

21CSC203P -Advanced Programming Practice Unit-3 Advanced Java Programming Paradigms

APP Faculties

Department of Computational Intelligence

SRM Institute of Science and Technology



Outline of the Presentation

- S-19: Concurrent Programming Paradigm
- S-20: Multithreading and Multitasking
- S-21: Thread classes and methods
- S-22: Declarative Programming Paradigm: Java Database Connectivity (JDBC)
- S-22: Connectivity with MySQL – Query Execution
- S-23: Graphical User Interface Based Programming Paradigm
- S-24: Java Applets-Basics
- S-24: Java Swing
- S-25: Model View Controller (MVC) and Widgets



Concurrent Programming Paradigm

- Computing systems model the world, and the world contains actors that execute independently of, but communicate with, each other. In modelling the world, many (possibly) parallel executions have to be composed and coordinated, and that's where the study of concurrency comes in.
- There are two common models for concurrent programming: shared memory and message passing.
 - **Shared memory.** In the shared memory model of concurrency, concurrent modules interact by reading and writing shared objects in memory.
 - **Message passing.** In the message-passing model, concurrent modules interact by sending messages to each other through a communication channel. Modules send off messages, and incoming messages to each module are queued up for handling

What is concurrent?

Concurrent describes things that are occurring, or people who are doing something, **at the same time**, such as "concurrent users" of a computer program.

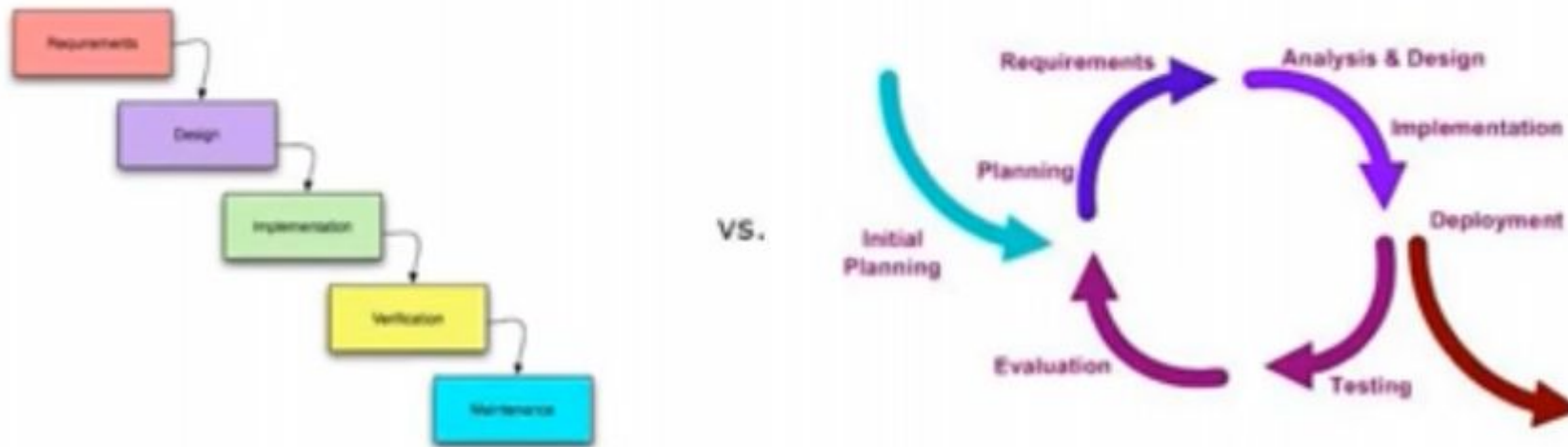
On the other hand, Consecutive refers to things that are arranged or happen in a sequential order.

Real life example?

Sequential Engineering vs Concurrent Design and Manufacturing

Real life example?

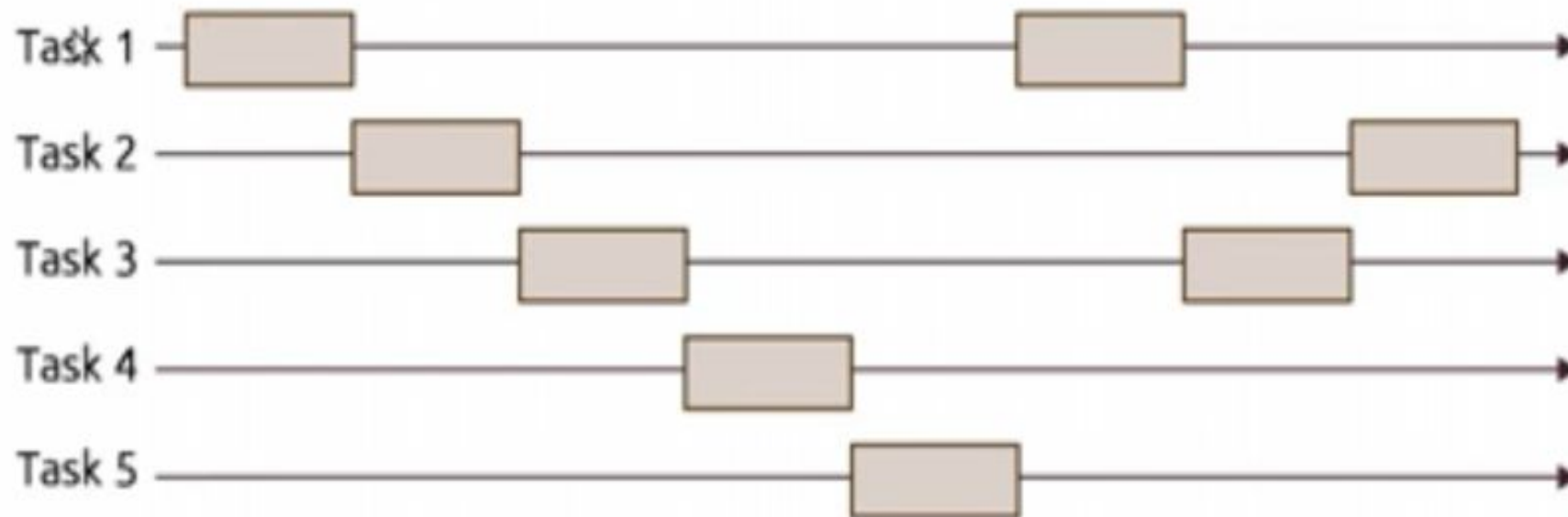
Sequential Engineering vs Concurrent Design and Manufacturing



What is concurrent programming/computing?

In a concurrent program, several streams of operations may execute concurrently. Each stream of operations executes as it would in a sequential program except for the fact that streams can communicate and interfere with one another.

Concurrency is a property of a system which enables overlapping of process lifetimes.



Concurrency vs Parallelism

Dealing with multiple things at once versus doing multiple things at once.

Concurrency

Tasks start, run, and complete in overlapping time periods



asyncio

Parallelism

Tasks run simultaneously



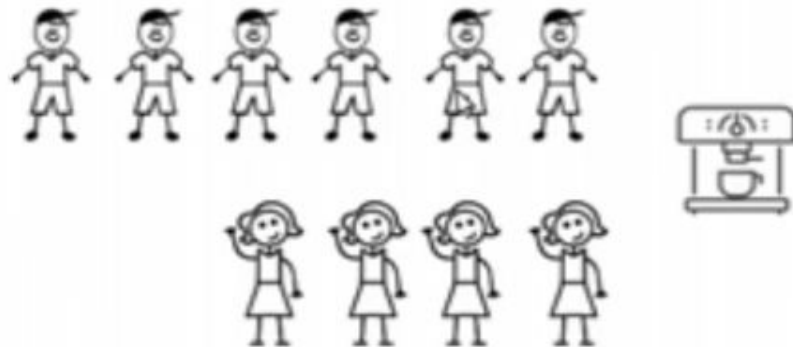
threads / processes
+ multicore

A simple story...

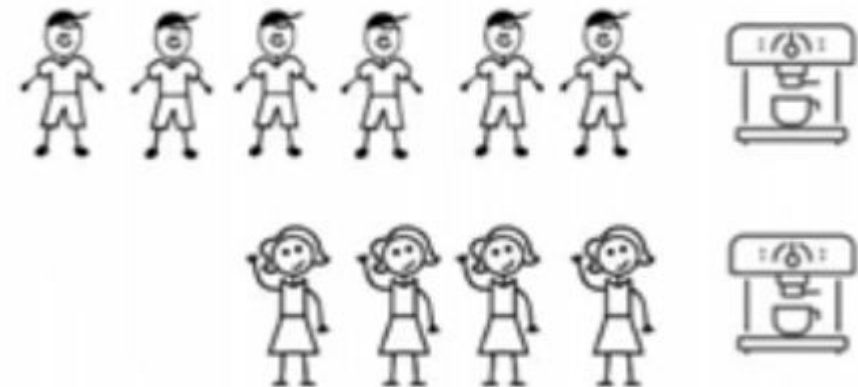
Sequential



Concurrent



Parallelism





Issues Concurrent Programming Paradigm

Concurrent programming is programming with multiple tasks. The major issues of concurrent programming are:

- Sharing computational resources between the tasks;
- Interaction of the tasks.

Objects shared by multiple tasks have to be safe for concurrent access. Such objects are called protected. Tasks accessing such an object interact with each other indirectly through the object.

An access to the protected object can be:

- Lock-free, when the task accessing the object is not blocked for a considerable time;
- Blocking, otherwise.

Blocking objects can be used for task synchronization. To the examples of such objects belong:

- Events;
- Mutexes and semaphores;
- Waitable timers;
- Queues



Multithreaded Programming

- Java provides built-in support for *multithreaded programming*.
- A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a *thread*, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.
- There are two distinct types of multitasking: **process-based** and **thread-based**.
- It is important to understand the difference between the two.



Process-based multitasking

- **Process-based multitasking** is the more familiar form.
- A *process* is, in essence, a program that is executing. Thus, *process-based* multitasking is the feature that allows your computer to run two or more programs concurrently.
- **For example, process-based multitasking** enables you to run the Java compiler at the same time that you are using a text editor.
- In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.



Thread-based multitasking

- *Thread-based multitasking* environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously.
- For instance, a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads.
- Thus, process-based multitasking deals with the “big picture,” and thread-based multitasking handles the details.



Thread Class and the Runnable Interface

- Java's multithreading system is built upon the **Thread** class, its methods, and its companion interface, **Runnable**.
Thread encapsulates a thread of execution.
- The **Thread** class defines several methods that help manage threads.

Method	Meaning
getName	Obtain a thread's name.
getPriority	Obtain a thread's priority.
isAlive	Determine if a thread is still running.
join	Wait for a thread to terminate.
run	Entry point for the thread.
sleep	Suspend a thread for a period of time.
start	Start a thread by calling its run method.



Main Thread

- When a Java program starts up, one thread begins running immediately. This is
- usually called the main thread of your program, because it is the one that is executed
- when your program begins. The main thread is important for two reasons:
 - It is the thread from which other “child” threads will be spawned.
 - Often it must be the last thread to finish execution because it performs various shutdown actions.



Main Thread (Continue...)

- Main thread is created automatically when your program is started, it can be controlled through a **Thread** object.
- To do so, you must obtain a reference to it by calling the method **currentThread()**, which is a **public static** member of **Thread**. Its general form is:

```
static Thread currentThread( )
```

- This method returns a reference to the thread in which it is called. Once you have a reference to the main thread, you can control it just like any other thread.

Example:

```
// Controlling the main Thread.
class CurrentThreadDemo {
    public static void main(String args[]) {
        Thread t = Thread.currentThread();

        System.out.println("Current thread: " + t);

        // change the name of the thread
        t.setName("My Thread");
        System.out.println("After name change: " + t);

        try {
            for(int n = 5; n > 0; n--) {
                System.out.println(n);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }
    }
}
```

```
Current thread: Thread[main,5,main]
After name change: Thread[My Thread,5,main]
```

```
5
4
3
2
```

- In this program, a reference to the current thread (the main thread, in this case) is obtained by calling **currentThread()**, and this reference is stored in the local variable **t**.
- Next, the program displays information about the thread. The program then calls **setName()** to change the internal name of the thread. Information about the thread is then redisplayed.
- Next, a loop counts down from five, pausing one second between each line. The pause is accomplished by the **sleep()** method.
- The argument to **sleep()** specifies the delay period in milliseconds. Notice the **try/catch** block around this loop.
- The **sleep()** method in **Thread** might throw an **InterruptedException**.
- This would happen if some other thread wanted to interrupt this sleeping one.
- This example just prints a message if it gets interrupted. In a real program, you would need to handle this differently.

Output





Creating a Thread

- In the most general sense, you create a thread by instantiating an object of type **Thread**.
- Java defines two ways in which this can be accomplished:
 - You can implement the **Runnable** interface.
 - You can extend the **Thread** class, itself.

Example that creates a new thread and starts it running



```
// Create a second thread.
class NewThread implements Runnable {
    Thread t;

    NewThread() {
        // Create a new, second thread
        t = new Thread(this, "Demo Thread");
        System.out.println("Child thread: " + t);
        t.start(); // Start the thread
    }

    // This is the entry point for the second thread.
    public void run() {
        try {
            for(int i = 5; i > 0; i--) {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }
}
```

```
class ThreadDemo {
    public static void main(String args[]) {
        new NewThread(); // create a new thread

        try {
            for(int i = 5; i > 0; i--) {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }
        System.out.println("Main thread exiting.");
    }
}
```

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
```

Output



Extending Thread

- The second way to create a thread is to create a new class that extends **Thread**, and then to create an instance of that class.
- The extending class must override the **run()** method, which is the entry point for the new thread. It must also call **start()** to begin execution of the new thread.
- The preceding program rewritten to extend **Thread**.

Extending Thread (Continue...)



```
// Create a second thread by extending Thread
class NewThread extends Thread {

    NewThread() {
        // Create a new, second thread
        super("Demo Thread");
        System.out.println("Child thread: " + this);
        start(); // Start the thread
    }

    // This is the entry point for the second thread.
    public void run() {
        try {
            for(int i = 5; i > 0; i--) {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }
}
```

Creating Multiple Threads

```
class ExtendThread {
    public static void main(String args[]) {
        new NewThread(); // create a new thread

        try {
            for(int i = 5; i > 0; i--) {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }
        System.out.println("Main thread exiting.");
    }
}
```

- This program generates the same output as the preceding version.
- Child thread is created by instantiating an object of **NewThread**, which is derived from **Thread**.

Creating Multiple Threads



```
// Create multiple threads.
class NewThread implements Runnable {
    String name; // name of thread
    Thread t;

    NewThread(String threadname) {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        t.start(); // Start the thread
    }

    // This is the entry point for thread.
    public void run() {
```

The output from this program is shown here:

```
New thread: Thread[One,5,main]
New thread: Thread[Two,5,main]
New thread: Thread[Three,5,main]
One: 5
Two: 5
Three: 5
One: 4
Two: 4
Three: 4
One: 3
Three: 3
Two: 3
```

```
One: 2
Three: 2
Two: 2
One: 1
Three: 1
Two: 1
One exiting.
Two exiting.
Three exiting.
Main thread exiting.
```

```
try {
    for(int i = 5; i > 0; i--) {
        System.out.println(name + ": " + i);
        Thread.sleep(1000);
    }
} catch (InterruptedException e) {
    System.out.println(name + "Interrupted");
}
System.out.println(name + " exiting.");
}

}

class MultiThreadDemo {
    public static void main(String args[]) {
        new NewThread("One"); // start threads
        new NewThread("Two");
        new NewThread("Three");

        try {
            // wait for other threads to end
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            System.out.println("Main thread Interrupted");
        }

        System.out.println("Main thread exiting.");
    }
}
```




Using is Alive() and join()

- The main thread to finish last. In the preceding examples, this is accomplished by calling **sleep()** within **main()**, with a long enough delay to ensure that all child threads terminate prior to the main thread.
- Two ways exist to determine whether a thread has finished. First, you can call **isAlive()** on the thread. This method is defined by **Thread**, and its general form is shown here:

```
final boolean isAlive( )
```

- The **isAlive()** method returns true if the thread upon which it is called is still running. It returns false otherwise.
- While **isAlive()** is occasionally useful, the method that you will more commonly use to wait for a thread to finish is called **join()**, shown here:

```
final void join( ) throws InterruptedException
```

Example

```
// Using join() to wait for threads to finish.
class NewThread implements Runnable {
    String name; // name of thread
    Thread t;

    NewThread(String threadname) {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        t.start(); // Start the thread
    }

    // This is the entry point for thread.
    public void run() {
        try {
            for(int i = 5; i > 0; i--) {
                System.out.println(name + ": " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println(name + " interrupted.");
        }
        System.out.println(name + " exiting.");
    }
}
```

```
class DemoJoin {
    public static void main(String args[]) {
        NewThread obl = new NewThread("One");
        NewThread ob2 = new NewThread("Two");
        NewThread ob3 = new NewThread("Three");

        System.out.println("Thread One is alive: "
                           + obl.t.isAlive());
        System.out.println("Thread Two is alive: "
                           + ob2.t.isAlive());
        System.out.println("Thread Three is alive: "
                           + ob3.t.isAlive());

        // wait for threads to finish
        try {
            System.out.println("Waiting for threads to finish.");
            obl.t.join();
        }
```

Example



Sample output from this program is shown here:

```
ob2.t.join();
ob3.t.join();
} catch (InterruptedException e) {
    System.out.println("Main thread Interrupted");
}

System.out.println("Thread One is alive: "
    + ob1.t.isAlive());
System.out.println("Thread Two is alive: "
    + ob2.t.isAlive());
System.out.println("Thread Three is alive: "
    + ob3.t.isAlive());

System.out.println("Main thread exiting.");
}
}
```

Output



```
New thread: Thread[One,5,main]
New thread: Thread[Two,5,main]
New thread: Thread[Three,5,main]
Thread One is alive: true
Thread Two is alive: true
Thread Three is alive: true
Waiting for threads to finish.
One: 5
Two: 5
Three: 5
One: 4
Two: 4
Three: 4
One: 3
Two: 3
Three: 3
One: 2
Two: 2
Three: 2
One: 1
Two: 1
Three: 1

Two exiting.
Three exiting.
One exiting.
Thread One is alive: false
Thread Two is alive: false
Thread Three is alive: false
Main thread exiting.
```



Thread Priorities

- Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run.
- To set a thread's priority, use the **setPriority()** method, which is a member of **Thread**. This is its general form:

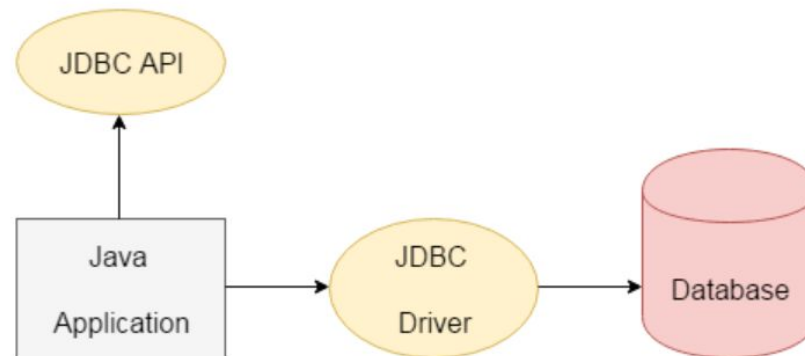
```
final void setPriority(int level)
```

- Here, *level* specifies the new priority setting for the calling thread. The value of *level* must be within the range **MIN_PRIORITY** and **MAX_PRIORITY**.
- Currently, these values are 1 and 10, respectively. To return a thread to default priority, specify **NORM_PRIORITY**, which is currently 5. These priorities are defined as **final** variables within **Thread**.
- You can obtain the current priority setting by calling the **getPriority()** method of **Thread**, shown here:

```
final int getPriority( )
```

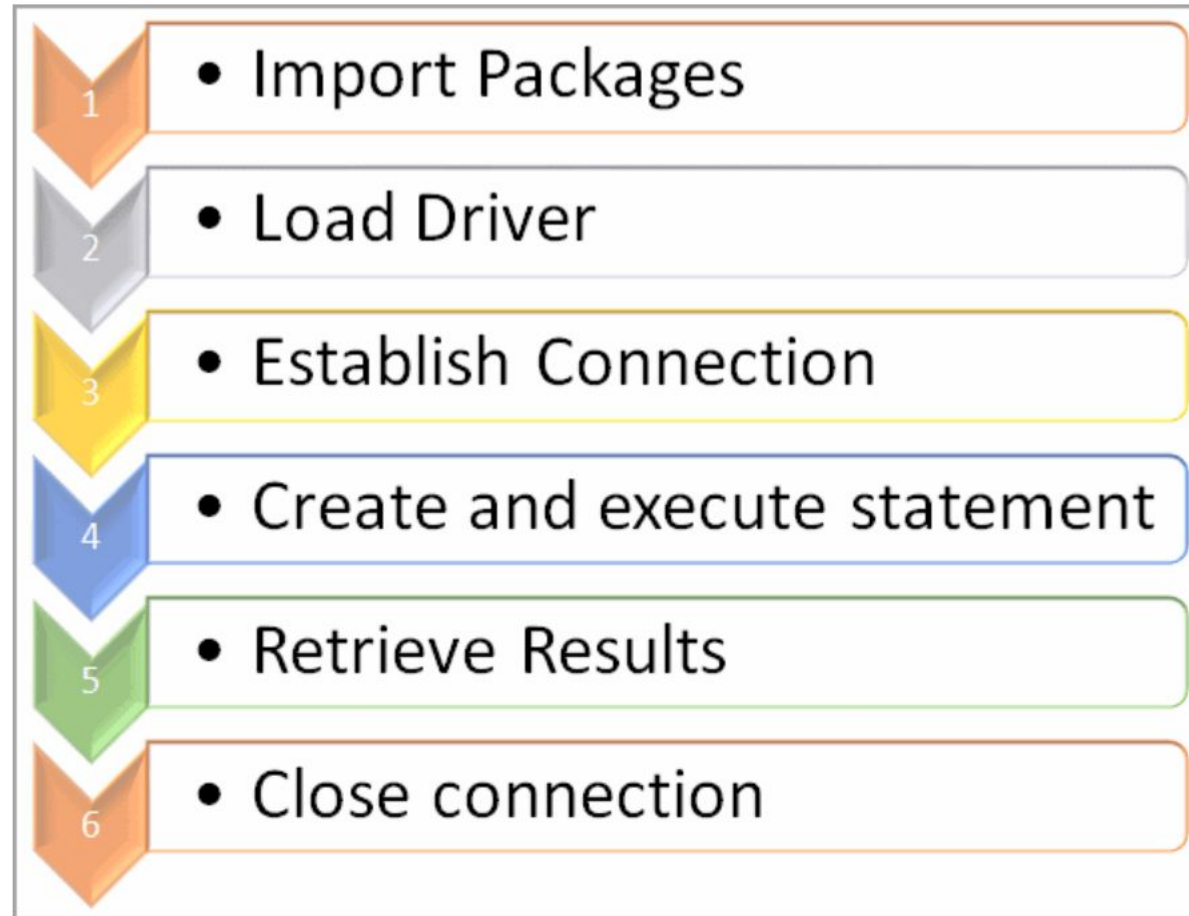
Java Database Connectivity (JDBC)

- JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.
- JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database.



JDBC Connection Steps

There are 6 basic steps to connect with JDBC.





Import Packages

- First, to import the existing packages to use it in our Java program. Import will make sure that JDBC API classes are available for the program. We can then use the classes and subclasses of the packages.

import java.sql.*;

JDBC API 4.0 mainly provides 2 important packages:

- java.sql
- javax.sql

java.sql package

This package provides classes and interfaces to perform most of the JDBC functions like creating and executing SQL queries.



Classes/ Interfaces	Description
BLOB	It represents SQL Blob value in Java program
CallableStatement	It is used to execute SQL stored procedures
CLOB	It represents SQL Clob value in Java program
Connection	It creates a connection (session) with a specific Database
Date	It provides support for Date SQL type
Driver	It creates an instance of a Driver with Driver Manager
DriverManager	It provides basic service to manage a set of JDBC Drivers
ParameterMetaData	It is an object which can be used to get the information about the types and properties of each parameter in a PreparedStatement Object
PreparedStatement	It is used to create and execute a parameterized query in the Java program
ResultSet	It is used to access the result row-by-row
ResultSetMetaData	It is used to get the information about the types and properties of the columns in a ResultSet object
RowId	It represents the SQL ROWID value
Savepoint	It represents savepoint in transaction
SQLData	It is used to map the SQL User Defined Type (UDT) to a class in Java program
SQLXML	It represents SQL XML type
Statement	It is used to execute a static SQL statement
DriverPropertyInfo	It provides Driver properties to make a connection
SQLException	It provides information on database errors
SQLTimeoutException	It is a subclass of SQLException thrown when the timeout specified by the statement has expired
SQLWarning	It is an exception that provides information on database access warnings



javax.sql package

- It is a JDBC extension API and provides server-side data access and processing in Java Program.

Classes/ Interfaces	Description
CommonDataSource	It is an interface that defines the methods which are common between DataSource, XADataSource and ConnectionPoolDataSource
ConnectionPoolDataSource	It is a factory for PooledConnection objects
DataSource	It is a factory for connections to the physical DataSource that the object represents
PooledConnection	It is used to manage Connection Pool
RowSet	It provides support to the JDBC API for Java beans Component Model
RowSetMetadata	It has the information about the columns in a RowSet object
ConnectionEvent	It provides information about the occurrence of connection-related events
ConnectionEventListener	It is used to register PooledConnection object events
RowSetEvent	It generates when an event occurs to a Rowset object
StatementEvent	It is sent to all StatementEventListeners which were registered with a



Load Driver

First, we should load/register the driver in the program before connecting to the Database. You need to register it only once per database in the program.

We can load the driver in the following 2 ways:

1. Class.forName()

2. DriverManager.registerDriver()



Class.forName()

In this way, the driver's class file loads into the memory at runtime. It implicitly loads the driver. While loading, the driver will register with JDBC automatically.

DB Name	JDBC Driver Name
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.driver.OracleDriver
Microsoft SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver
MS Access	net.ucanaccess.jdbc.UcanaccessDriver
PostgreSQL	org.postgresql.Driver
IBM DB2	com.ibm.db2.jdbc.net.DB2Driver
Sybase	com.sybase.jdbcSybDriver
TeraData	com.teradata.jdbc.TeraDriver



DriverManager.registerDriver()

DriverManager is an inbuilt class that is available in the java.sql package.

It acts as a mediator between Java application and database which you want to connect. Before you connect with the database, you need to register the driver with DriverManager.

The main function of DriverManager is to load the driver class of the Database and create a connection with DB.

- **Public static void registerDriver(driver)** – This method will register the driver with the Driver Manager. If the driver is already registered, then it won't take any action.
- It will throw **SQLException** if the database error occurs.
- It will throw **NullPointerException** if the driver is null.

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())
```

```
DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver())
```




Establish Connection

After loading the driver, the next step is to create and establish the connection. Once required, packages are imported and drivers are loaded and registered, then we can go for establishing a Database connection.

DriverManager class has the getConnection method, we will use this method to get the connection with Database.

To call getConnection() method, we need to pass 3 parameters. The 3 parameters are string data type URL, a username, and a password to access the database.

- **The getConnection() method is an overloaded method. The 2 methods are:**
- **getConnection(URL,username,password);** – It has 3 parameters URL, username, password.
- **getConnection(URL);** – It has only one parameter. URL has a username and password also.

Establish Connection

The following table lists the JDBC connection strings for the different databases:

Database	Connection String/DB URL
MySQL	<code>jdbc:mysql://HOST_NAME:PORT/DATABASE_NAME</code>
Oracle	<code>jdbc:oracle:thin:@HOST_NAME:PORT:SERVICE_NAME</code>
Microsoft SQL Server	<code>jdbc:sqlserver://HOST_NAME:PORT;DatabaseName=< DATABASE_NAME></code>
MS Access	<code>jdbc:ucanaccess://DATABASE_PATH</code>
PostgreSQL	<code>jdbc:postgresql://HOST_NAME:PORT/DATABASE_NAME</code>
IBM DB2	<code>jdbc:db2://HOSTNAME:PORT/DATABASE_NAME</code>
Sybase	<code>jdbc:Sybase:Tds:HOSTNAME:PORT/DATABASE_NAME</code>
TeraData	<code>jdbc:teradata://HOSTNAME/database=< DATABASE_NAME>,tmode=ANSI,charset=UTF8</code>

Example:

**Connection con =
DriverManager.getConnection(jdbc:oracle:thin:@localhost
:1521:xe,System,Pass123@)**

Here in this example,

- **thin** refers to the Driver type.
- **localhost** is where the Oracle database is running.
- **1521** is the port number to connect to DB.
- **xe** – SID
- **System** – User name to connect to the Oracle Database.
- **Pass123@** – Password



Create And Execute Statement

Once the connection has established, we can interact with the connected Database. First, we need to create the statement to perform the SQL query and then execute the statement.

(i) Create Statement

Now we will create the statement object that runs the query with the connected database. We use the `createStatement` method of the *Connection* class to create the query.

There are 3 statement interfaces are available in the `java.sql` package. These are explained below:

a) Statement

This interface is used to implement simple SQL statements with no parameter. It returns the `ResultSet` object.

```
Statement statemnt1 = conn.createStatement();
```



Create And Execute Statement

b) PreparedStatement

This PreparedStatement interface extends the Statement interface. So, it has more features than the Statement interface. It is used to implement parameterized and precompiled SQL statements. The performance of the application increases because it compiles the query only once.

- It is easy to reuse this interface with a new parameter. It supports the IN parameter. Even we can use this statement without any parameter.

```
String select_query = "Select * from states where state_id = 1";
```

```
PreparedStatement prpstmt = conn.prepareStatement(select_query);
```



Create And Execute Statement

c) CallableStatement

CallableStatement interface extends the PreparedStatement interface. So, it has more features than the PreparedStatement interface. It is used to implement a parameterized SQL statement that invokes procedure or function in the database. A stored procedure works like a method or function in a class. It supports the IN and OUT parameters.

- The CallableStatement instance is created by calling the prepareCall method of the Connection object.

```
CallableStatementcallStmt = con.prepareCall("{call procedures(?,?)}");
```



Create And Execute Statement

(ii) Execute The Query

There are 4 important methods to execute the query in Statement interface. These are explained below:

- **ResultSet executeQuery(String sql)**
- **int executeUpdate(String sql)**
- **boolean execute(String sql)**
- **int []executeBatch()**

a) **ResultSet executeQuery(String sql)**

The executeQuery() method in Statement interface is used to execute the SQL query and retrieve the values from DB. It returns the ResultSet object. Normally, we will use this method for the SELECT query.

b) **executeUpdate(String sql)**

The executeUpdate() method is used to execute value specified queries like INSERT, UPDATE, DELETE (DML statements), or DDL statements that return nothing. Mostly, we will use this method for inserting and updating.

c) **execute(String sql)**

The execute() method is used to execute the SQL query. It returns true if it executes the SELECT query. And, it returns false if it executes INSERT or UPDATE query.

d) **executeBatch()**

This method is used to execute a batch of SQL queries to the Database and if all the queries get executed successfully, it returns an array of update



Retrieve Results

- When we execute the queries using the `executeQuery()` method, the result will be stored in the `ResultSet` object.
- The returned `ResultSet` object will never be null even if there is no matching record in the table. `ResultSet` object is used to access the data retrieved from the Database.

`ResultSet rs 1= statemnt1.executeQuery(QUERY);`

- The `executeQuery()` method for the `SELECT` query. When someone tries to execute the insert/update query, it will throw `SQLException` with the message “`executeQuery` method can not be used for update”.
- A `ResultSet` object points to the current row in the Resultset. To iterate the data in the `ResultSet` object, call the `next()` method in a while loop. If there is no more record to read, it will return `FALSE`.
- `ResultSet` can also be used to update data in DB. We can get the data from `ResultSet` using getter methods such as `getInt()`, `getString()`, `getDate()`. We need to pass the column index or column name as the parameter to get the values using Getter methods.

Close Connection

- To close the JDBC connection. need to make sure that we have closed the resource after we have used it. If we don't close them properly we may end up out of connections.
- When we close the connection object, Statement and ResultSet objects will be closed automatically.

conn.close();

- From Java 7 onwards, we can close the JDBC connections automatically using a try-catch block. JDBC connection should be opened in the parenthesis of the try block. Inside the try block, you can do the database connections normally as we do.
- Once the execution exits the try block, it will automatically close the connection. In this case, we don't need to close the connection by calling conn.close method in the Java program.

```
try(Connection conn = DriverManager.getConnection(url, user, password))
{
    //database connection and operation
}
```



Java JDBC Connection Example

Implement the 6 basic steps to connect with database using JDBC in Java program.

Create Table

- Before that, first, create one table and add some entries into it.
- Below is the SQL query to create a table.

```
create table employee_details (empNum number(10), lastName varchar(50), firstName varchar(50), email varchar(255), deptNum number(10), salary number(10));
```

Insert Data Into Table

- Using the following queries, insert the data into the “employee_details” table.

```
insert into employee_details values (1001, 'Luther', 'Martin', 'ml@gmail.com', 1, 13000);
```

```
insert into employee_details values (1002, 'Murray', 'Keith', 'km@gmail.com', 2, 25000);
```

```
insert into employee_details values (1003, 'Branson', 'John', 'jb@gmail.com', 3, 15000);
```

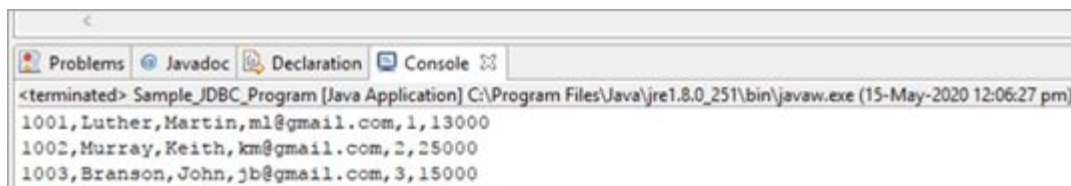
```
insert into employee_details values (1004, 'Martin', 'Richard', 'rm@gmail.com', 4, 16000);
```

```
insert into employee_details values (1005, 'Hickman', 'David', 'dh@gmail.com', 5, 17000);
```

Java Program - Oracle

```
import java.sql.*;
public class Sample_JDBC_Program {
    public static void main(String[] args) throws
        ClassNotFoundException, SQLException {
        // store the SQL statement in a string
        String QUERY = "select * from employee_details"
        //register the oracle driver with DriverManager
        Class.forName("oracle.jdbc.driver.OracleDriver");
        /*Here we have used Java 8 so opening the connection in try
        statement*/
        try(Connection conn =
            DriverManager.getConnection("jdbc:oracle:thin:system/pass123@
            localhost:1521:XE"))
        {
            Statement statemnt1 = conn.createStatement();
            //Created statement and execute it
            ResultSet rs1 = statemnt1.executeQuery(QUERY);
            {
```

```
        //Get the values of the record using while loop
        while(rs1.next())
        {
            int empNum = rs1.getInt("empNum");
            String lastName = rs1.getString("lastName");
            String firstName = rs1.getString("firstName");
            String email = rs1.getString("email");
            String deptNum = rs1.getString("deptNum");
            String salary = rs1.getString("salary");
            //store the values which are retrieved using ResultSet and print it
            System.out.println(empNum + "," +lastName+ "," +firstName+ ","
            +email +"," +deptNum +"," +salary);
        }
    }
    catch (SQLException e) {
        //If exception occurs catch it and exit the program
        e.printStackTrace();
    }
}
```



```
<terminated> Sample_JDBC_Program [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (15-May-2020 12:06:27 pm)
1001,Luther,Martin,ml@gmail.com,1,13000
1002,Murray,Keith,km@gmail.com,2,25000
1003,Branson,John,jb@gmail.com,3,15000
```



Java Program - Mysql

```
import java.sql.*;
public class javamysql {
    public static void main(String arg[])
    { Connection connection = null;
        try {
            // below two lines are used for connectivity.
            Class.forName("com.mysql.cj.jdbc.Driver");
            connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","mydbuser", "mydbuser");
            // mydb is database; mydbuser is name of database ;mydbuser is password of database
            Statement statement;
            statement = connection.createStatement();
            ResultSet resultSet;
            resultSet = statement.executeQuery("select * from designation");
            int code;
            String title;
            while (resultSet.next()) {
                code = resultSet.getInt("code");
                title = resultSet.getString("title").trim();
                System.out.println("Code : " + code + " Title : " + title);
            }
            resultSet.close();
            statement.close();
            connection.close();
        }
        catch (Exception exception) {
            System.out.println(exception);
        }
    }
}
```

JAVA APPLETS

Learning Objectives:

- **Introduction**

- CUI (Character User Interface)
- GUI (Graphical User Interface)
- Abstract Window Toolkit (AWT)

- **What are Applets in Java?**

- Applet Basics
- Life Cycle of an Applet
- Types of Applets in Java
- How to run an Applet
- Sample Applet Programs
- Advantages and Disadvantages
- Applet Features over HTML

Introduction

There are two ways that users can interact with the programme:

They are:

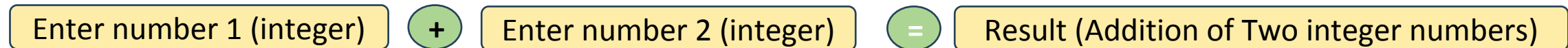
- CUI (Character User Interface)
- GUI (Graphical User Interface)

- **Character User Interface(CUI):**

In CUI user interacts with the application by typing characters or commands. In CUI user should remember the commands. It is not user friendly.

Example: In a Java program, we can add two numbers(integers) , declare the numbers, do the addition, and report the results. It makes it hard for the end user to understand how to interact with this program throughout compilation and execution (feed input and get outcomes).

Let's take into consideration that the aforementioned program can have the user-friendly (GUI) following configuration:



Whenever the end-user compile the program, can enter two numbers in the given space and clicking on the “equal button”, can get the results in the specified space, which is easy to understand. This is an example for GUI for addition of two numbers.

• Graphical User Interface (GUI)

Java **Abstract Window Toolkit (AWT)** is an *Application Program Interface (API)* to develop GUI or window-based application in java. The Abstract Window Toolkit(AWT) support for applets. The AWT contains numerous classes and methods that allow you to create and manage the GUI window.

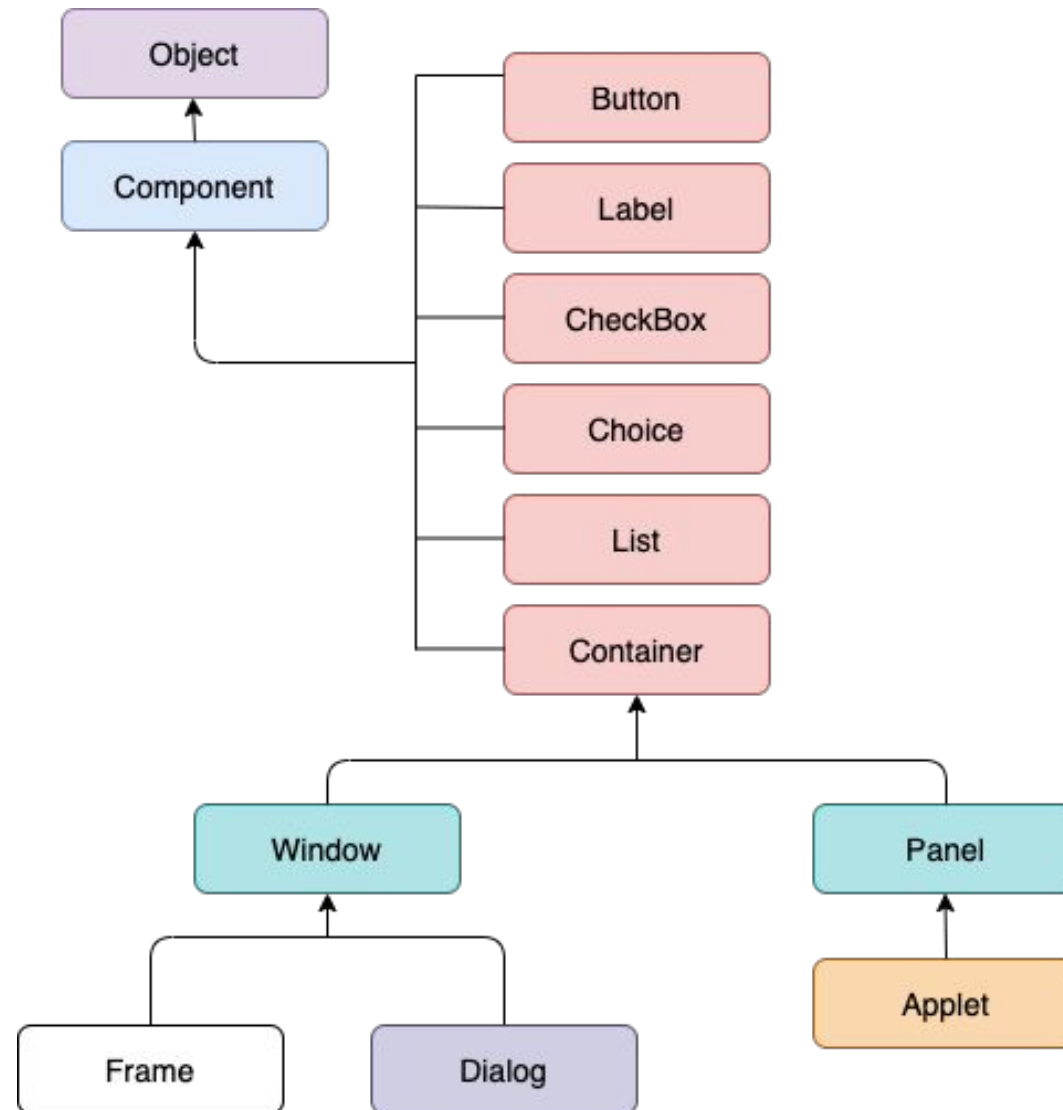
Abstract Window Toolkit (AWT)

- AWT is an Application programming interface (API) for creating Graphical User Interface (GUI) in Java. It allows Java programmers to develop window-based applications.
- AWT represents a class library to develop applications using GUI.
- AWT provides various components *like button, label, checkbox*, etc. used as objects inside a [Java](#) Program. [AWT](#) components use the resources of the operating system, i.e., they are platform-dependent, which means, component's view can be changed according to the view of the operating system. The classes for AWT are provided by the **Java.awt package** for various AWT components.
- The **Java.awt** package contains classes and interfaces that help create graphical user interfaces and enable more user-friendly programme interaction.

JAVA APPLETS



The hierarchy of Java AWT :



- **Object Class:**

Object class is the superclass of all Java classes. All Java classes inherited from this class.

- **Component Class:**

The component class, which takes the top position in the AWT hierarchy, is an abstract class that contains all of the component's screen-visible characteristics. The Component object gives details about the foreground and background colours that are currently selected. Additionally, it provides information about the presently selected font colour.

- **Container:**

A component called the container includes additional components like a button, textfield, label, etc. It is a subclass of the Component class, too.

- **Panel:**

The panel can be defined as a container that can be used to hold other components. However, it doesn't contain the title bar, menu bar, or border.

- **Window:**

A window can be defined as a container that doesn't contain any border or menu bar. It creates a top-level view. However, we must have a frame, dialog, or another window for creating a window.

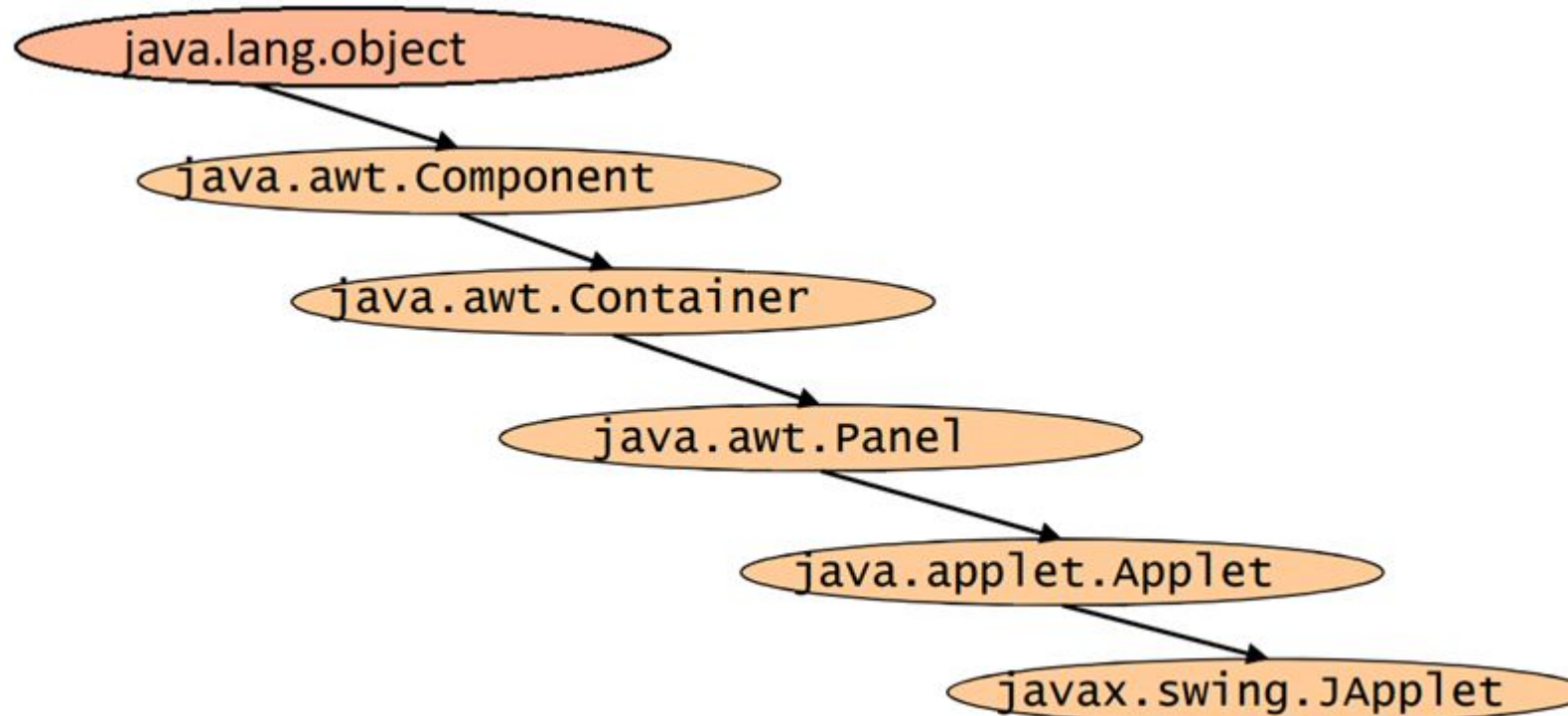
- **Frame:**

The frame is a subclass of Window. It can be defined as a container with components like button, textfield, label, etc. In other words, AWT applications are mostly created using frame container.

What are Applets in Java?

The interactive components of web applications are provided by Java applets, which can be easily run by many different platforms' browsers. They can launch automatically when the pages are browsed and are tiny, portable Java programmes embedded in HTML pages.

- Applet inherits `awt Component` class and `awt Container` class



Applet Basics:

- All applets are subclasses of *Applet*. Thus, all applets must *import java.applet*. Applets must also *import java.awt* package.
- Applets are not executed by the console-based Java run-time interpreter. Rather, they are executed by either a *Web browser* or an *applet viewer*.
- Execution of an *applet does not begin at main() [In other words, there is no main() method in an Applet]*. Output to your applet's window is not performed by `System.out.println()`. Rather, it is handled with various AWT methods, such as *drawString()*, which outputs a string to a specified X,Y location. Input is also handled differently than in an application.
- Once an applet has been compiled, it is included in an *HTML file using the APPLET tag*. The applet will be executed by a Java-enabled web browser when it encounters the APPLET tag within the HTML file.
- To view and test an applet more conveniently, simply include a comment at the head of your Java source code file that contains the *APPLET tag*.

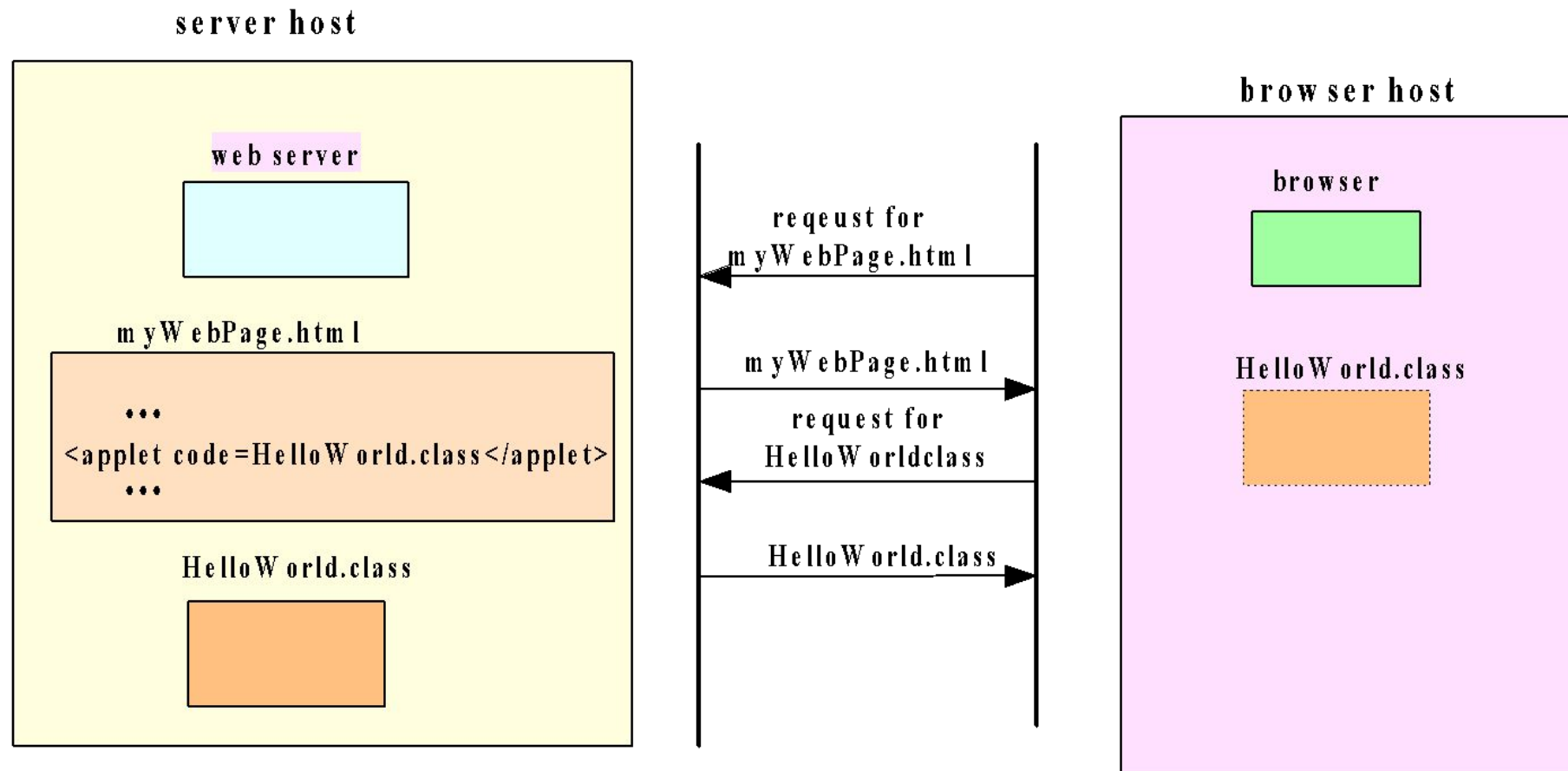
Java applets are one of three kinds of Java programs:

- An *application* is a standalone program that can be invoked from the command line.
- An *applet* is a program that runs in the context of a browser session.
- A *servlet* is a program that is invoked on demand on a server program and that runs in the context of a web server process.

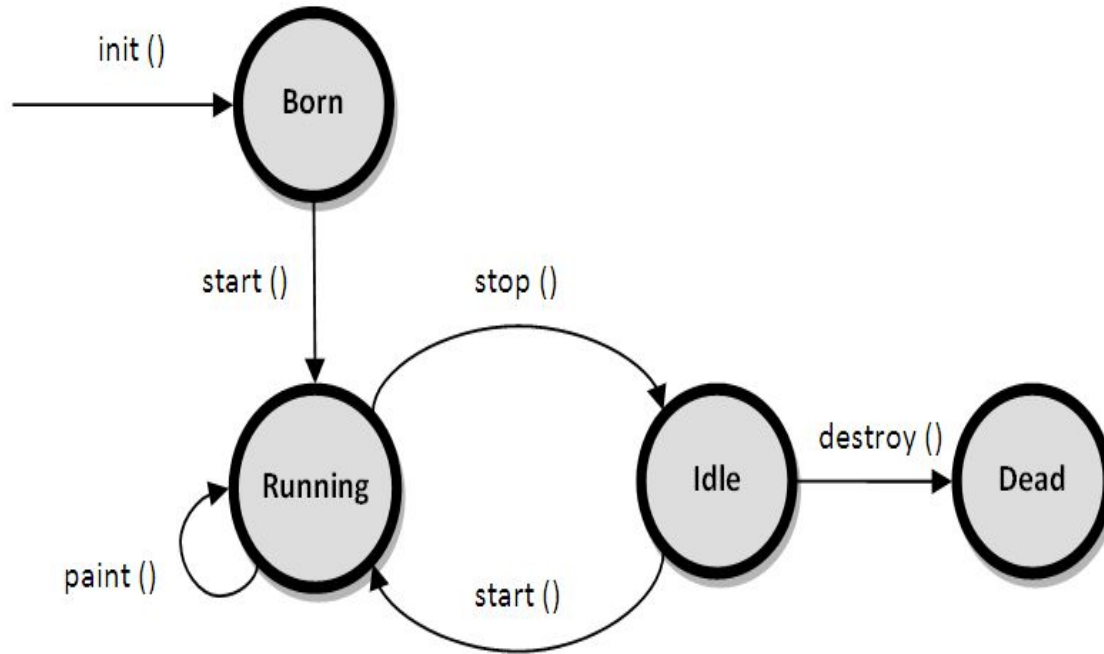
JAVA APPLETS



- Applets are programs stored on a web server, like web pages.
- When an applet is referred to in a web page that has been fetched and processed by a browser, the browser generates a request to fetch (or download) the applet program, then executes the program in the browser's execution context, *on the client host*.



Life Cycle of an Applet



It's essential in understanding the sequence whereby the different methods indicated in the figure are invoked. The following methods are called in this order when an applet begins:

- 1.**init()**
- 2.**start()**
- 3.**paint()**

When an applet is terminated, the following sequence of method calls takes place:

- 1.**stop()**
- 2.**destroy()**

- **public void init():** is used to initialize the Applet. It is invoked only once.
- **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
- **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- **public void destroy():** is used to destroy the Applet. It is invoked only once.
- **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc. Within the method **paint()** we will call the **drawString()** method to print a text message in the applet window.

Types of Applets in Java

1. Local Applet
2. Remote Applet

Local Applet

- *Local Applet* is developed locally and stored in the local system. A web page doesn't need to get the information from the internet when it finds the local Applet in the system. It is specified or defined by the file name or pathname. There are two attributes used in defining an applet, i.e., the codebase that specifies the path name and code that defines the name of the file that contains Applet's code..

- **Example:**

```
<applet  
  codebase = "MyOwnApplet"  
  code = "FirstApplet.class"  
  width = 120  
  height = 120>  
</applet>
```



Fig : Local Applet

Remote Applet

- *Remote applets* are stored in a remote computer and an Internet connection is needed to access them. The remote applet is designed and developed by other developers. To find and load a remote applet, you need to know the address of the network applet, i.e., the Uniform Resource Locator (URL).

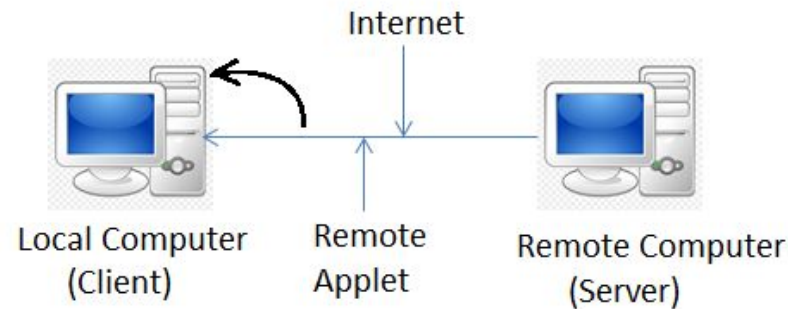


Fig : Remote Applet

- **Example:**

```
<applet  
  codebase = "http://www.myconnect.com/applets/"  
  code = "FirstApplet.class"  
  width = 120  
  height = 120>  
</applet>
```

Difference between Local Applet and Remote Applet:

Local Applet	Remote Applet
There is no need to define the Applet's URL in Local Applet.	We need to define the Applet's URL in Remote Applet.
Local Applet is available on our computer.	Remote Applet is not available on our computer.
To use it or access it, we don't need Internet Connection.	To use it or access it on our computer, we need an Internet Connection.
It is written on our own and then embedded into the web pages.	It was written by another developer.
We don't need to download it.	It is available on a remote computer, so we need to download it to our system.

How to run an applet

An applet can be run two ways:

1. Through HTML file.
2. Using the appletviewer tool (for testing purposes).

1. Through HTML file:

For an applet to run in a web browser, we must write a short HTML text file that contains a tag to load an applet. For this, we can use **<applet> or <object>** tags. The HTML <applet> tag specifies an applet. It embeds Java applets in HTML documents. It does not support HTML5.

Through HTML file:

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML applet Tag</title>
  </head>

  <body>
    <applet code = "FirstApplet.class" width = "300" height = "200"></applet>
  </body>

</html>
```

- In the **HTML text file above**, the code attribute of the <applet> tag specifies the applet class to execute.
- The width and height attributes are also required. They define the initial size of the panel on which the applet is running.
- The applet command must be closed with the </applet> tag.

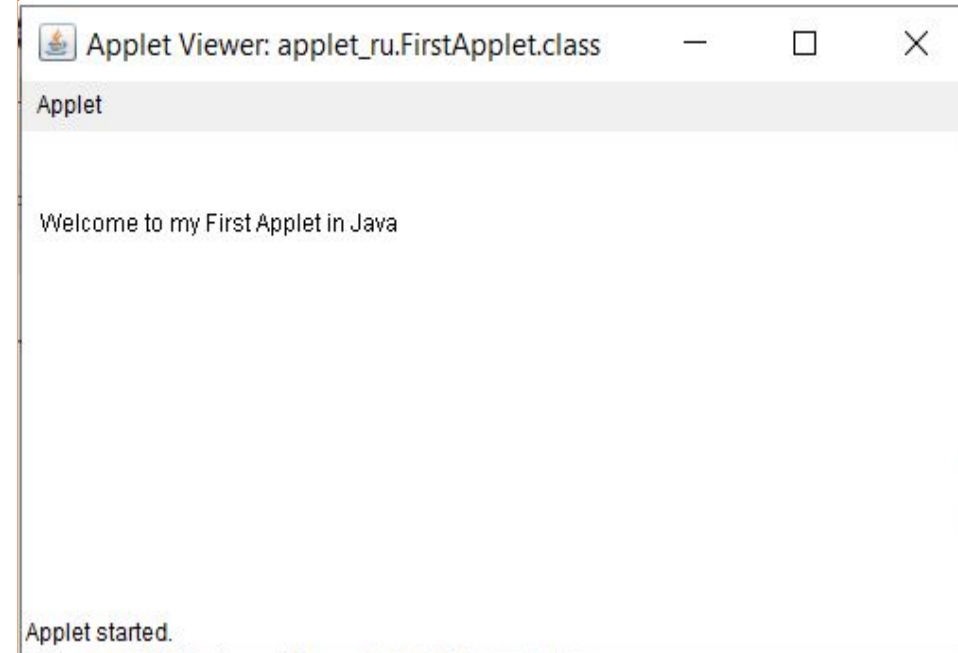
JAVA APPLETS



Below is the applet class embedded in the HTML file above:

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

public class FirstApplet extends Applet
{
    public void paint (Graphics gh)
    {
        gh.drawString("Welcome to my first applet..!", 300, 150);
    }
}
```



2. Using the appletviewer tool:

The appletviewer runs an applet in the window. It is usually the fastest and easiest way to test an applet. Create an applet containing the <applet> tag in the comment and compile it. It is for testing purposes only.

//FirstApplet.java

```
import java.applet.Applet;
import java.awt.Graphics;
public class FirstApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("welcome to my first applet",10,50);
    }
}
```

```
/* This is saved as FirstApplet.html
<applet code="FirstApplet.class" width="300"
height="300">
</applet>
*/
```

To run an applet using the appletviewer tool, write in the command prompt:

c:\>javac FirstApplet.java

c:\>appletviewer FirstApplet.java

c:\>appletviewer FirstApplet.html

Here,

- **javac** is the compiler that compiles java codes using a command line.
- **FirstApplet.java** is the applet class to be tested.
- **appletviewer** is a tool that tests the applet class.

Output

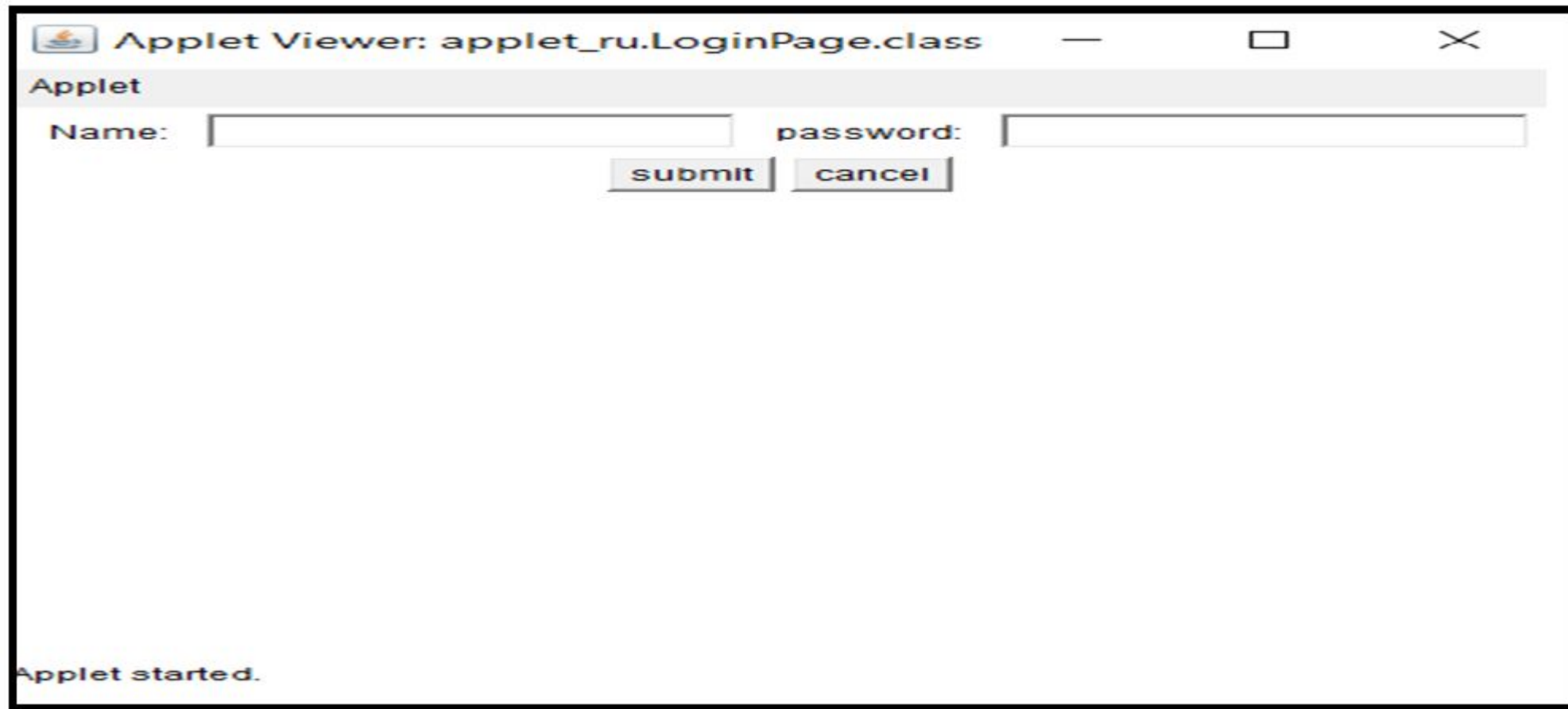


Sample Applet Program-2:

```
package applet_ru;
import java.awt.*;
import java.applet.*;
public class LoginPage extends Applet
{
    TextField Uname,pass;
    Button b1,b2;
    public void init()
    {
        Label n=new Label("UserName:",Label.CENTER);
        Label p=new Label("password:",Label.CENTER);
        Uname=new TextField(20);
        pass=new TextField(20);
        pass.setEchoChar('$');
        b1=new Button("submit");
        b2=new Button("cancel");
```

```
        add(n);
        add(Uname);
        add(p);
        add(pass);
        add(b1);
        add(b2);
        n.setBounds(70,90,90,60);
        p.setBounds(70,130,90,60);
        Uname.setBounds(280,100,90,20);
        pass.setBounds(200,140,90,20);
        b1.setBounds(100,260,70,40);
        b2.setBounds(180,260,70,40);
    }
    public void paint(Graphics g)
    {
        repaint();
    }
}
```

Output:



Sample Applet Program-3

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

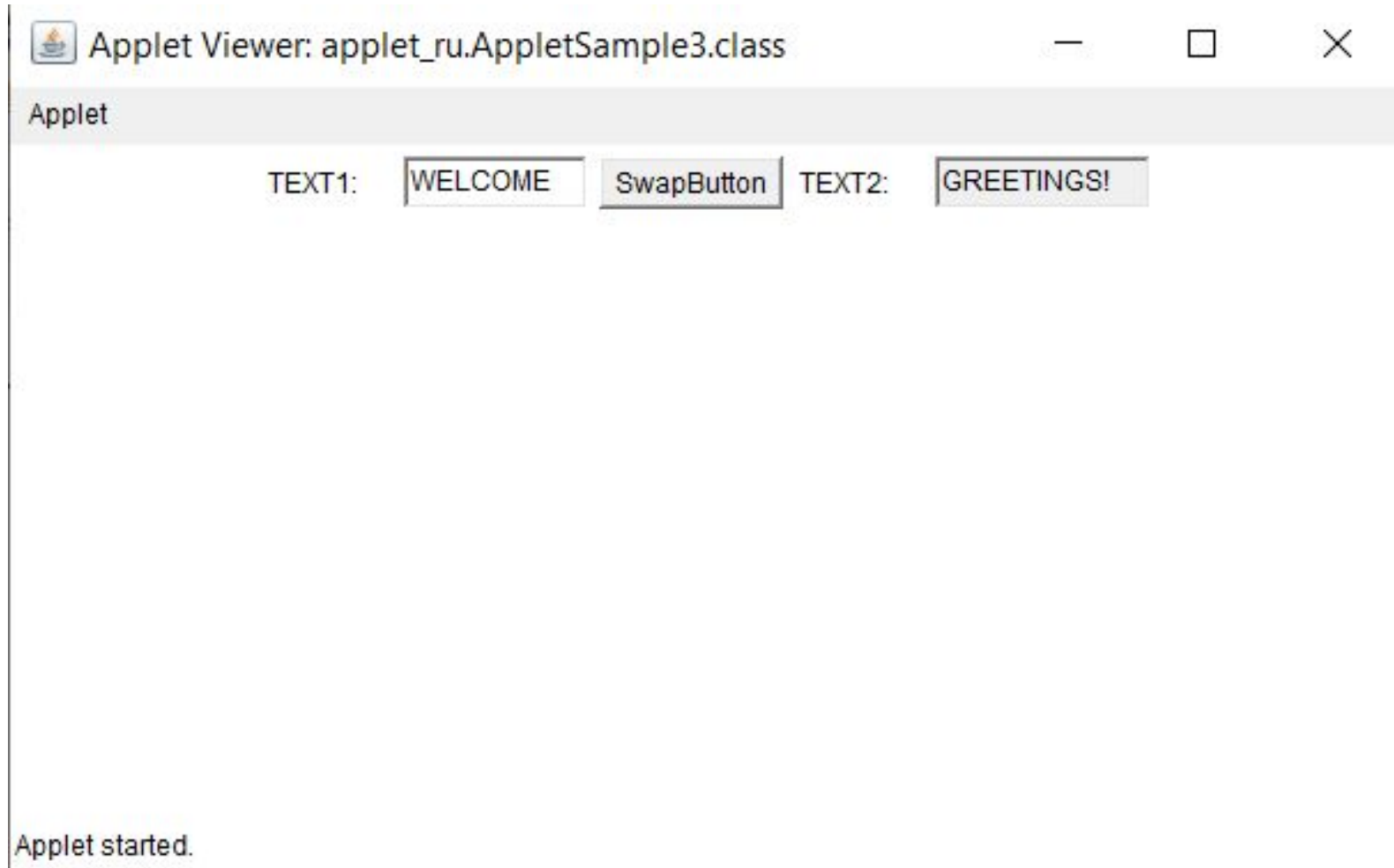
public class AppletSample3 extends Applet
implements ActionListener {
    public void init() {
        Label label1 = new Label("TEXT1: ");
        textField1 = new TextField("WELCOME");
        swapBtn = new Button("SwapButton");
        swapBtn.addActionListener(this);
        Label label2 = new Label("TEXT2: ");
        textField2 = new TextField("GREETINGS!");
        textField2.setEditable(false);
        add(label1);
        add(textField1);
        add(swapBtn);
        add(label2);
        add(textField2);
    }
}
```

```
public void actionPerformed(ActionEvent e)
{
    String temp = textField1.getText();
    textField1.setText(textField2.getText());
    textField2.setText(temp);
}

TextField textField1, textField2;
Button swapBtn;
}
```

The above applet program is used to swap Text1 string to Text2 and vice versa while clicking on the button “SwapButton”.

Output:



Commonly used 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.

Some of the AWT supplies User Interface components

- Text Fields
- Text area
- Labels
- Checkboxes
- Buttons
- Lists
- Sliders and scrollbars
- Drawing areas
- Menus
- Containers

EVENT HANDLING

The pathway to efficient applet programming is how to handle events. The majority of events to which the applet will react come from the user. The most frequent various types of events processed involve those triggered by the keyboard, mouse, and various controllers, including push buttons. Events are supported by the *java.awt.event* package.

Steps to perform Event Handling

- Register the component with the Listener
- Registration Methods
- For registering the component with the Listener, many classes provide the registration methods.

For example:

Button	• <code>public void addActionListener(ActionListener a){}</code>
MenuItem	• <code>public void addActionListener(ActionListener a){}</code>
TextField	<ul style="list-style-type: none">• <code>public void addActionListener(ActionListener a){}</code>• <code>public void addTextListener(TextListener a){}</code>
TextArea	• <code>public void addTextListener(TextListener a){}</code>
Checkbox	• <code>public void addItemListener(ItemListener a){}</code>
Choice	• <code>public void addItemListener(ItemListener a){}</code>
List	<ul style="list-style-type: none">• <code>public void addActionListener(ActionListener a){}</code>• <code>public void addItemListener(ItemListener a){}</code>

Advantages of applets

- The response time is quicker because it operates on the client-side. It is secured.
- Web browsers must adhere to tight security regulations while using applets.
- It is compatible with browsers on various operating systems, including Linux, Windows, and Mac OS.

Disadvantages of applets

- The client browser requires a plugin to run the applet.
- The mobile browser on iOS or Android does not run any Java applets. Desktop browsers have dropped support for Java applets along with the rise of mobile operating systems.

Applet features over HTML

- It displays the dynamic web pages of a web application.
- Play audio files.
- Display documents.
- Play animations.

Java SWING - Introduction

- Swing in Java is a lightweight and platform-independent class of the Java foundation class. It's used to make applications that run in windows.
- It contains elements such as a button, a scroll bar, and a text field. A graphical user interface is created by combining all of these elements.
- Swing Framework includes several classes that provide more vital and more versatile GUI (Graphical User Interface) components than AWT(Abstract Window Toolkit).
- Swing is a Sun Microsystems-published official Java GUI toolkit that closely resembles the look and feels of modern Java GUIs. It is used to create the graphical user interface for Java Class.

Features of Java Swing





Java SWING FEATURES

Platform Independent

It is platform-independent, as the swing components used to construct the program are not platform-specific. It works on any platform and in any location.

Customizable

Swing controls are simple to customize. It can be changed, and the swing component application's visual appearance is independent of its internal representation.

Plugging

- Java Swing has pluggable look and feel. This feature allows users to change the appearance and feel of Swing components without having to restart the application.
- The Swing library will enable components to have the same look and feel across all platforms, regardless of where the program is running.
- The Swing library provides an API that allows complete control over the appearance and feel of an application's graphical user interface



Java SWING FEATURES(Contd.)

- MVC stands for **Model-View-Controller**.
- The model agrees with the state information associated with the Component in MVC terminology.
- The view determines how the component appears on screen and any aspects of the view that are influenced by the model's current state.
- The controller is in charge of determining how the component responds to the user.

Manageable Look and Feel

- It's simple to manage and configure.
- Its mechanism and composition pattern allows changes to be made to the settings while the program runs.
- Constant changes to the application code can be made without making any changes to the user interface.



Java SWING FEATURES (Contd.)

Lightweight

- Lightweight Components: The JDK's AWT has supported lightweight component development.
- A component must not rely on non-Java [O/s based] system classes to be considered light.
- The look and feel classes in Java help Swing components have their view.



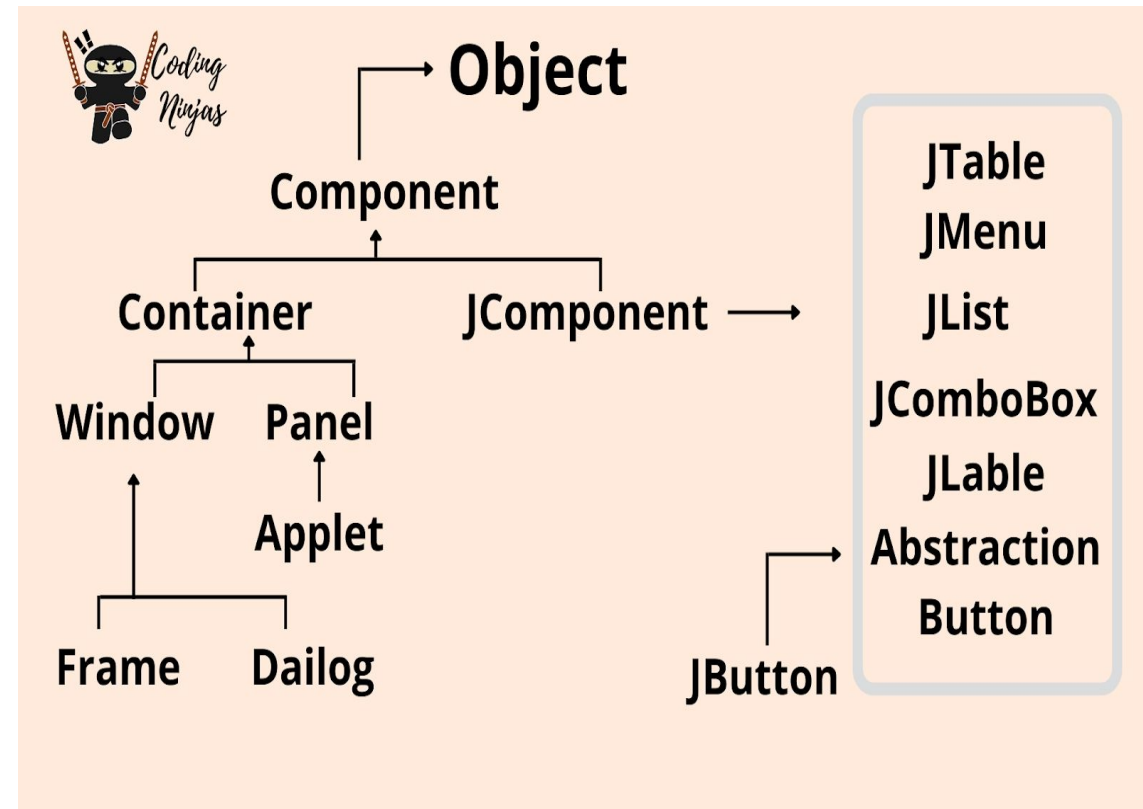
JFC and Swing

JFC stands for Java Foundation Classes, a set of classes used to create graphical user interfaces (GUIs) and add rich graphical features and interactivity to Java applications.

The Java Classes Foundation holds Java Swing (JFC).

Swing Hierarchy

JFC stands for Java Foundation Classes, a set of classes used to create graphical user interfaces (GUIs) and add rich graphical features and interactivity to Java applications. The Java Classes Foundation holds Java Swing (JFC).





What are Swing Components in Java?

- A component is independent visual control, and Java Swing Framework contains a large set of these components, providing rich functionalities and allowing high customization.
- They all are derived from JComponent class. All these components are lightweight components.
- This class offers some standard functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.
- A container holds a group of components.
- It delivers a space where a component can be managed and displayed. Containers are of two types:

Swing Components

Top-level Containers	It inherits the Component and Container of AWT.
	We cannot contain it within other containers.
	Heavyweight.
	Example: JFrame, JDialog, JApplet
Lightweight Containers	It inherits JComponent class.
	It is a general-purpose container.
	We can use it to organize related components together.
	Example: JPanel

Top Swing components in Java

Jbutton

- [JButton](#) class to create a push button on the UI.
- The button can include some display text or images.
- It yields an event when clicked and double-clicked.
- Can implement a JButton in the application by calling one of its constructors.

Syntax:

```
JButton okBtn = new JButton("Click");
```

Display:



JLabel

- Use [JLabel](#) class to render a read-only text label or images on the UI. It does not generate any event.

Syntax:

```
JLabel textLabel = new JLabel("This is 1st L...");
```

- This constructor returns a label with specified text.

```
JLabel imgLabel = new JLabel(carIcon);
```

It returns a label with a car icon.



The JLabel Contains four constructors. They are as follows:

1. JLabel()
2. JLabel(String s)
3. JLabel(Icon i)
4. JLabel(String s, Icon i, int horizontalAlignment)



JTextField

The JTextField renders an editable single-line text box. Users can input non-formatted text in the box.

Can initialize the text field by calling its constructor and passing an optional integer parameter.

This parameter sets the box width measured by the number of columns.

Also, it does not limit the number of characters that can be input into the box.

Syntax:

```
JTextField txtBox = new JTextField(50);
```

It is the most widely used text component.

It has three constructors:

1. JTextField(int cols)
2. JTextField(String str, int cols)
3. JTextField(String str)



Note: cols represent the number of columns in the text field.

JCheckBox

The JCheckBox renders a check-box with a label.

The check-box has two states, i.e., on and off. On selecting, the state is set to "on," and a small tick is displayed inside the box.

Syntax:

```
CheckBox chkBox = new JCheckBox("Java Swing", true);
```

- It returns a checkbox with the label Pepperoni pizza.
- Notice the second parameter in the constructor.
- It is a boolean value that denotes the default state of the check-box.
- True means the check-box defaults to the "on" state.



JRadioButton

- A radio button is a group of related buttons from which can select only one.
- Use JRadioButton class to create a radio button in Frames and render a group of radio buttons in the UI.
- Users can select one choice from the group.

Syntax:

```
JRadioButton jrb = new JRadioButton("Easy");
```

Display:

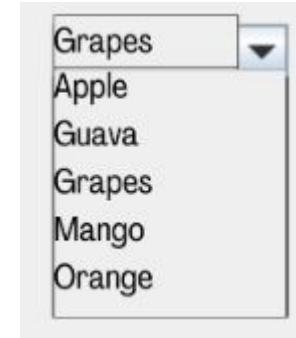


JComboBox

- The combo box is a combination of text fields and a drop-down list
- Use JComboBox component to create a combo box in Swing.

Syntax:

```
JcomboBox jcb = new JComboBox(name);
```



JTextArea

In Java, the Swing toolkit contains a JTextArea Class. It is under package javax.swing.JTextArea class. It is used for displaying multiple-line text.

Declaration:

```
public class JTextArea extends JTextComponent
```

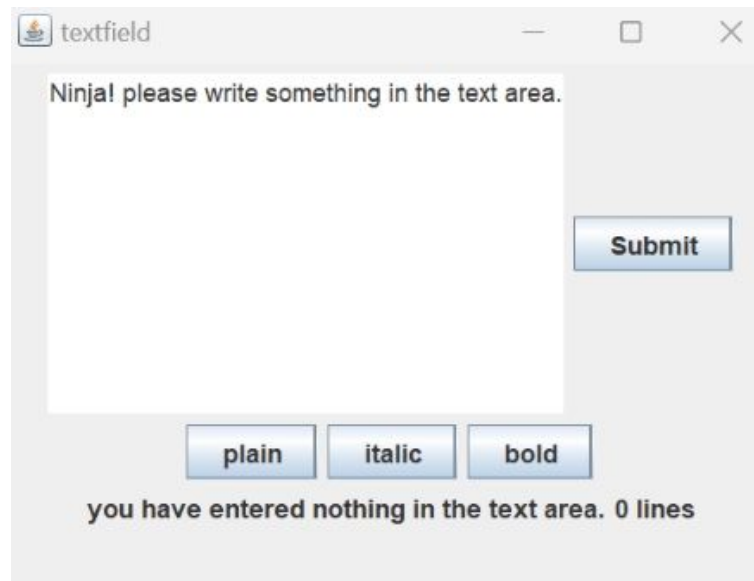

Syntax:

```
TextArea textArea_area=new JTextArea("Ninja! please write something in the text area.");
```

The JTextArea Contains four constructors. They are as follows:

1. JTextArea()
2. JTextArea(String s)
3. JTextArea(int row, int column)
4. JTextArea(String s, int row, int column)

Display:



JPasswordField

- In Java, the Swing toolkit contains a JPasswordField Class.
- It is under package javax.swing.JPasswordField class.
- It is specifically used for the password, and we can edit them.

Declaration:

public class JPasswordField extends JTextField

Syntax:

JPasswordField password = new JPasswordField();

The JPasswordFieldContains 4 constructors. They are as follows:

- JPasswordField()
- JPasswordField(int columns)
- JPasswordField(String text)
- JPasswordField(String text, int columns)



JTable

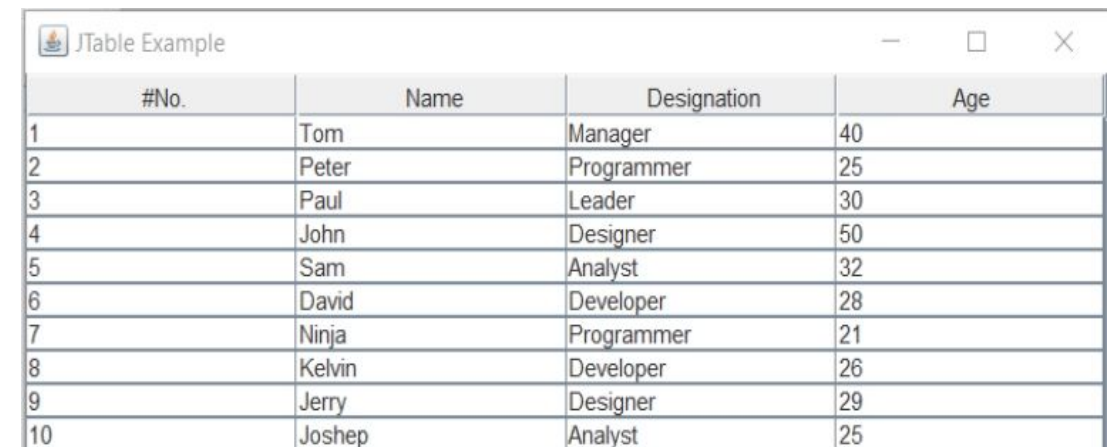
- In Java, the Swing toolkit contains a JTable Class.
- It is under package javax.swing.JTable class.
- It is used to draw a table to display data.

Syntax:

`JTable table = new JTable(table_data, table_column);`

The JTable contains two constructors. They are as follows:

1. `JTable()`
2. `JTable(Object[][] rows, Object[] columns)`



#No.	Name	Designation	Age
1	Tom	Manager	40
2	Peter	Programmer	25
3	Paul	Leader	30
4	John	Designer	50
5	Sam	Analyst	32
6	David	Developer	28
7	Ninja	Programmer	21
8	Kelvin	Developer	26
9	Jerry	Designer	29
10	Joshep	Analyst	25



JList

- In Java, the Swing toolkit contains a JList Class.
- It is under package javax.swing.JList class.
- It is used to represent a list of items together.
- We can select one or more than one items from the list.

Declaration:

public class JList extends JComponent implements Scrollable, Accessible

Syntax:

```
DefaultListModel<String> list1 = new DefaultListModel<>();
```

```
list1.addElement("Apple");
```

```
list1.addElement("Orange");
```

```
list1.addElement("Banan");
```

```
list1.addElement("Grape");
```

```
JList<String> list_1 = new JList<>(list1);
```

The JListContains 3 constructors. They are as follows:

`JList()`

`JList(ary[] listData)`

`JList(ListModel<ary> dataModel)`

Display:



JOptionPane

In Java, the Swing toolkit contains a JOptionPane Class. It is under package javax.swing.JOptionPane class. It is used for creating dialog boxes for displaying a message, confirm box, or input dialog box.

Declaration:

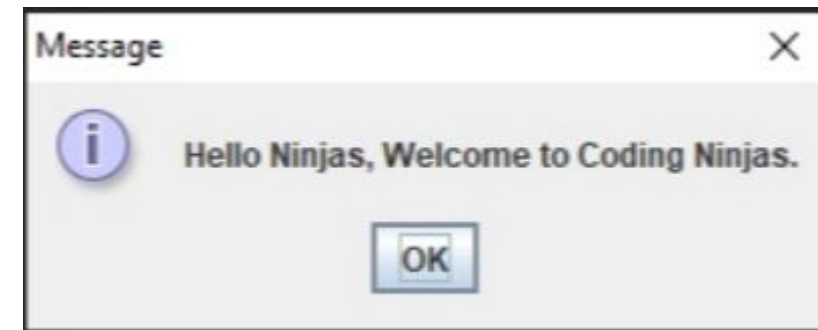
public class JOptionPane extends JComponent implements Accessible

Syntax:

```
JOptionPane.showMessageDialog(jframe_obj, "Good Morning, Evening & Night.");
```

The JOptionPane Contains 3 constructors. They are as following:

1. JOptionPane()
2. JOptionPane(Object message)
3. JOptionPane(Object message, int messageType)



JScrollBar

- In Java, the Swing toolkit contains a JScrollBar class.
- It is under package javax.swing.JScrollBar class.
- It is used for adding horizontal and vertical scrollbars.

Declaration:

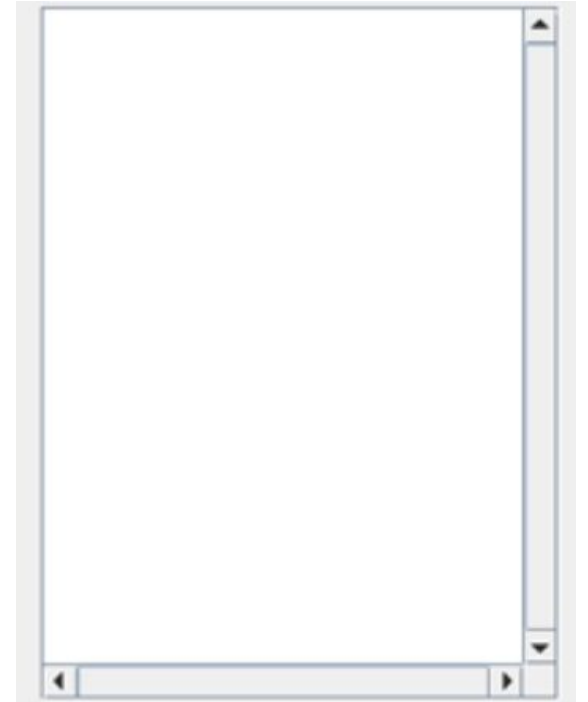
public class JScrollBar extends JComponent implements Adjustable, Accessible

Syntax:

```
JScrollBar scrollBar = new JScrollBar();
```

The JScrollBar contains 3 constructors. They are as following:

1. JScrollBar()
2. JScrollBar(int orientation)
3. JScrollBar(int orientation, int value, int extent, int min_, intmax_)





JMenuBar, JMenu and JMenuItem

- In Java, the Swing toolkit contains a JMenuBar, [JMenu](#), and JMenuItem class.
- It is under package javax.swing.JMenuBar, javax.swing.JMenu and javax.swing.JMenuItem class.
- The JMenuBar class is used for displaying menubar on the frame.
- The JMenu Object is used to pull down the menu bar's components.
- The JMenuItem Object is used for adding the labeled menu item.

JMenuBar, JMenu and JMenuItem Declarations:

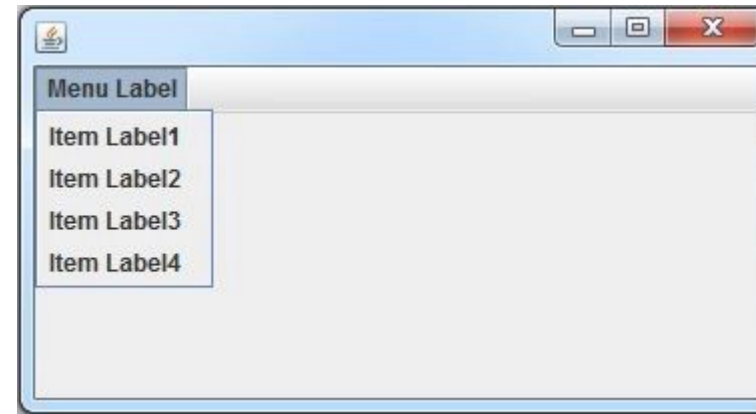
public class JMenuBar extends JComponent implements MenuElement, Accessible

public class JMenu extends JMenuItem implements MenuElement, Accessible

public class JMenuItem extends AbstractButton implements Accessible, MenuElement

Syntax:

```
JMenuBar menu_bar = new JMenuBar();  
JMenu menu = new JMenu("Menu");  
menuItem1 = new JMenuItem("Never");  
menuItem2 = new JMenuItem("Stop");  
menuItem3 = new JMenuItem("Learing");  
menu.add(menuItem1);  
menu.add(menuItem2);  
menu.add(menuItem3);
```



JPopupMenu

- In Java, the Swing toolkit contains a JPopupMenu Class.
- It is under package javax.swing.JPopupMenu class.
- It is used for creating popups dynamically on a specified position.

Declaration:

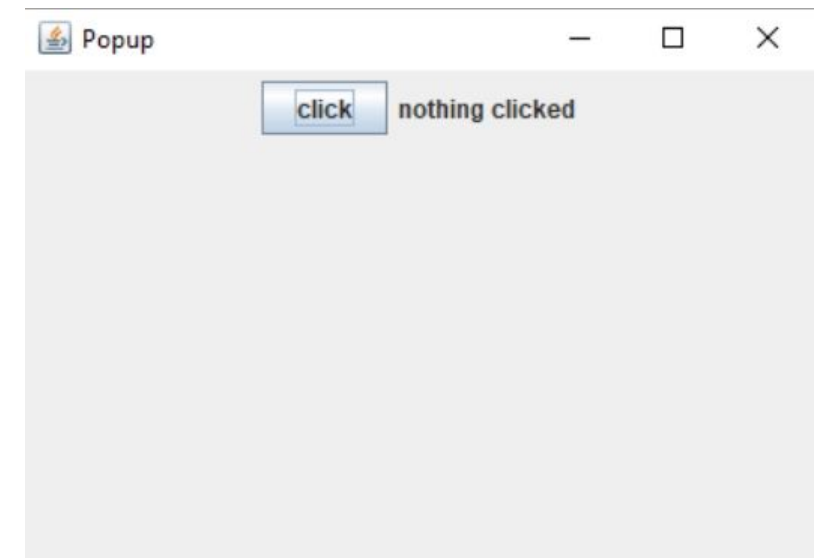
public class JPopupMenu extends JComponent implements Accessible, MenuElement

Syntax:

```
final JPopupMenu popupmenu1 = new JPopupMenu("Edit");
```

The JPopupMenu Contains 2 constructors. They are as follows:

1. JPopupMenu()
2. JPopupMenu(String label)





JCheckBoxMenuItem

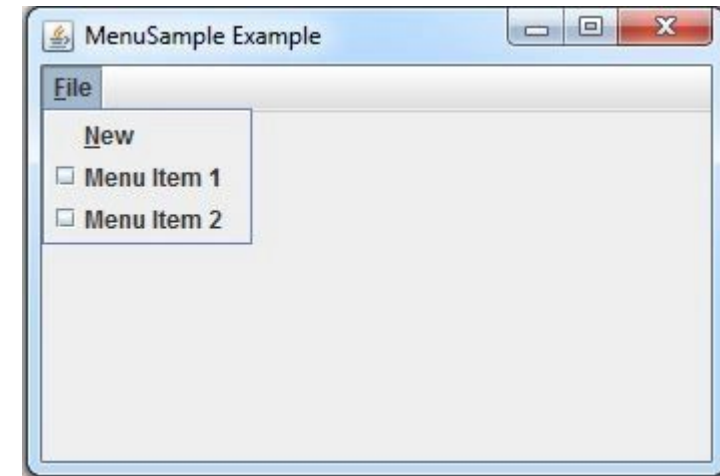
- In Java, the Swing toolkit contains a JCheckBoxMenuItem Class.
- It is under package javax.swing.JCheckBoxMenuItem class.
- It is used to create a checkbox on a menu.

Syntax:

```
JCheckBoxMenuItem item = new JCheckBoxMenuItem("Option_1");
```

The JCheckBoxMenuItem Contains 2 constructors. They are as following:

1. JCheckBoxMenuItem()
2. JCheckBoxMenuItem(Action a)
3. JCheckBoxMenuItem(Icon icon)
4. JCheckBoxMenuItem(String text)
5. JCheckBoxMenuItem(String text, boolean b)
6. JCheckBoxMenuItem(String text, Icon icon)
7. JCheckBoxMenuItem(String text, Icon icon, boolean b)



JSeparator

- In Java, the Swing toolkit contains a JSeparator Class.
- It is under package javax.swing.JSeparator class.
- It is used for creating a separator line between two components.

Declaration:

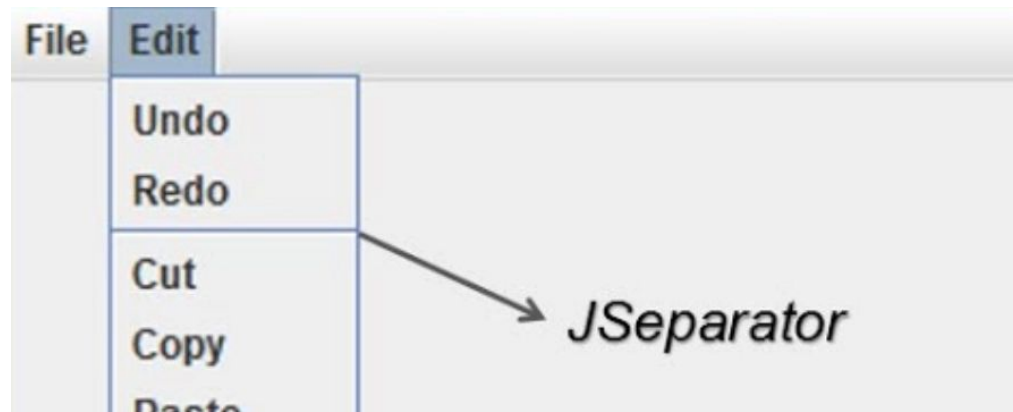
public class JSeparator extends JComponent implements SwingConstants, Accessible

Syntax:

```
jmenu_Item.addSeparator();
```

The JSeparator Contains 2 constructors. They are as following:

1. JSeparator()
2. JSeparator(int orientation)



Model-View-Controller



- MVC is a blueprint for organizing code which represents one of several design patterns. To understand the need for these patterns, think about good and bad software.
- Good software is easy to change and work with, while bad software is difficult to modify. Design patterns help us create the good kind of software.

Software evolves over time due to various factors:

- A new feature needs to be added.
- A bug needs to be fixed.
- Software needs to be optimized.
- The software design needs to be improved.

Indicators of bad software includes:

- Rigidity – Software requires a cascade of changes when a change is made in one place.
- Fragility – Software breaks in multiple places when a change is made.
- Needless complexity – Software is overdesigned to handle any possible change.
- Needless repetition – Software contains duplicate code.



- Software can be intentionally designed to handle changes that might occur later on.
- The best approach is to create components in the application that are loosely connected to each other.
- In a loosely coupled application, you can make a change to one component of an application without making changes to other parts.
- There are several principles that enable reducing dependencies between different parts of an application.
- Software design patterns represent strategies for applying software design principles. MVC is one of those patterns.

In general, a visual component is a composite of three distinct aspects:

- The way that the component looks when rendered on the screen
- The way that the component reacts to the user
- The state information associated with the component
- Throughout time, a particular architectural approach has demonstrated remarkable efficiency: namely, the Model-View-Controller, often referred to as MVC.

Model-View-Controller



- The Model-View-Controller (MVC) software design pattern is a method for separating concerns within a software application.
- As the name implies, the MVC pattern has three layers:
 - The Model defines the business layer of the application,
 - the Controller manages the flow of the application,
 - the View defines the presentation layer of the application.

Model: Handles data and business logic. Represents the business layer of the application

View: Presents the data to the user whenever asked for. Defines the presentation of the application

Controller: Entertains user requests and fetch necessary resources. Manages the flow of the application

Each of the components has a demarcated set of tasks which ensures smooth functioning of the entire application along with complete modularity.

Model-View-Controller



Model :

- Model is where the application's data objects are stored. It represents knowledge as a structure of objects.
- The model doesn't know anything about views and controllers but it can contain logic to update controller if its data changes.
- The model is quite simply the data for our application.
- The data is “modelled” in a way it's easy to store, retrieve, and edit.
- The model is how we apply rules to our data, which eventually represents the concepts our application manages.
- For any software application, everything is modelled as data that can be handled easily.
- What is a user, a book, or a message for an app? Nothing really, only data that must be processed according to specific rules. Like, the date must not be higher than the current date, the email must be in the correct format, the name mustn't be more than “x” characters long, etc.

Model-View-Controller



Model:

- Whenever a user makes any request from the controller, it contacts the appropriate model which returns a data representation of whatever the user requested.
- This model will be the same for a particular work, irrespective of how we wish to display it to the user.
- That is why we can choose any available view to render the model data.
- Additionally, a model also contains the logic to update the relevant controller whenever there is any change in the model's data.



VIEW:

- The **view** determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.
- View can also update the model by sending appropriate messages. Users interact with an application through its View.
- As the name suggests, the view is responsible for rendering the data received from the model. There may be pre-designed templates where you can fit the data, and there may even be several different views per model depending on the requirements.
- Any web application is structured keeping these three core components in mind. There may be a primary controller that is responsible for receiving all the requests and calling the specific controller for specific actions.



CONTROLLER:

- The **controller** determines how the component reacts to the user.
- Example : When the user clicks a check box, the controller reacts by changing the model to reflect the user's choice (checked or unchecked).
- The controller is like housekeeper of the application – it performs coordination between model and view to entertain a user request. The user requests are received as HTTP get or post request – for example, when the user clicks on any GUI elements to perform any action.
- The primary function of a controller is to call and coordinate with the model to fetch any necessary resources required to act.
- Usually, on receiving a user request, the controller calls the appropriate model for the task at hand.

Advantages of the MVC Architecture

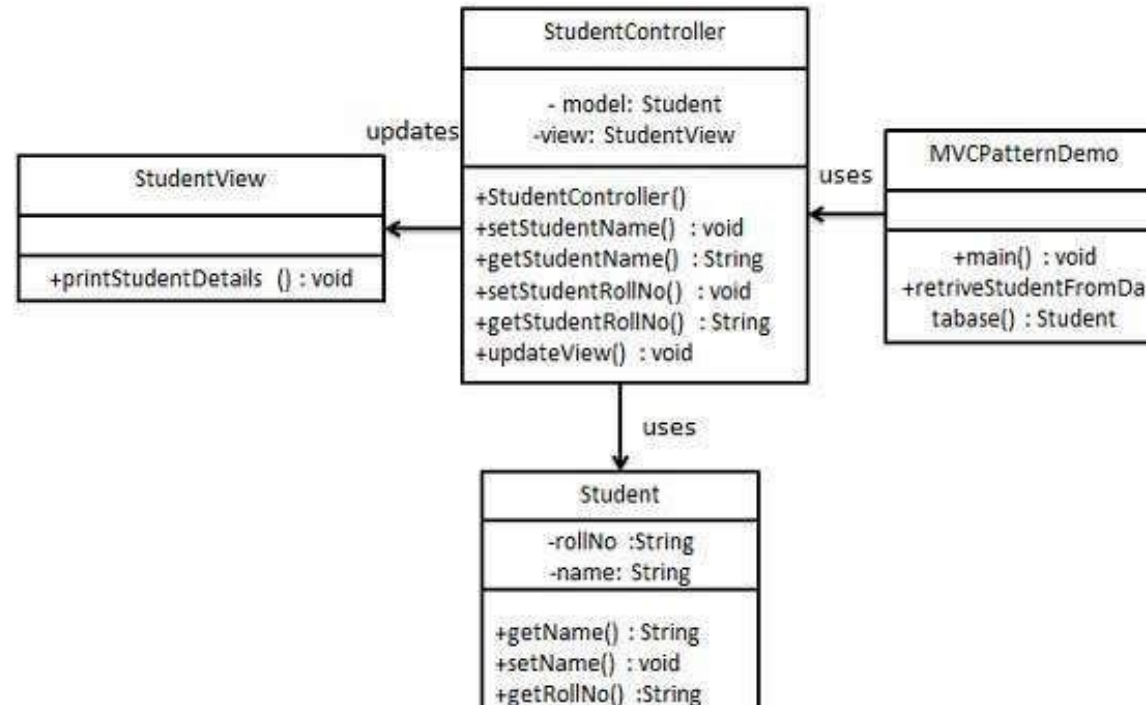


- A common problem faced by application developers these days is the support for different type of devices.
- The MVC architecture solves this problem as developers can create different interfaces for different devices, and based on from which device the request is made, the controller will select an appropriate view.
- The model sends the same data irrespective of the device being used, which ensures a complete consistency across all devices.
- The MVC separation beautifully isolates the view from the business logic.
- It also reduces complexities in designing large application by keeping the code and workflow structured.
- This makes the overall code much easier to maintain, test, debug, and reuse.

Model-View-Controller



- We are going to create a *Student* object acting as a model.
- *StudentView* will be a view class which can print student details on console
- *StudentController* is the controller class responsible for storing data in the *Student* object and updating *StudentView* accordingly.
- *MVCPatternDemo*, our demo class, will use *StudentController* to demonstrate the use of MVC pattern.





Model-View-Controller

Step 1: Create the Model :

```
public class Student {  
    private String rollNo;  
    private String name;  
    public String getRollNo() {  
        return rollNo;  
    }  
    public void setRollNo(String rollNo) {  
        this.rollNo = rollNo;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

Step 2: Create the View

```
public class StudentView {  
    public void printStudentDetails(String studentName, String studentRollNo){  
        System.out.println("Student: ");  
        System.out.println("Name: " + studentName);  
        System.out.println("Roll No: " + studentRollNo);  
    }  
}
```



Model-View-Controller

Step 3: Create the Controller

```
public class StudentController {  
    private Student model;  
    private StudentView view;  
    public StudentController(Student model, StudentView view) {  
        this.model = model;  
        this.view = view;  
    }  
    public void setStudentName(String name){  
        model.setName(name);  
    }  
    public String getStudentName(){  
        return model.getName();  
    }  
    public void setStudentRollNo(String rollNo){  
        model.setRollNo(rollNo);  
    }  
    public String getStudentRollNo(){  
        return model.getRollNo();  
    }  
    public void updateView(){  
        view.printStudentDetails(model.getName(), model.getRollNo());  
    }  
}
```

Step 4: Create the main Java file

- “MVCPatternDemo.java” fetches the student data from the database or a function (in this case we’re using a function to set the values) and pushes it on to the Student model.
- Then, it initializes the view we had created earlier.
- Further, it also initializes our controller and binds it to the model and the view.
- The updateView() method is a part of the controller which updates the student details on the console.

```
public class MVCPatternDemo {  
    public static void main(String[] args) {  
        //fetch student record based on his roll no from the database  
        Student model = retrieveStudentFromDatabase();  
        //Create a view : to write student details on console  
        StudentView view = new StudentView();  
        StudentController controller = new StudentController(model, view);  
        controller.updateView();  
        //update model data  
        controller.setStudentName("John");  
        controller.updateView();  
    }  
    private static Student retrieveStudentFromDatabase() {  
        Student student = new Student();  
        student.setName("Robert");  
        student.setRollNo("10");  
        return student;  
    }  
}
```



Step 5: Test the Result

Student:

Name: Robert

Roll No: 10

Student:

Name: John

Roll No: 10

Widgets



- Graphical User Interface (GUI) elements are the visual components that allow users to interact with software applications.
- These elements are often referred to as "widgets," which are essentially building blocks that make up the user interface.
- Each widget serves a specific purpose and provides a way for users to input information, view data, or trigger actions. Here is a list of controls in the javax.swing package

Input Components

- Buttons (JButton, JRadioButtons, JCheckBox)
- Text (JTextField, JTextArea)
- Menus (JMenuBar, JMenu, JMenuItem)
- Sliders (JSlider)
- JComboBox (uneditable) (JComboBox)
- List (Jlist)

Information Display Components

- JLabel
- Progress bars (JProgressBar)
- Tool tips (using JComponent's setToolTipText(s) method)

Choosers

- File chooser (JFileChooser)
- Color chooser (JColorChooser)

More complex displays

- Tables (JTable)
- Trees (JTree)