)	Set the scroll pane's client.

JScrollPane Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JtextArea;
public class JScrollPaneExample {
  private static final long serialVersionUID = 1L;
  private static void createAndShowGUI() {
    // Create and set up the window.
    final JFrame frame = new JFrame("Scroll Pane Example");
    // Display the window.
    frame.setSize(500, 500);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // set flow layout for the frame
    frame.getContentPane().setLayout(new FlowLayout());
    JTextArea textArea = new JTextArea(20, 20);
    JScrollPane scrollableTextArea = new JScrollPane(textArea);
    scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWA
YS);
    scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    frame.getContentPane().add(scrollableTextArea);
  }
  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createAndShowGUI();
      }
    });
  }
}
```

Output:



JTabbedPane

- The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

- The declaration for javax.swing.JTabbedPane class.
  - ✓ **public class** JTabbedPane **extends** JComponent **implements** Serializable, Accessible, Sw ingConstants

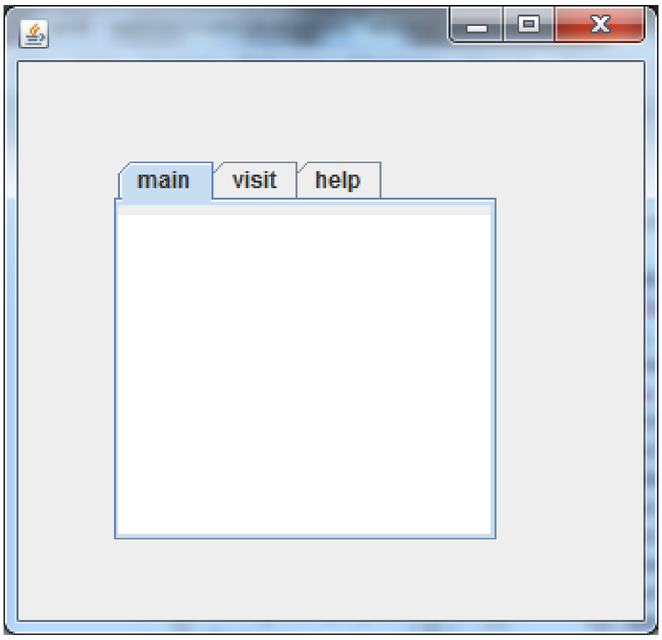
Constructors of JTabbedPane:

Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
   f=new JFrame();
   JTextArea ta=new JTextArea(200,200);
   JPanel p1=new JPanel();
   p1.add(ta);
   JPanel p2=new JPanel();
   JPanel p3=new JPanel();
   JTabbedPane tp=new JTabbedPane();
   tp.setBounds(50,50,200,200);
   tp.add("main",p1);
   tp.add("visit",p2);
   tp.add("help",p3);
   f.add(tp);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
public static void main(String[] args) {
   new TabbedPaneExample();
}}
```

Output:

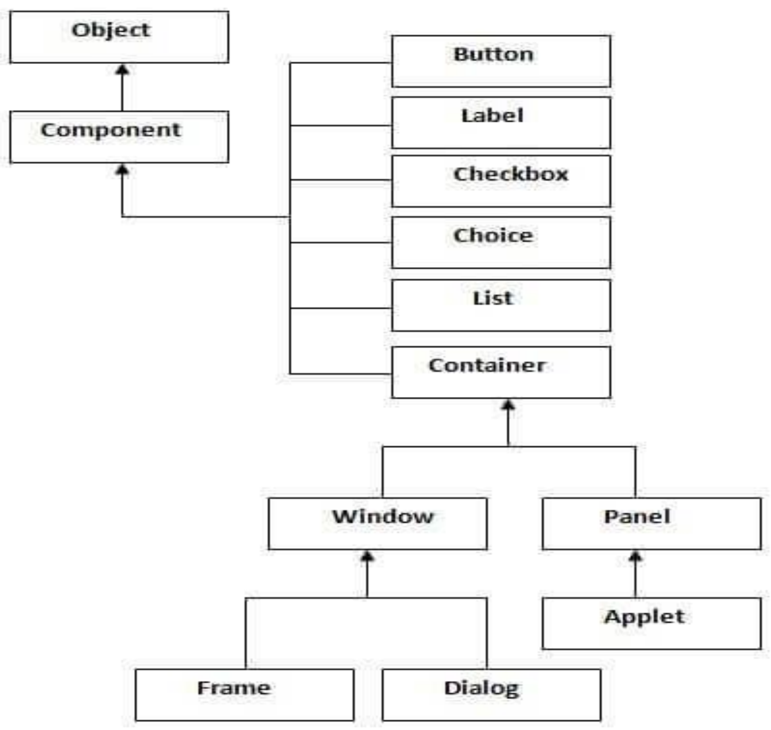


AWT Classes

- Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.
- The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

- The window is the container that have no borders and menu bars.
- You must use frame, dialog or another window for creating a window.

Panel

- The Panel is the container that doesn't contain title bar and menu bars.
- It can have other components like button, textfield etc.

Frame

- The Frame is the container that contain title bar and can have menu bars.
- It can have other components like button, textfield etc.

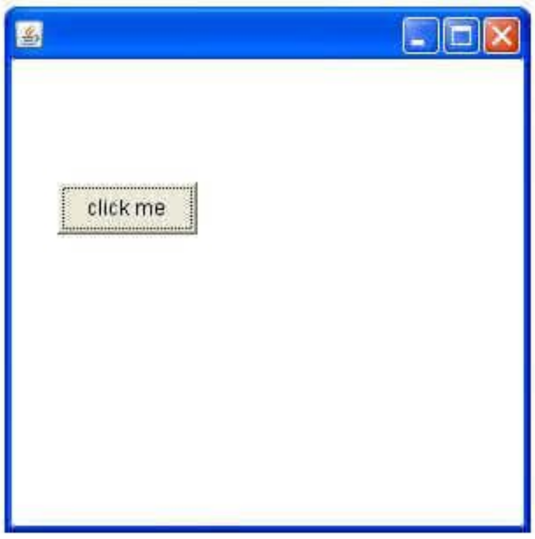
Methods of Component class

Method	**Description**
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

AWT Example

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

Output:



Event and Listener (Event Handling)

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc.
- The java.awt.event package provides many event classes and Listener interfaces for event handling.

Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener

AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Working with Graphics, Color and Font in Applet

- java.awt.Graphics class provides many methods for graphics programming.

Methods of Graphics class

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.