

**Summary of CoreJava:**

1. Java Programming Components (Java Alphabets)
2. Java Programming Concepts
3. Object Oriented Programming features

**1. Java Programming Components (Java Alphabets)**

- (a) Variables
- (b) Methods
- (c) Constructors
- (d) Blocks
- (e) Class
- (f) Interface
- (g) Abstract Class

**2. Java Programming Concepts**

- (a) Object Oriented Programming
- (b) Exception Handling Process
- (c) Java Collection Framework
- (d) Multi-Threading Concept
- (e) File Storage in Java
- (f) Networking in Java

**3. Object Oriented Programming features**

- (a) Class
- (b) Object
- (c) Abstraction
- (d) Encapsulation
- (e) Polymorphism
- (f) Inheritance

**Note:**

- ⇒ Using CoreJava Components, Concepts and Construction rules we can develop
- ⇒ NonServer-Applications (which means Stand-Alone-Applications)

**Define Stand-Alone-Application?**

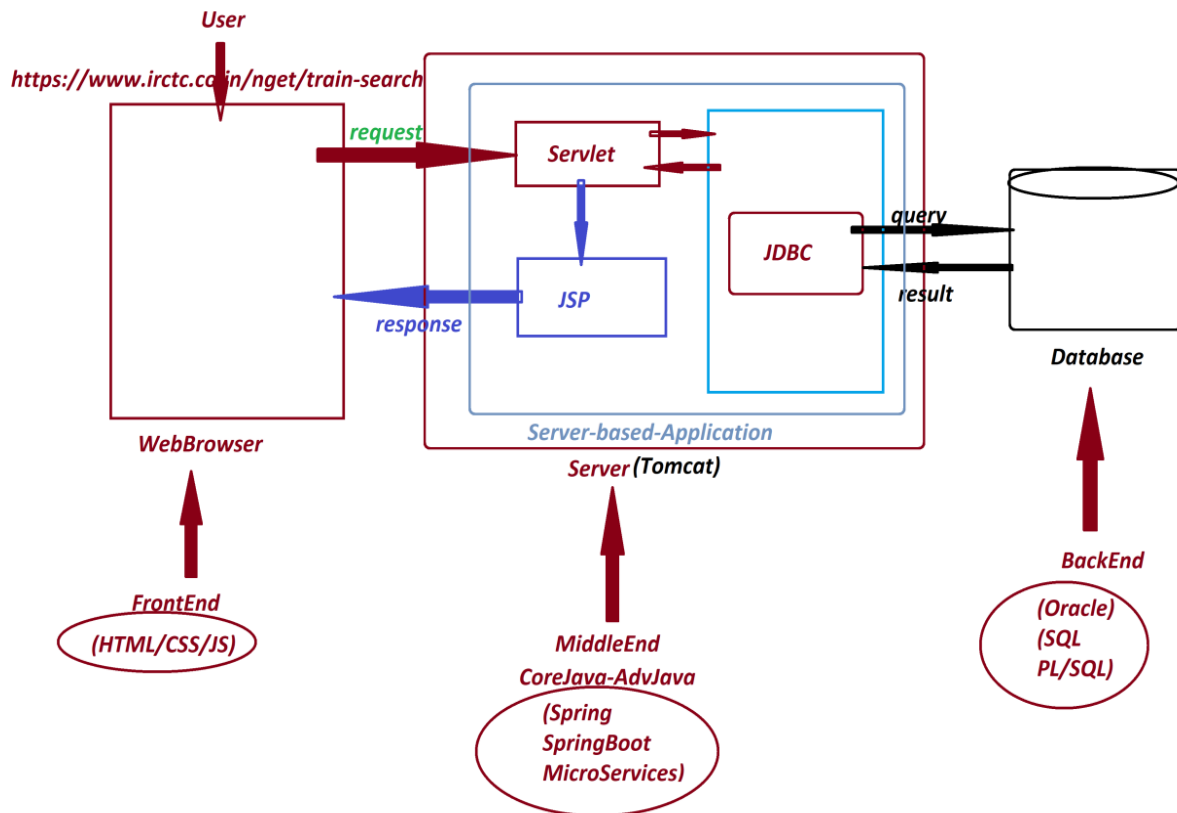
- ⇒ The Application which is installed in one Computer and performs actions in the same computer, is known as Stand-Alone-Application or NonServer-Application.

**FAQ****Define Server based Applications?**

- ⇒ The Applications which are executed in server-environment are known as Server based Applications.
- ⇒ These Server based applications are categorized into two types:
  1. *Web Applications*
  2. *Enterprise Applications*

**1. Web Applications:**

- ⇒ The Applications which are constructed using AdvJava technologies like JDBC, Servlet and JSP are known as Web Applications.
- ⇒ These Web Applications are available in 3-tier Architecture.

**Diagram:****2. Enterprise Applications:**

- ⇒ The Applications which are executed in distributed environment and depending on the features like "Security", "Load Balancing" and "Clustering" are known as Enterprise Applications or Enterprise Distributed Applications.
- ⇒ Enterprise Applications are available in n-tier Architecture

Ex: *Java-Frameworks*

*Java-Tools*

-----

**\*imp****JDBC:(Part-1)**

- ⇒ JDBC stands for '**Java DataBase Connectivity**' and which is used to interact with database product.

-----

**FAQ****Define Storage?**

- ⇒ The memory location where the data is available for access is known as Storage.

-----

**Types of Storages:**

- ⇒ According to Java Application development, the storages are categorized into four types:

- 1. Field Storage**
- 2. Object Storage**
- 3. File Storage**
- 4. Database Storage**

**1. Field Storage:**

- ⇒ The memory generated to hold single data value is known as Field Storage.
- ⇒ When we use Primitive datatypes like byte, short, int, long, float, double, char and boolean will generate Field Storages.

**2. Object Storage:**

- ⇒ The memory generated to hold group values is known as Object-Storage.
- ⇒ when we use Non- Primitive datatypes like Class, Interface, Array and Enum will generate Object Storage.

Date: 26/2/2025 (Day-2)

**Example**

```

class Addition
{
    static int a;
    int b;
    void add()
    {
        int c = a+b;
        Sop(c);
    }
}

```

```

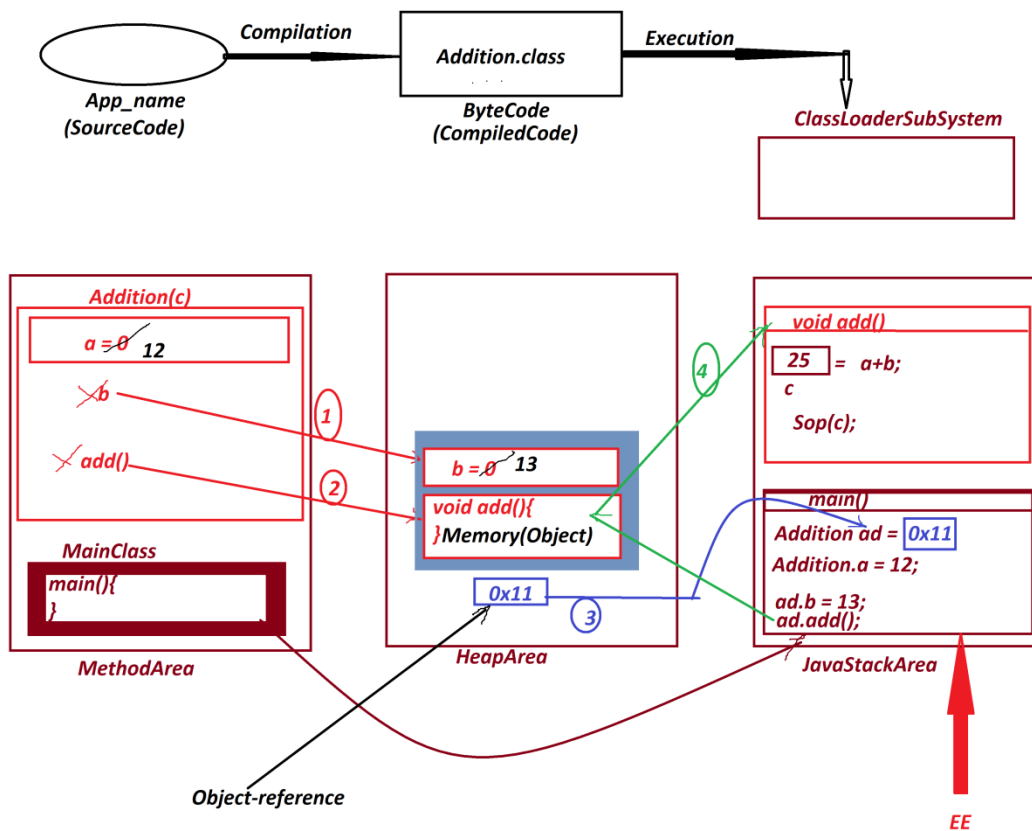
Addition ad = new Addition();
ad.a = 12;
ad.b = 13;
ad.add();

```

```

Addition ob2 = ad;

```

**Diagram**

## **FAQ**

### **What is the difference b/w**

- (i) Object**
- (ii) Object reference**
- (iii) Object reference Variable**

#### **(i) Object:**

⇒ The memory generated to hold instance members of Class is known as Object. It is created using the new keyword.

```
class Book {  
    String title;  
    double price;  
}  
public class ObjectExample {  
    public static void main(String[] args) {  
        Book b1 = new Book(); // Object creation  
    }  
}  
-----  
-> Here, new Book(); creates an object in memory.
```

#### **(ii) Object reference:**

⇒ The address location where the Object is created is known as Object reference.

```
class Book {  
    String title;  
    double price;  
}  
public class ObjectReferenceExample {  
    public static void main(String[] args) {  
        Book b1 = new Book(); // 'b1' holds the reference of the object  
    }  
}
```

```

        System.out.println(b1); // Prints memory address (hashcode) of the
        object
    }
}

```

---

-> Here, b1 is storing the **reference** (memory location) of the object.

### (iii) Object reference Variable:

⇒ The Nonprimitive-data-type variable which is holding Object reference is known as Object reference Variable or Object name.

```

class Book {
    String title;
    double price;
}

public class ObjectReferenceVariableExample {
    public static void main(String[] args) {
        Book b1 = new Book(); // 'b1' is an Object Reference Variable
        Book b2; // Declaring an Object Reference Variable without initializing
        b2 = b1; // Assigning reference of 'b1' to 'b2'

        System.out.println("b1 reference: " + b1);
        System.out.println("b2 reference: " + b2);
    }
}

/** OUTPUT
b1 reference: Book@5e91993f
b2 reference: Book@5e91993f
*/

```

---

-> Here, b1 and b2 are **Object Reference Variables** that hold the same **Object Reference**.

-----

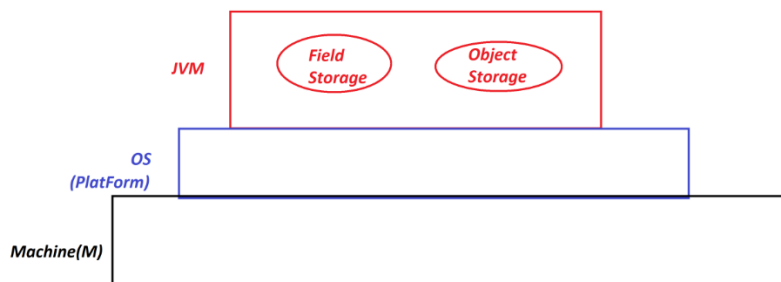
**\*imp****List of Objects generated from CoreJava:**

1. User defined Class Objects
2. String-Objects
3. WrapperClass-Objects
4. Array-Objects
5. Collection<E>-Objects
6. Map<K, V>-Objects
7. Enum<E>-Objects

-----

**Note:**

- ⇒ The Field and Object Storages which are generated part of JVM while Application execution will be destroyed automatically when JVM Shutdowns.
- ⇒ when we want to have permanent storage for Applications, then we have to take the support of any one of the following:
  - File Storage
  - Database Storage



---

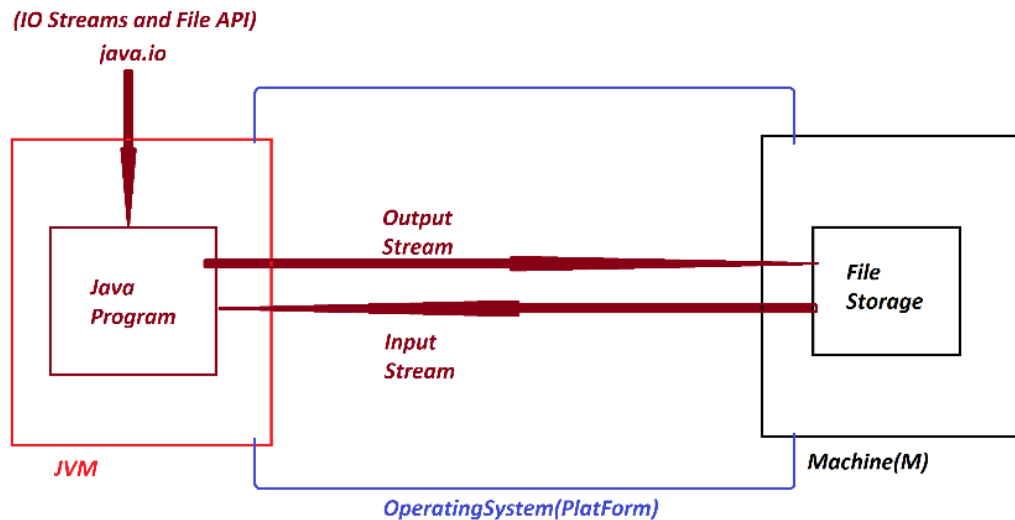
Date: 27/02/2025 (Day-2)

**3. File Storage:**

- ⇒ The smallest permanent storage of Computer System which is 'controlled and managed' by the Operating System is known as File Storage.



- ⇒ In the process of establishing communication b/w Java-Program and File-Storage, the Java-Program must be Constructed using 'Classes and Interfaces' available from 'java.io' package (IO Streams and File API)



### Disadvantages of File Storage:

- (a) Data redundancy
- (b) Data Inconsistency
- (c) Difficulty in accessing data
- (d) Limited data sharing
- (e) File System corruption
- (f) Security Problems

-----

#### (a) Data redundancy:

- ⇒ Same information will be duplicated in different files known as Data redundancy. (data duplication)

-----

#### (b) Data Inconsistency:

- ⇒ data can be inconsistent due to data redundancy

-----

#### (c) Difficulty in accessing data:

- ⇒ Difficulty in accessing data, because the data is available in scattered form and there is no quering process.

-----

*(d) Limited data sharing:*

- ⇒ Limited data sharing because data in scattered form.

-----

*(e) File System corruption:*

- ⇒ File System can be Corrupted due to fragmentation or metadata corruption.

-----

*(f) Security Problems:*

- ⇒ File System will have Security Problems.

-----

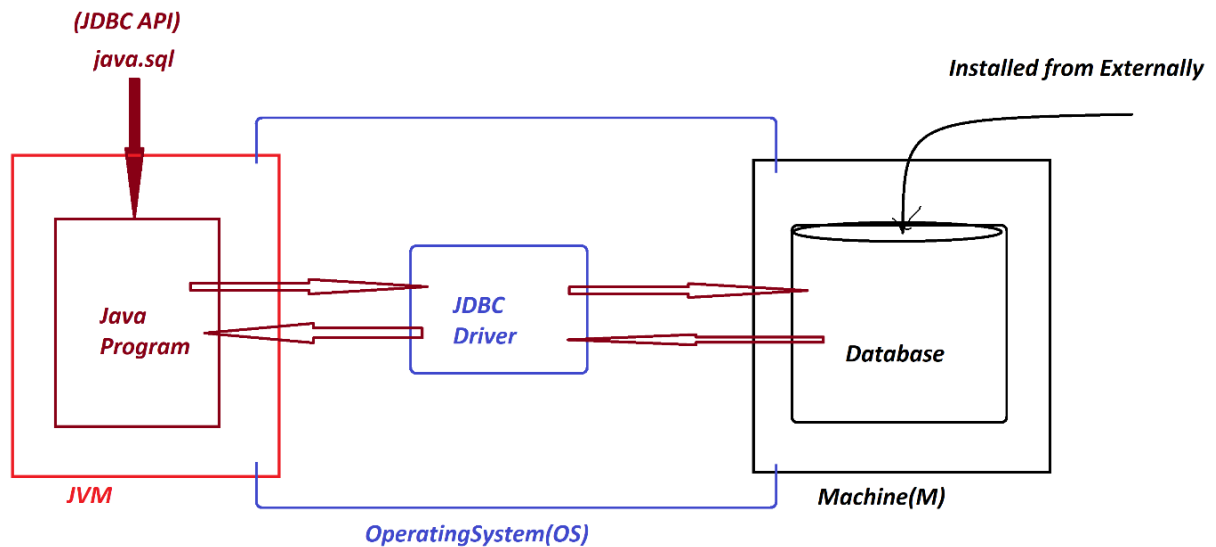
**Note**

- ⇒ Because of disadvantages in File-Storage, the File-Storage cannot be taken as major backend for Java-Applications.
- ⇒ To Overcome all disadvantages of File-Storage, we use Database Storage.

=====

**\*imp****4. Database Storage:**

- ⇒ The largest permanent storage of Computer System, which is installed from externally is known as Database Storage.
- ⇒ In the process of establishing Communication b/w Java-Program and Database-Product, the Java-Program must be constructed using 'Classes and Interfaces' from 'java.sql' package (JDBC API) and the Java-Program must take the support of JDBC-Driver



### FAQ

#### Define 'driver'?

- ⇒ The small s/w program which is used to establish communication b/w two end-points is known as 'driver'.
- ⇒ *Ex: Audio drivers, Video drivers, N/W drivers*

### FAQ

#### Define JDBC driver?

- ⇒ The driver which is used to establish communication b/w Java-Program and DB-Product is known as JDBC driver.

#### Types of drivers:

- ⇒ According Vendor the JDBC drivers are categorized into four types:
  1. *JDBC-ODBC bridge driver (Type-1 driver)*
  2. *Native API driver (Type-2 driver)*
  3. *Network Protocol driver (Type-3 driver)*
  4. *Thin driver (Type-4 driver)*

### Note

- ⇒ According to realtime application development, we use only 'Thin driver'

Date: 28/02/2025 (Day-4)

**\*imp****Creating System Environment ready to execute JDBC Applications:****step-1** : Download and Install Database Product(Oracle)**step-2** : Perform Login process to Database Product

- DB UserName : system
- DB Password : tiger

**step-3** : Create table with name Customer72

(phno, cid, name, city, mid) Primary Key : phno

```
SQL> create table Customer72(
    phno number(15),
    cid varchar2(15),
    name varchar2(15),
    city varchar2(15),
    mid varchar2(25),
    primary key(phno)
);
```

-----

**step-4** : Insert min 5 Customer details from SQL-Command-Line

```
insert into Customer72
values(9898981234, 'HM9898981234', 'Alex', 'Hyd', 'a@gmail.com');
insert into Customer72
values(7676761234, 'HM7676761234', 'Raj', 'Hyd', 'rj@gmail.com');
insert into Customer72
values(8686861234, 'HM8686861234', 'Ram', 'Hyd', 'rm@gmail.com');
```

-----

SQL&gt; Select \* from Customer72;

PHNO	CID	NAME	CITY	MID
9898981234	HM9898981234	Alex	Hyd	a@gmail.com
7676761234	HM7676761234	Raj	Hyd	rj@gmail.com
8686861234	HM8686861234	Ram	Hyd	rm@gmail.com

-----

**step-5 :**

- ⇒ Copy DB-Jar file from "lib" folder of Oracle to User defined folder(on Desktop)

**C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib**

ojdbc6.jar - Oracle11

**FAQ****Define JAR?**

- ⇒ JAR stands for 'Java Archive' and which is compressed format of more number of Class files.

**Note**

- ⇒ This DB-Jar file will provide JDBC drivers.

-----

**step-6 :** Find the PortNo and ServiceName of Database Product(Oracle)  
PortNo and ServiceName is available from 'tnsnames.ora' file of 'Admin' folder of network

**C:\oraclexe\app\oracle\product\11.2.0\server\network\ADMIN**

PortNo : 1521

ServiceName : XE (Express Edition)

-----

***\*imp******Steps used to establish communication to Database product:***

*step-1* : Loader driver

*step-2* : Creating Connection to Database Product

*step-3* : preparing JDBC-statement

*step-4* : Executing the query

*step-5* : Closing the connection from Database

-----

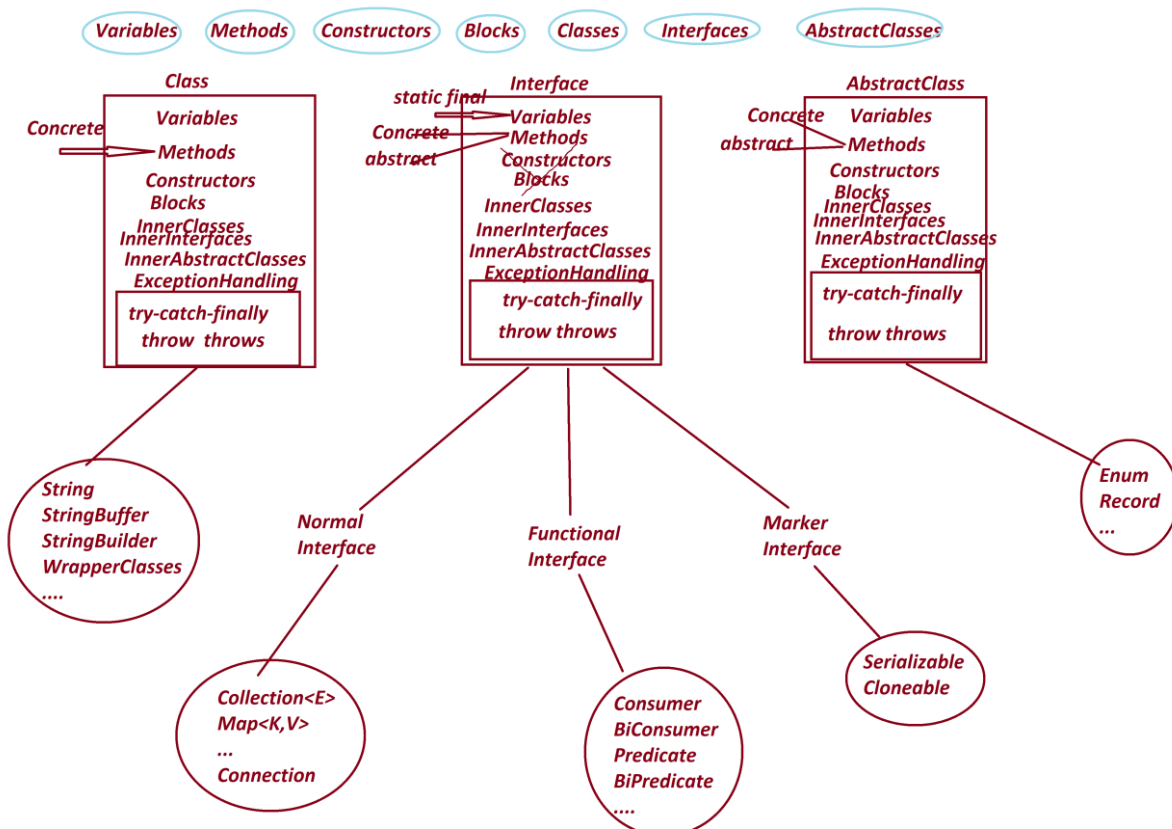
**\*imp****JDBC API:**

- ⇒ 'java.sql' package is known as JDBC-API and which provide 'classes and Interfaces' to Construct JDBC-Applications.

⇒ 'Connection' is a Normal interface from java.sql package and which is root of JDBC API.

⇒ The following are some important methods of 'Connection' interface:

1. createStatement()
2. prepareStatement()
3. prepareCall()
4. getAutoCommit()
5. setAutoCommit()
6. setSavepoint()
7. releaseSavepoint()
8. commit()
9. rollback()
10. close()



**\*imp**

## Application Development design Models:

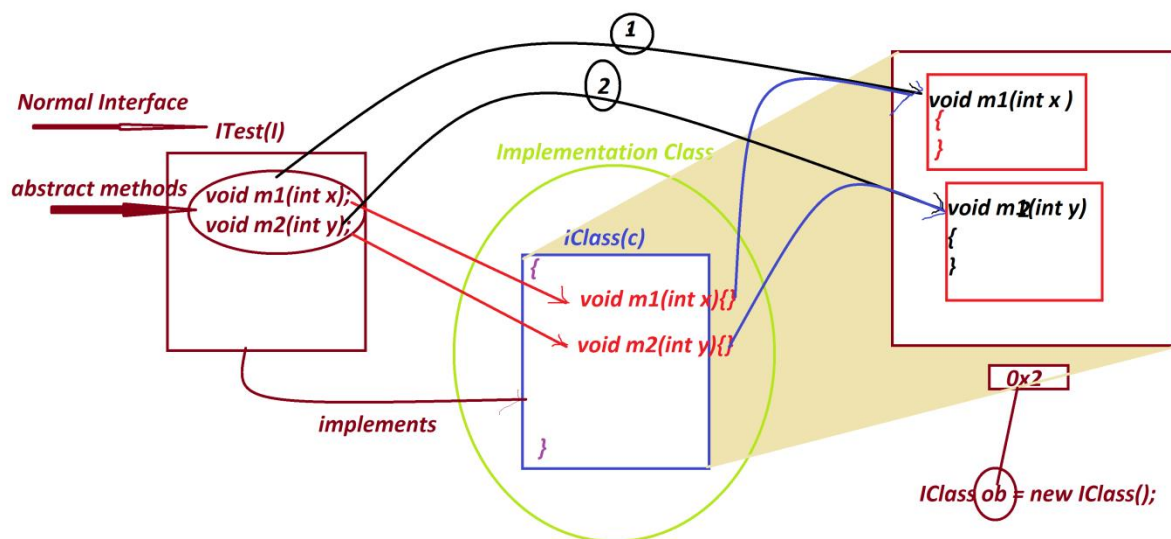
**Model-1 : Interface with Implementation class with name.**

**Model-2 : Interface without Implementation Class\_Name  
(Anonymous InnerClass as Implementation)**

-----

**Model-1 : Interface with Implementation class with name.**

**Diagram:**



**ProjectName : CoreJava\_Model\_1**

```
//P1 : ITest.java
package p1;
public interface ITest
{
    public abstract void m1(int x);
    public abstract void m2(int y);
}

//p1 : IClass.java
package p1;
public class IClass implements ITest
{
    public void m1(int x)
    {
        System.out.println("*****Implemented m1(x)*****");
        System.out.println("The value x:"+x);
    }
}
```

```

    }

    public void m2(int y)
    {
        System.out.println("*****Implemented m2(xy*****");
        System.out.println("The value y:"+y);
    }
}

```

***p2 : DemoModel1.java(MainClass)***

```

package p2;
import p1.*;
public class DemoModel1
{
    public static void main(String[] args)
    {
        IClass ob = new IClass();//Implementation Object
        ob.m1(11);
        ob.m2(23);
    }
}

```

***o/P:***

***\*\*\*\*\*Implemented m1(x)\*\*\*\*\****

***The value x:11***

***\*\*\*\*\*Implemented m2(xy\*\*\*\*\****

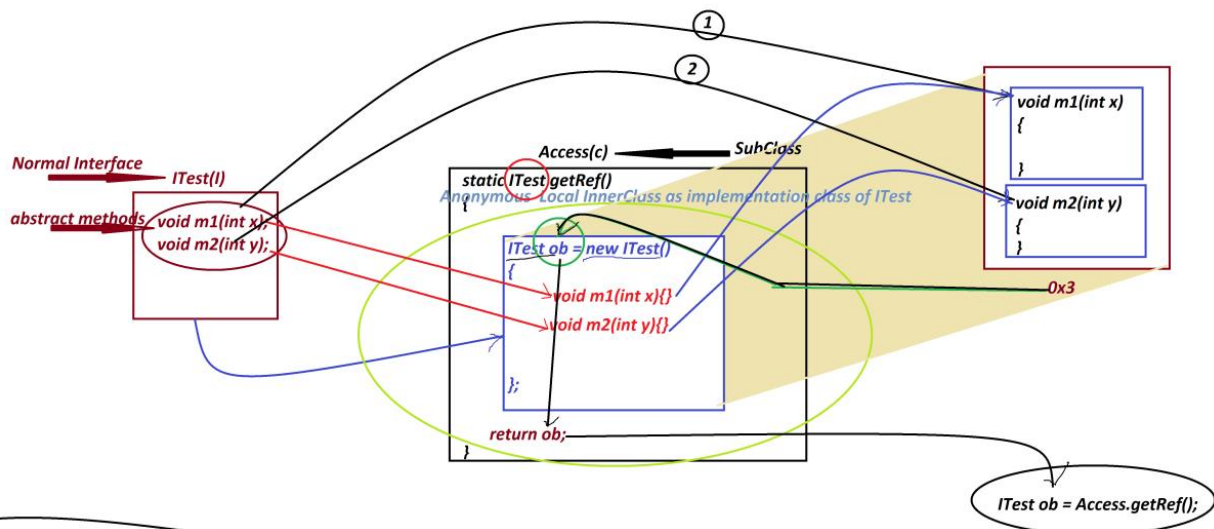
***The value y:23***



**\*imp**

## **Model-2 : Interface without Implementation Class\_Name (Anonymous InnerClass as Implementation)**

### **Diagram**



### **ProjectName : CoreJava\_Model\_2**

**p1 : ITest.java**

```
package p1;
public interface ITest
{
    public abstract void m1(int x);
    public abstract void m2(int y);
}
```

**p1 : Access.java**

```
package p1;
public class Access
{
    public static ITest getRef()
    {
        ITest ob = new ITest()
        {
            public void m1(int x)
            {

```

```

        System.out.println("*****Implemented m1(x)*****");
        System.out.println("The value x:"+x);
    }
    public void m2(int y)
    {
        System.out.println("*****Implemented m2(y)*****");
        System.out.println("The value y:"+y);
    }
};
return ob;
} //OuterClass static method
} //OuterClass

```

*p2 : DemoModel2.java(MainClass)*

```

package p2;
import p1.*;
public class DemoModel2
{
    public static void main(String[] args)
    {
        ITest ob = Access.getRef();//Creating and Accessing Implementation
Object
        ob.m1(11);
        ob.m2(12);
    }
}

```

**o/p:**

**\*\*\*\*\*Implemented m1(x)\*\*\*\*\***

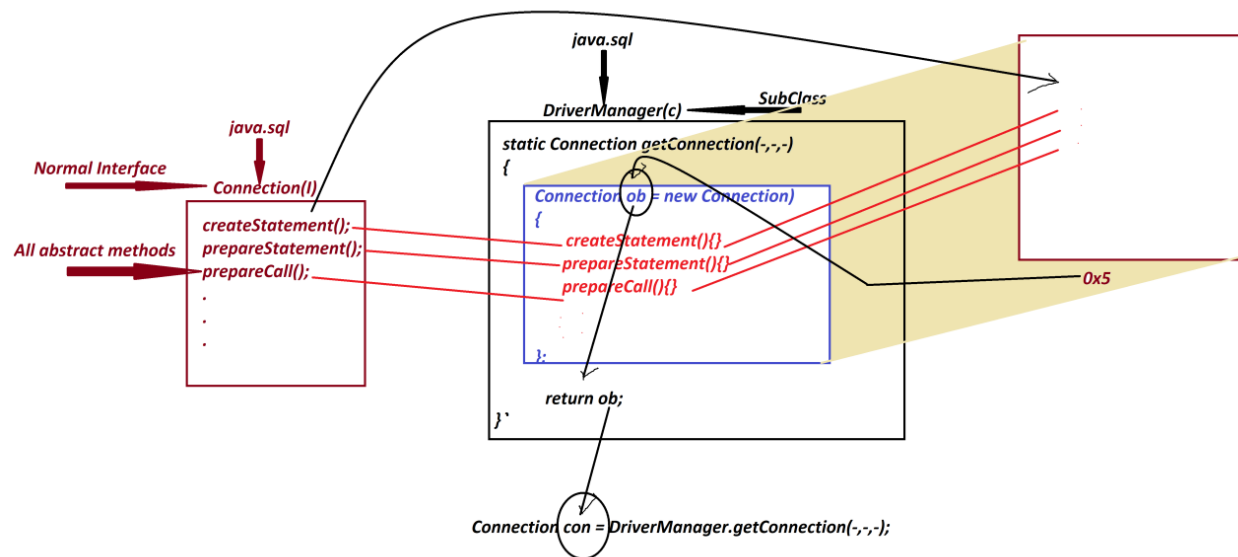
**The value x:11**

**\*\*\*\*\*Implemented m2(y)\*\*\*\*\***

**The value y:12**

## Diagram representing generating 'Connection' implementation

Object:



Date: 04/03/2025 (Day-6)

**Note**

⇒ we use `getConnection()` - method is from 'DriverManager' to create implementation Object for 'Connection' interface, because `getConnection()`- method internally holding 'Anonymous Local InnerClass as implementation class of Connection interface' and which generate Connection-Implementation Object.

-----

**Method Signature of getConnection():**

```
public static java.sql.Connection getConnection
    (java.lang.String, java.lang.String, java.lang.String)
    throws java.sql.SQLException;
```

-----

**Syntax**

```
Connection con = DriverManager.getConnection("DB-URL", "DB-UName", "DB-PWord");
```

<i>DB-URL</i>	=>	<i>jdbc:oracle:thin:@localhost:1521:XE</i>
<i>DB-UName</i>	=>	<i>system</i>
<i>DB-PWord</i>	=>	<i>tiger</i>

```
Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:xe", "system", "tiger");
```

-----

**\*imp****JDBC statements:**

- ⇒ JDBC statements will specify the type of operation to be performed on DB Product.
- ⇒ These JDBC statements are categorized into three types:

- 1. Statement**
- 2. PreparedStatement**
- 3. CallableStatement**

## 1.1 Statement

⇒ 'Statement' is an interface from `java.sql` package and which is used to execute normal queries without IN-Parameters.

(Normal queries means Create, Insert, Select, Update and delete)

⇒ we use `createStatement()`-method from 'Connection' interface to create implementation object for 'Statement' interface, because this `createStatement()` -method internally holding 'Anonymous Local InnerClass as implementation class of Statement-Interface' and which generate Statement-Object.

### Method Signature of `createStatement()`;

```
public abstract java.sql.Statement createStatement() throws  
java.sql.SQLException;
```

### Syntax

```
Statement stm = con.createStatement();
```

⇒ The following are two important methods of 'Statement' interface:

- (a) `executeQuery()`
- (b) `executeUpdate()`

#### (a) `executeQuery()`:

⇒ `executeQuery()`- method is used to execute select-queries Method

#### ⇒ Signature of `executeQuery()`:

```
public abstract java.sql.ResultSet executeQuery(java.lang.String)  
throws java.sql.SQLException;
```

### Syntax

```
ResultSet rs = stm.executeQuery("select-query");
```

(b) `executeUpdate()`:

⇒ `executeUpdate()`-method is used to execute NonSelect-Queries.\

⇒ **Method Signature of `executeUpdate`:**

```
public abstract int executeUpdate(java.lang.String) throws
java.sql.SQLException;
```

⇒ **Syntax**

```
int k = stm.executeUpdate("NonSelect-Query");
```

=====

**\*imp**

### Creating JDBC Application Using IDE Eclipse:

-----

**step-1** : Open IDE Eclipse, while opening name the WorkSpace and click 'Launch'

-----

**step-2** : Create Java Project

-----

**step-3** : Add DB-Jar file to Java-Project through 'Build path' RightClick on Project-> Build Path -> Configure Build Path -> Libraries -> select 'Classpath' and click 'Add External JARs' -> Browse and select DB-Jar file from user defined folder-> Open-> Apply -> Apply and Close.

-----

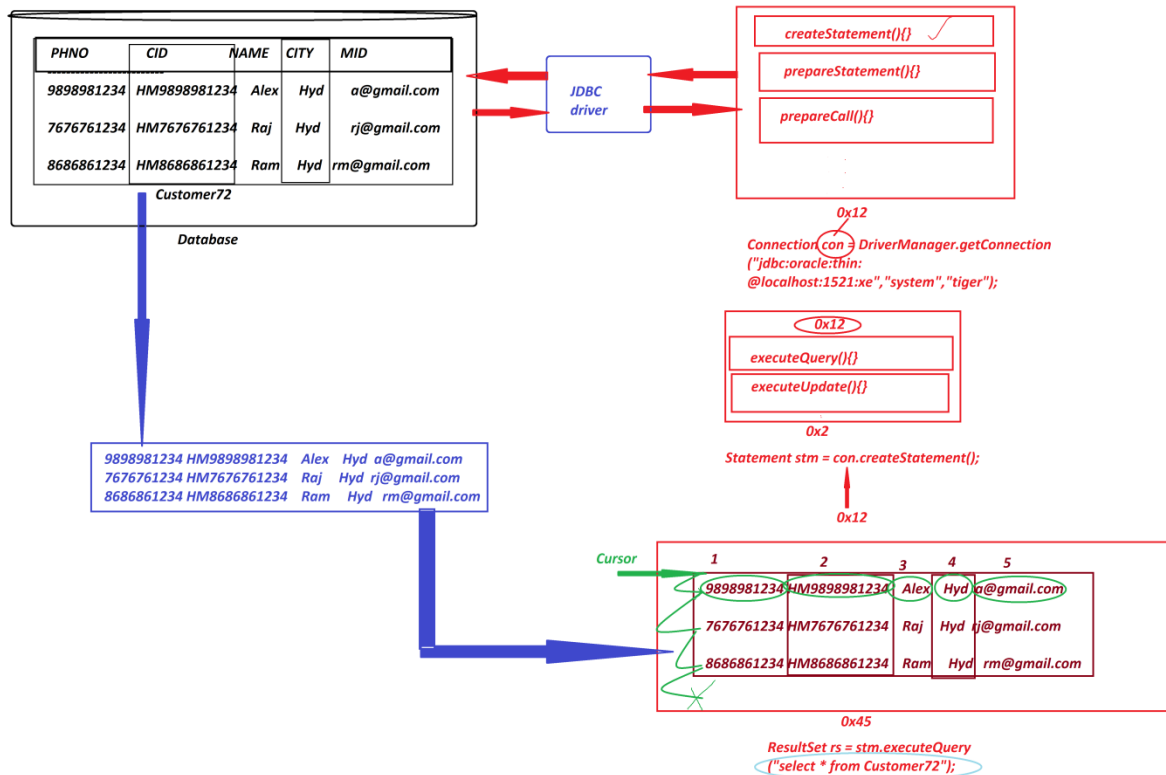
**step-4** : Create package in 'src'

-----

**step-5** : Create class(JDBC Program) in package and write JDBC-code to display all Customer details.

=====

```
DBCon1.java
-----
package test;
import java.sql.*;
public class DBCon1
{
    public static void main(String[] args)
    {
        try
        {
            //step-1 : Loader driver
            Class.forName("oracle.jdbc.driver.OracleDriver");
            //step-2 : Creating Connection to Database Product
            Connection con = DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:xe",
                 "system","tiger");
            //step-3 : preparing JDBC-statement
            Statement stm = con.createStatement();
            //step-4 : Executing the query
            ResultSet rs = stm.executeQuery("select * from Customer72");
            while(rs.next())
            {
                System.out.println(rs.getLong(1)+"\t"
                                   +rs.getString(2)+"\t"+
                                   rs.getString(3)+"\t"+
                                   rs.getString(4)+"\t"+
                                   rs.getString(5));
            }
            //end of loop
            //step-5 : Closing the connection from Database
            con.close();
        } //end of try
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```



Date: 05/03/2025 (Day-7)

**Note**

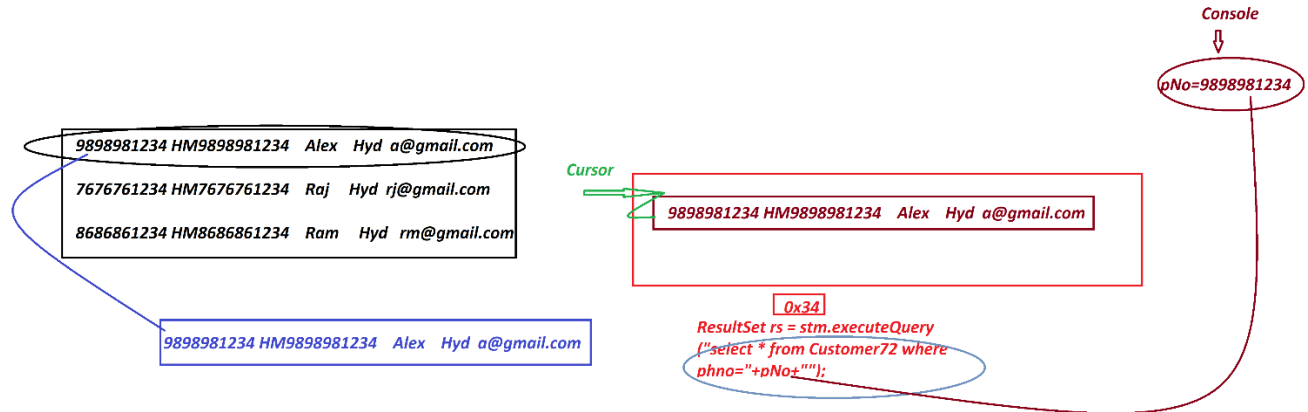
1. **'forName()'** method is from java.lang.Class and which is used to load the class at runtime or execution time, in this process the class is not identified by the Compiler at compilation stage.
2. **executeQuery()** -method will create implementation object for 'ResultSet-Interface' and the object will hold the result generated from select-queries, and the method also generate one cursor pointing before the first row.
3. we use **'next()'** method to move the cursor on ResultSet-object and which generate boolean result.

-----



**Example**

**Q. Construct JDBC Application to display Customer details based on PhoneNo.**

**Diagram****Program- DBCon2.java**

```
package test;
import java.sql.*;
import java.util.*;
public class DBCon2
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        try(s;)
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
            Statement stm = con.createStatement();
            System.out.println("Enter the Cust-PhoneNo to display
details:");
            long pNo = s.nextLong();
            ResultSet rs = stm.executeQuery
                ("select * from Customer72 where phno="+pNo+"");
```

```

        if(rs.next()) {
            System.out.println(rs.getLong(1)+"\t"
                               +rs.getString(2)+"\t"
                               +rs.getString(3)+"\t"
                               +rs.getString(4)+"\t"
                               +rs.getString(5));
        }else {
            System.out.println("Invalud Cust-PhNo....");
        }
        con.close();
    } //end of try
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
}

```

**o/P:**

Enter the Cust-PhoneNo to display details: 9898981234

9898981234 HM9898981234 Alex Hyd a@gmail.com

### **Example**

**Q. Construct JDBC Application to read Customer details from Console and insert into Customer72**

Table (Insert Operation)

**Program : DBCon3.java**

```

package test;
import java.sql.*;
import java.util.*;
public class DBCon3 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try(s){
            Class.forName("oracle.jdbc.driver.OracleDriver");

```

```

        Connection con = DriverManager.getConnection

        ("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
        Statement stm = con.createStatement();
        System.out.println("Enter the Cust-PhoneNO:");
        long phNo = Long.parseLong(s.nextLine());
        String custId = "HM"+phNo;
        System.out.println("Enter the Cust-Name:");
        String name = s.nextLine();
        System.out.println("Enter the Cust-City:");
        String city = s.nextLine();
        System.out.println("Enter the Cust-MailId:");
        String mId = s.nextLine();
        int k = stm.executeUpdate
        ("insert into Customer72
values("+phNo+", '"+custId+"', '"+name+"', '"+city+"', '"+mId+"'");
        if(k>0) {
            System.out.println("Customer details added
Successfully...");
        }
        con.close();
    }catch(Exception e) {
        e.printStackTrace();
    }
}
}

```

**o/p:**

Enter the Cust-PhoneNO: 4545451234

Enter the Cust-Name: RTER

Enter the Cust-City: Hyd

Enter the Cust-MailId: r@gmail.com

Customer details added Successfully...

**Assignment**

DB Table : BookDetails72

(bcode,bname,bauthor,bprice,bqty)

primary key : bcode

program-1 : Construct JDBC Application to insert 5 book details

Program-2 : Construct JDBC Application to display all book details

program-3 : Construct JDBC Application to display book details based on bookCode

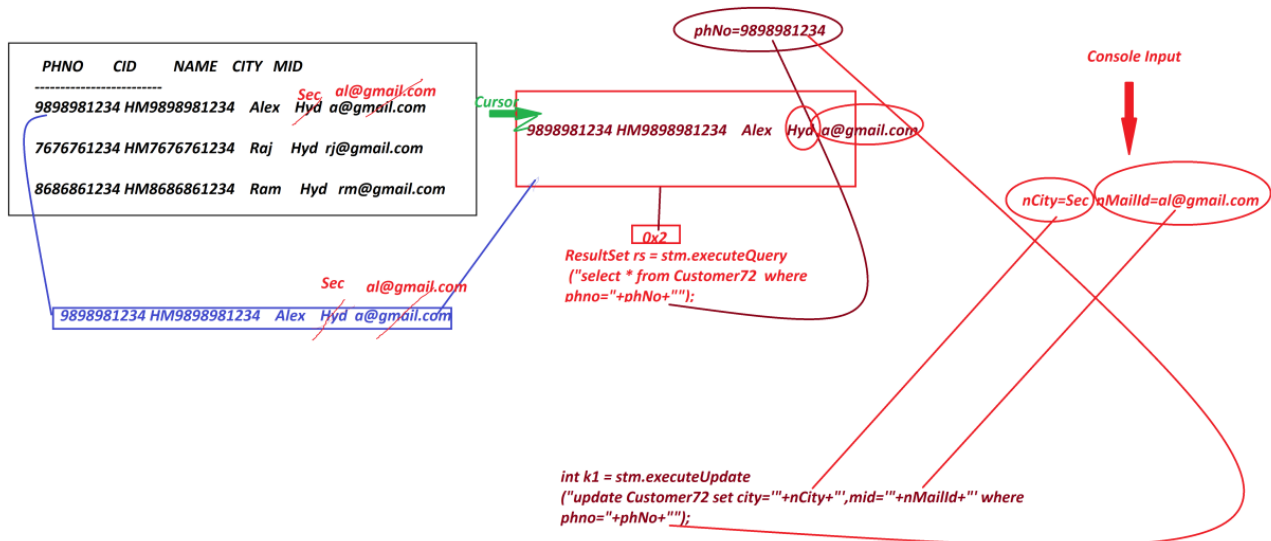
[\[View Program\]](#)

Date: 06/03/2025 (Day-8)

**Example**

### Construct JDBC Application to perform Update and Delete Operations on Customer table.

(based Customer PhoneNo)



[\[View Program\]](#)

**Example****Construct JDBC Application to perform the following operations**

- 1.create
- 2.insert
- 3.update
- 4.delete

**Program : DBCon5.java**

```

package test;
import java.sql.*;
import java.util.*;
public class DBCon5 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try(s){
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
            Statement stm = con.createStatement();
            System.out.println("Enter the
query(create/insert/update/delete)");
            String qry = s.nextLine();
            int k = stm.executeUpdate(qry);
            System.out.println("The value in k:"+k);
            if(k>=0) {
                System.out.println("Query executed Successfully...");
            }
            con.close();
        }catch(SQLException sqe) {
            System.out.println(sqe.getMessage());
            System.out.println("Error Code:"+sqe.getErrorCode());
        }catch(SQLIntegrityConstraintViolationException sie) {
            System.out.println(sie.getMessage());

```

```

        System.out.println("Error Code:"+sie.getErrorCode());
    }catch(Exception e) {
        e.printStackTrace();
    }
}
}

o/p:(Create)
Enter the query(create/insert/update/delete)
create table emp72(id varchar2(10),name varchar2(15),desg varchar2(10),primary key(id))
The value in k:0
Query executed Successfully...

o/p:(Insert)
Enter the query(create/insert/update/delete)
insert into Emp72 values('A11','Alex','SE')
The value in k:1
Query executed Successfully...

o/p:(Update)
Enter the query(create/insert/update/delete)
update Emp72 set desg='ME' where id='A11'
The value in k:1
Query executed Successfully...

o/p:(Delete)
Enter the query(create/insert/update/delete)
delete from Emp72 where id='A21'
The value in k:1
Query executed Successfully...

o/p:
Enter the query(create/insert/update/delete)
create table Emp72(id varchar2(10),name varchar2(15),desg varchar2(10),primary key(id))
Table already available...
Error Code:955

955 - Create
001 - Insert
-----

```

### **Assignment**

*DB Table : Product72(code,name,price,qty)*  
*Primary key : code*

*Construct JDBC Application to perform the following operations based on User Choice:*

- 1.AddProduct*
- 2.ViewAllProducts*
- 3.ViewProductByCode*
- 4.UpdateProductByCode(price-qty)*
- 5.DeleteProductByCode*
- 6.Exit*

*Note:*

*=>repeat the above choice(operations) until we perform exit-operation*

**[[View Program](#)]**

---

**\*imp****1.2 PreparedStatement**

- ⇒ 'PreparedStatement' is an interface from java.sql package and which is used to execute normal queries with IN-Parameters.
- ⇒ we use prepareStatement() -method from 'Connection' interface to create implementation Object for 'PreparedStatement' interface, because the prepareStatement()-method internally holding 'Anonymous Local innerclass' as implementation class of PreparedStatement Interface and which generate PreparedStatement-Object
- ⇒ Method signature of **prepareStatement()**

```
public abstract java.sql.PreparedStatement  
prepareStatement(java.lang.String) throws java.sql.SQLException;
```

**Syntax**

```
PreparedStatement ps = con.prepareStatement("query-structure");
```

- ⇒ The following are two important methods of PreparedStatement:
  - (a) **executeQuery()**
  - (b) **executeUpdate()**

**(a) executeQuery()**

- ⇒ *executeQuery()* - method is used to execute select-queries.
- ⇒ Method Signature of **executeQuery()**:

```
public abstract java.sql.ResultSet executeQuery() throws  
java.sql.SQLException;
```
- ⇒ **syntax**

```
ResultSet rs = ps.executeQuery();
```



**(b) executeUpdate()**

- ⇒ **executeUpdate()** -method is used to execute NonSelect queries.
- ⇒ Method signature of **executeUpdate()**
- ⇒ **public abstract int executeUpdate() throws java.sql.SQLException;**
- ⇒ **syntax**

```
int k = ps.executeUpdate();
```

-----

**Note**

**executeQuery()** and **executeUpdate()** methods are with parameter in 'Statement' and without parameter in 'PreparedStatement'.

-----

**Example :****(Demonstrating PreparedStatement)**

DBTable : BankCustomer72(accno,cid,cname,balance,acctype)

primary key : accno

*create table BankCustomer72(*

*accno number(15),*

*cid varchar2(15),*

*cname varchar2(15),*

*balance number(10,2),*

*acctype varchar2(15),*

*primary key(accno));*

-----

**Construct JDBC Application to perform the following operations based on Choice:**

**1.AddBankCustomer**

**2.ViewAllBankCustomers**

**3.Exit**

**Note: repeat the process until we perform exit operation**

**Program** DBCon6.java

```
package test;
import java.sql.*;
import java.util.*;
public class DBCon6
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        try(s){
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
            PreparedStatement ps1 = con.prepareStatement
                ("insert into BankCustomer72 values(?,?,?,?,?)");
                //Compilation process
            PreparedStatement ps2 = con.prepareStatement
                ("select * from BankCustomer72");//Compilation Process
            while(true) {
                System.out.println("*****Operations Choice*****");
                System.out.println("\t1.AddBankCustomer"
                    + "\n\t2.ViewAllBankCustomers"
                    + "\n\t3.Exit");
                System.out.println("Enter Your Choice:");
                int choice = Integer.parseInt(s.nextLine());
                switch(choice) {
                    case 1:
                        //read data from console into Local variables
                        System.out.println("Enter the CustAccNo:");
                        long accNo = Long.parseLong(s.nextLine());
                        String cId = "SB"+accNo;
                        System.out.println("Enter the CustName:");
                        String cName = s.nextLine();
                        System.out.println("Enter the Cust-Balance:");
                        float balance = Float.parseFloat(s.nextLine());
                        System.out.println("Enter the Cust-AccType:");
                        String accType = s.nextLine();

                        //Load data to PreparedStatement Object using Setter methods
                        ps1.setLong(1, accNo);
                        ps1.setString(2, cId);
                        ps1.setString(3, cName);
```

```

        ps1.setFloat(4, balance);
        ps1.setString(5, accType);

        int k = ps1.executeUpdate();//Execution Process
        if(k>0) {
            System.out.println("BankCustomer Added
Successfully...");
        }

        break;
    case 2:
        ResultSet rs = ps2.executeQuery();//Execution process
        while(rs.next()) {
            System.out.println(rs.getLong(1)+"\t"
                               +rs.getString(2)+"\t"
                               +rs.getString(3)+"\t"
                               +rs.getFloat(4)+"\t"
                               +rs.getString(5));

        }//end of loop
        break;
    case 3:
        System.out.println("Operations Stopped...");
        System.exit(0);
    default:
        System.out.println("Invalid Choice....");
    }//end of switch
} //end of while
}catch(Exception e) {
    e.printStackTrace();
}
}
}

```

*o/p:*

*\*\*\*\*\*Operations Choice\*\*\*\*\**

*1.AddBankCustomer*

*2.ViewAllBankCustomers*

*3.Exit*

*Enter Your Choice:*

*2*

*6123456      SB6123456    Alex    12000.0      Savings*

454541234 SB454541234 Ram 16000.0 Savings

\*\*\*\*\*Operations Choice\*\*\*\*\*

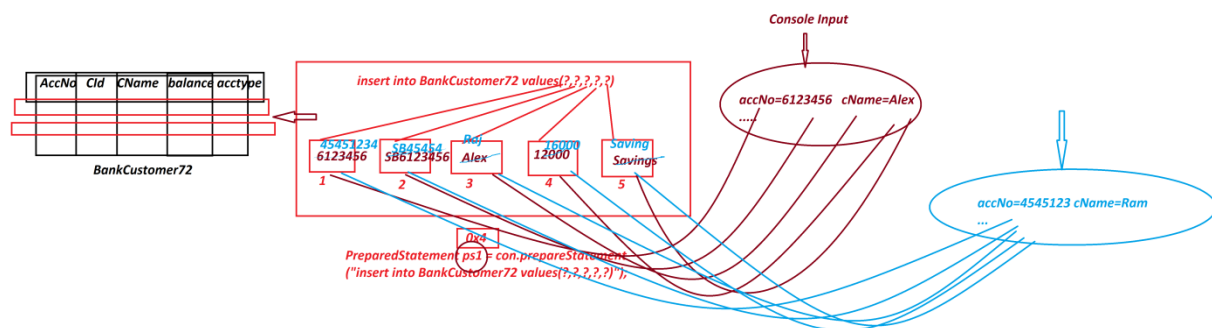
- 1.AddBankCustomer
- 2.ViewAllBankCustomers
- 3.Exit

Enter Your Choice:

3

Operations Stopped...

## **Diagram**



Date: 08/03/2025 (Day-10)

## **Example**

Construct JDBC Application to perform the following operations on Choice based on `AccNo`

- 1.UpdateBankCustomer
- 2.DeleteBankCustomer

## **Program** DBCon7.java

```
package test;
import java.util.*;
import java.sql.*;
public class DBCon7 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try(s){
            Class.forName("oracle.jdbc.driver.OracleDriver");
```

```

        Connection con = DriverManager.getConnection

        ("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
        PreparedStatement ps1 = con.prepareStatement
        ("select * from BankCustomer72 where accno=?");
                                //Compilation process
        PreparedStatement ps2 = con.prepareStatement
        ("update BankCustomer72 set balance=? where
accno=?");
                                //Compilation Process
        PreparedStatement ps3 = con.prepareStatement
        ("delete from BankCustomer72 where accno=?");
                                //Compilation Process

        System.out.println("Enter the Cust-AccNo to perform
Update/Delete operation:");
        long accNo = s.nextLong();
        ps1.setLong(1, accNo);
        ResultSet rs = ps1.executeQuery();
        if(rs.next()) {
            System.out.println("*****Operation Choice*****");
            System.out.println("\t1.UpdateBankCustomer"
                + "\n\t2.DeleteBankCustomer");
            System.out.println("Enter your Choice:");
            int choice = s.nextInt();
            switch(choice) {
                case 1:
                    System.out.println("Existing
balance:"+rs.getFloat(4));
                    System.out.println("Enter the new balance:");
                    float nBal = s.nextFloat();
                    ps2.setFloat(1, nBal);
                    ps2.setLong(2, accNo);
                    int k1 = ps2.executeUpdate();
                    if(k1>0) {

```

```

                                System.out.println("Customer Updated
Successfully...");
                                }
                                break;
                                case 2:
                                    ps3.setLong(1, accNo);
                                    int k2 = ps3.executeUpdate();
                                    if(k2>0) {
                                        System.out.println("Customer deleted
Successfully....");
                                    }
                                    break;
                                default:
                                    System.out.println("Invalid Choice....");
                                }//end of switch
                            }else {
                                System.out.println("Invalid accNo....");
                            }
                            con.close();
                        }catch(Exception e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
        }
    }
}

```

o/p:(Update)

Enter the Cust-AccNo to perform Update/Delete operation:

454541234

\*\*\*\*\*Operation Choice\*\*\*\*\*

1.UpdateBankCustomer

2.DeleteBankCustomer

Enter your Choice:

1

Existing balance:16000.0

Enter the new balance:

20000

Customer Updated Successfully...

o/p:(Delete)

Enter the Cust-AccNo to perform Update/Delete operation:

454541234

\*\*\*\*\*Operation Choice\*\*\*\*\*

1.UpdateBankCustomer

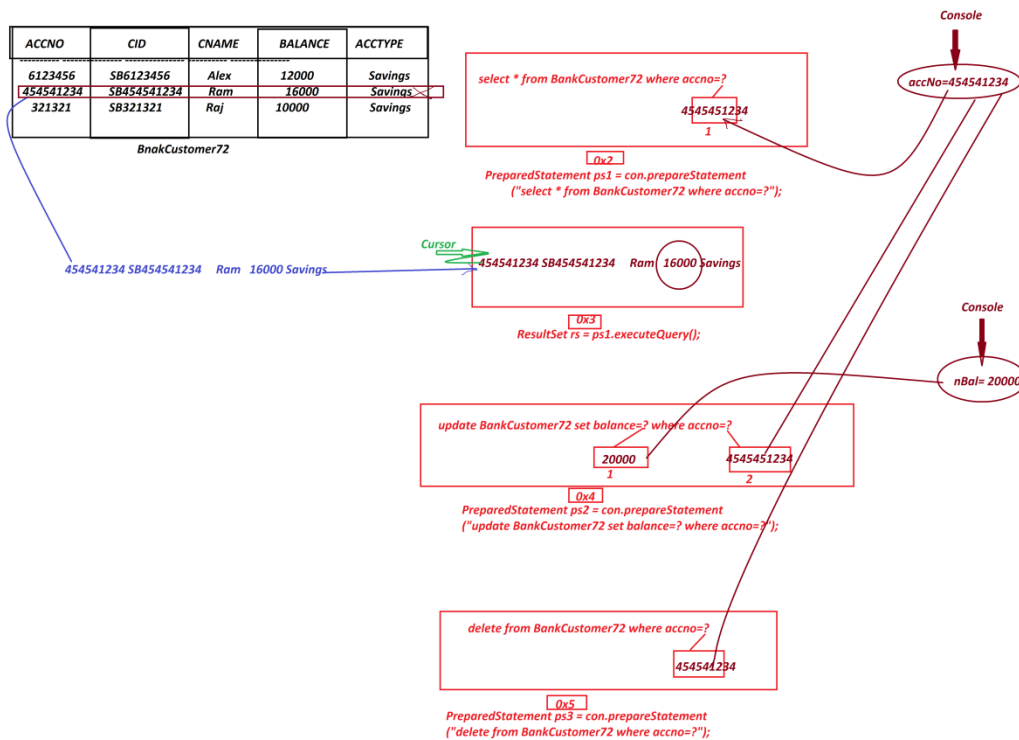
2.DeleteBankCustomer

Enter your Choice:

2

Customer deleted Successfully....

## Diagram



**'ResultSet' in JDBC****\*imp**

- ⇒ 'ResultSet' is an interface from java.sql package and which is instantiated to hold the result generated from select-queries.
- ⇒ 'ResultSet' Objects are categorized into two types:

1. *NonScrollable ResultSet Objects*

2. *Scrollable ResultSet Objects*

**1. NonScrollable ResultSet Objects:**

- ⇒ In NonScrollable ResultSet Objects the cursor can be moved only in one direction, from top-of-table-data to bottom-of-table-data, which means only in forward direction.
- ⇒ we use the following syntax to create NonScrollable ResultSet Object:

**Syntax-1** : Using 'Statement'

```
Statement stm = con.createStatement();
ResultSet rs = stm.executeQuery("select-query");
```

**Syntax-2** : Using 'PreparedStatement'

```
PreparedStatement ps = con.prepareStatement("select-query-structure");
ResultSet rs = ps.executeQuery();
```

**2. Scrollable ResultSet Objects**

- ⇒ In Scrollable ResultSet Objects the cursor can be moved in both directions, which means can be moved in forward and backward directions.
- ⇒ we use the following syntax to create Scrollable ResultSet Object:

**syntax-1** : Using 'Statement'

```
Statement stm = con.createStatement(type, mode);
ResultSet rs = stm.executeQuery("select-query");
```

**syntax-2** : Using 'PreparedStatement'

```
PreparedStatement ps =
con.prepareStatement(type, mode, "select-query-structure");
ResultSet rs = ps.executeQuery();
```



---

*Date: 10/03/2025 (Day-11)***Define "type"?**

- ⇒ "type" specifies the direction of Cursor on ResultSet Object.
- ⇒ The following fields from ResultSet-Interface will specify the "type"

```
public static final int TYPE_FORWARD_ONLY;  
public static final int TYPE_SCROLL_INSENSITIVE;  
public static final int TYPE_SCROLL_SENSITIVE;
```

-----

**Define "mode"?**

- ⇒ "mode" specifies the action to be performed on ResultSet Object.
- ⇒ The following fields from ResultSet-Interface will specify the "mode":

```
public static final int CONCUR_READ_ONLY;  
public static final int CONCUR_UPDATABLE;
```

-----

**Note**

- ⇒ we use the following some important methods to control cursor on ResultSet Object:

- 1. afterLast()**
- 2. beforeFirst()**
- 3. first()**
- 4. last()**
- 5. previous()**
- 6. next()**
- 7. absolute(int)**
- 8. relative(int)**

**1. afterLast()**

- ⇒ afterLast() - method will make the cursor point after the last row in ResultSet Object.

**2. beforeFirst()**

⇒ `beforeFirst()` - method will make the cursor point before the first row in `ResultSet` Object.

**3. first()**

⇒ `first()` - method will make the cursor point to the first row of `ResultSet` Object.

**4. last()**

⇒ `last()` - method will make the cursor point to the last row of `ResultSet` Object.

**5. previous()**

⇒ `previous()` - method is used to move the cursor in backward direction.

**6. next()**

⇒ `next()` - method is used to move the cursor in forward direction.

**7. absolute(int)**

⇒ `absolute(int)` - method is used to move the cursor to specified row number.

**8. relative(int)**

⇒ `relative(int)` - method is used to take incre/decre value as parameter and move the cursor in forward or backward direction from current cursor position.

**Example**

**Program:** DBCon8.java

```
package test;

import java.sql.*;

public class DBCon8 {

    public static void main(String[] args) {

        try {

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

            Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");

            System.out.println("*****Statement*****");

            Statement stm = con.createStatement

                (ResultSet.TYPE_SCROLL_INSENSITIVE,

                 ResultSet.CONCUR_READ_ONLY);

            ResultSet rs1 = stm.executeQuery("select * from Customer72");

            System.out.println("-----3rd row-----");

            rs1.absolute(3);

            System.out.println(rs1.getLong(1)+"\t"

                               +rs1.getString(2)+"\t"

                               +rs1.getString(3)+"\t"

                               +rs1.getString(4)+"\t"

                               +rs1.getString(5));

            System.out.println("-----relative(-2)-----");

            rs1.relative(-2);

            System.out.println(rs1.getLong(1)+"\t"

                               +rs1.getString(2)+"\t"

                               +rs1.getString(3)+"\t"

                               +rs1.getString(4)+"\t"

                               +rs1.getString(5));

            System.out.println("-----last row-----");

            rs1.last();

            System.out.println(rs1.getLong(1)+"\t"

                               +rs1.getString(2)+"\t"

                               +rs1.getString(3)+"\t"
```

```

        +rs1.getString(4)+"\t"
        +rs1.getString(5));
System.out.println("-----first row-----");
rs1.first();
System.out.println(rs1.getLong(1)+"\t"
        +rs1.getString(2)+"\t"
        +rs1.getString(3)+"\t"
        +rs1.getString(4)+"\t"
        +rs1.getString(5));
System.out.println("*****PreparedStatement*****");
PreparedStatement ps = con.prepareStatement(
        "select * from BankCustomer72",
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
ResultSet rs2 = ps.executeQuery();
System.out.println("-----reverse-----");
rs2.afterLast();
while(rs2.previous()) {
        System.out.println(rs2.getLong(1)+"\t"
                +rs2.getString(2)+"\t"
                +rs2.getString(3)+"\t"
                +rs2.getString(4)+"\t"
                +rs2.getString(5));

        }//end of loop
}catch(Exception e) {
        e.printStackTrace();
    }
}
}

```

o/p:

\*\*\*\*\*Statement\*\*\*\*\*

-----3rd row-----

4545451234 HM4545451234 RTER Hyd r@gmail.com

-----relative(-2)-----

9898981234 HM9898981234 Alex Sec al@gmail.com

```

-----last row-----
4545451234 HM4545451234      RTER Hyd  r@gmail.com
-----first row-----
9898981234 HM9898981234      Alex Sec  al@gmail.com
*****PreparedStatement*****
-----reverse-----
321321      SB321321      Raj  10000 Savings
6123456      SB6123456      Alex 12000 Savings

```

### **Assignment**

*DB Table: Employee72(eid,ename,edesg,bsal,hra,da,totsal)*

*Primary Key : eid*

*Construct JDBC Application to perform the following Operations based on Choice:*

- 1.AddEmployee*
- 2.ViewAllEmployees*
- 3.ViewEmployeeByCode*
- 4.UpdateEmployeeById(bSal)*
- 5.DeleteEmployeeById*
- 6.Exit*

*Calcutions:*

*hra = 91% of bSal*

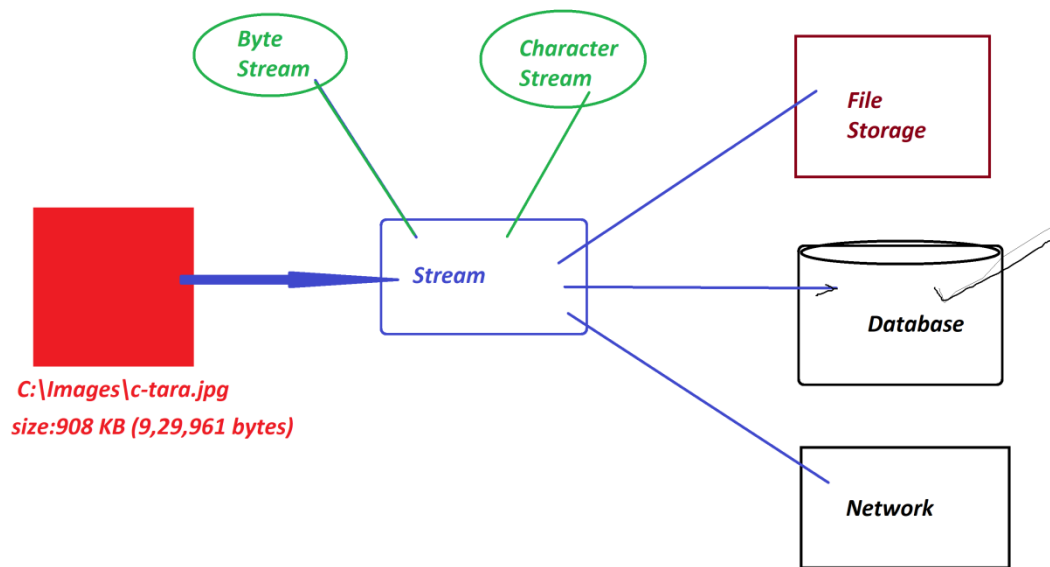
*da = 63% of bSal*

*totSal = bSal+hra+da*

*Exception: Min bSal must be 12000/-,else raise the exception*

**pending**

---



### **\*imp**

#### **Streams with Database product**

#### **Define stream? (Normal definition)**

⇒ The continuous flow of data is known as stream.

#### **Types of streams**

⇒ Java language support two types of streams:

##### **1. Byte Stream (Binary Stream)**

##### **2. Character Stream**

#### **1. Byte Stream(Binary Stream)**

- ⇒ The continuous flow of data in the form of 8-bits is known as Byte Stream or Binary Stream.
- ⇒ Through Byte Stream we can send all Multi-Media data formats, which means Text, Audio, Video, Image and Animation.

#### **2. Character Stream**

- ⇒ The Continuous flow of data in the form of 16-bits is known as Character Stream or Text Stream.

- ⇒ Character Stream is preferable for Text data, and which is not preferable for Audio, Video, Image and Animation data.

-----

- ⇒ We use the following SQL-Types to store Stream data:

**(a)BLOB**

**(b)CLOB**

### **(a)BLOB**

- ⇒ BLOB stands for 'Binary Large Objects' and which is used to store Byte Stream data.

### **(b)CLOB**

- ⇒ CLOB stands for 'Character Large Objects' and which is used to store Character Stream data.

-----

### **\*imp**

### **Storing Stream data to Database product**

#### **Step-1 :**

Create DB Table with name StreamTab72(id, name, mfile)

```
create table StreamTab72(id varchar2(10), name varchar2(15), mfile BLOB,  
primary key(id));
```

#### **Step-2 : Construct JDBC Application to store Image to Database product**

**Program : DBCon9.java**

```
package test;  
import java.util.*;  
import java.io.*;  
import java.sql.*;  
public class DBCon9  
{  
    public static void main(String[] args)
```

```

{
    Scanner s = new Scanner(System.in);
    try(s){
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
        PreparedStatement ps = con.prepareStatement
            ("insert into StreamTab72 values(?,?,?)");
        System.out.println("Enter the User-Id:");
        String id = s.nextLine();
        System.out.println("Enter the User-Name:");
        String name = s.nextLine();
        System.out.println("Enter the Location(fPath&fName) of User-
Image(Source)");
        String path = s.nextLine();
        File f = new File(path);
        if(f.exists()) {
            FileInputStream fis = new FileInputStream(path);
            ps.setString(1,id);
            ps.setString(2,name);
            ps.setBinaryStream(3, fis, f.length());
            int k = ps.executeUpdate();
            if(k>0) {
                System.out.println("Image Stored Successfully...");
            }
            fis.close();
        }else {
            System.out.println("Invalid fPath or fName....");
        }
        con.close();
    }catch(Exception e) {
        e.printStackTrace();
    }
}

```



}

**o/p:**

Enter the User-Id:

A121

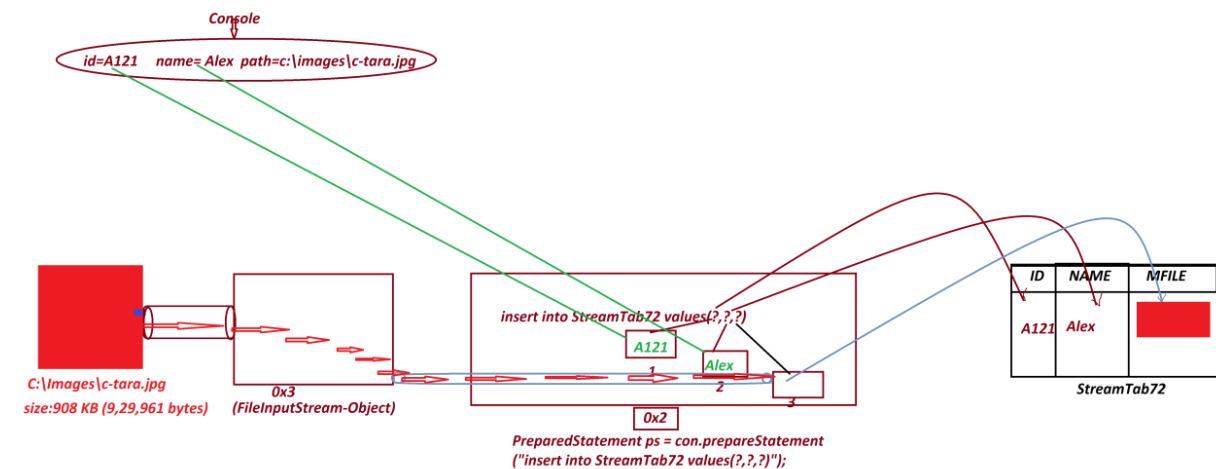
Enter the User-Name:

Alex

Enter the Location(fPath&amp;fName) of User-Image(Source)

C:\Images\c-tara.jpg

Image Stored Successfully...

**Diagram**

Date: 12/03/2025 (Day-13)

**Retrieving Stream(Image) from Database Product**

```
Program : DBCon10.java
package test;
import java.util.*;
import java.io.*;
import java.sql.*;
public class DBCon10 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try(s){
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
            PreparedStatement ps = con.prepareStatement
                ("select * from StreamTab72 where id=?");
            System.out.println("Enter the User-Id to retrieve details:");
            String id = s.nextLine();
            ps.setString(1, id);
            ResultSet rs = ps.executeQuery();
            if(rs.next()) {
                Blob b = rs.getBlob(3);
                byte by[] = b.getBytes(1, (int)b.length());
                System.out.println("User-Id:"+rs.getString(1));
                System.out.println("User-Name:"+rs.getString(2));
                System.out.println("Enter the location(fPath&fName-
destination) to store Stream: ");
                String path = s.nextLine();
                FileOutputStream fos = new FileOutputStream(path);
                fos.write(by);
                System.out.println("Stream stored to specified
location....");
                fos.close();
            }else {
```

```

        System.out.println("Invalid User Id....");
    }
    }catch(Exception e) {
        e.printStackTrace();
    }
}
}

```

o/p:

Enter the User-Id to retrieve details:

A121

User-Id:A121

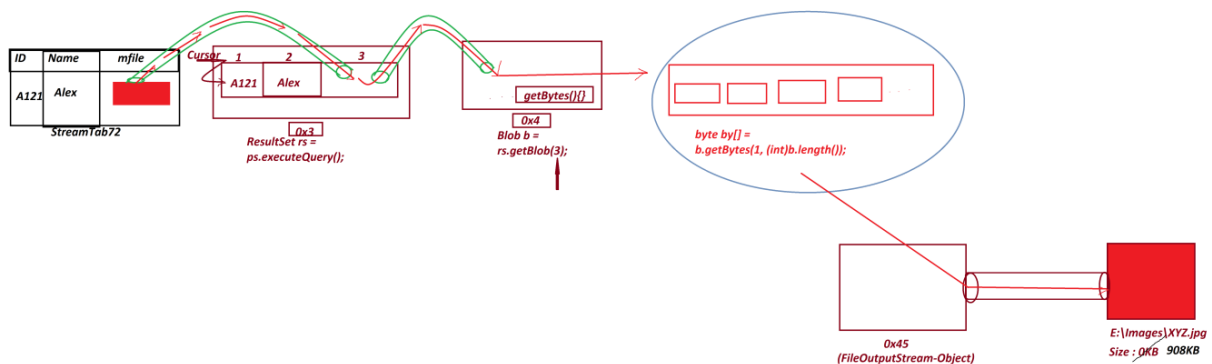
User-Name:Alex

Enter the location(fPath&fName-destination) to store Stream:

E:\Images\XYZ.jpg

Stream stored to specified location....

### Diagram



**FAQ****Define 'FileInputStream'?**

⇒ 'FileInputStream' is a class from java.io package and which is instantiated to link(open) the file to read Byte-Stream data.

⇒ **Syntax**

```
FileInputStream fis = new FileInputStream("fPath&fName");
```

-----

**FAQ****Define 'FileOutputStream'?**

⇒ 'FileOutputStream' is a class from java.io package and which is instantiated to create a new file with 0KB and links(opens) the file to write Byte-stream data.

⇒ **Syntax**

```
FileOutputStream fos = new FileOutputStream("fPath&fName");
```

-----

**FAQ****Define setBinaryStream() method?**

⇒ setBinaryStream()-method belongs to PreparedStatement and which links stream to parameter-index-field of PreparedStatement Object.

⇒ Through this method we must specify para-index-no,location-of-stream and length-of-stream.

⇒ **Syntax**

```
ps.setBinaryStream(3, fis, f.length());
```

-----

**FAQ****define getBlob() method?**

⇒ getBlob()-method is from ResultSet and which is used to instantiate Blob-Interface and this Blob-Object internally linked to Stream-column of ResultSet.

⇒ **Syntax**

```
Blob b = rs.getBlob(3);
```

-----

### **FAQ \*imp**

#### **Define getBytes() method?**

⇒ getBytes()-method is from Blob, and which is used to convert stream into byte-Array.

⇒ **Syntax**

```
byte by[] = b.getBytes(1, (int)b.length());
```

-----

### **\*imp**

#### **CallableStatement**

⇒ 'CallableStatement' is an interface from java.sql package and which is used to execute Procedures and Functions on Database product.

⇒ we use prepareCall()-method from 'Connection-interface' to create implementation Object for 'CallableStatement Interface', because the prepareCall()-method internally holding 'Anonymous Local InnerClass' as implementation class of 'CallableStatement' and generate CallableStatement Object.

⇒ Method Signature of **prepareCall()**:

```
public abstract java.sql.CallableStatement  
prepareCall(java.lang.String) throws java.sql.SQLException;
```

⇒ **Syntax**

```
CallableStatement cs = con.prepareCall("{call Proce_name/Func_name}");
```

-----

## What is the difference

### (i)function

### (ii)member function

### (iii)method

#### (i) function

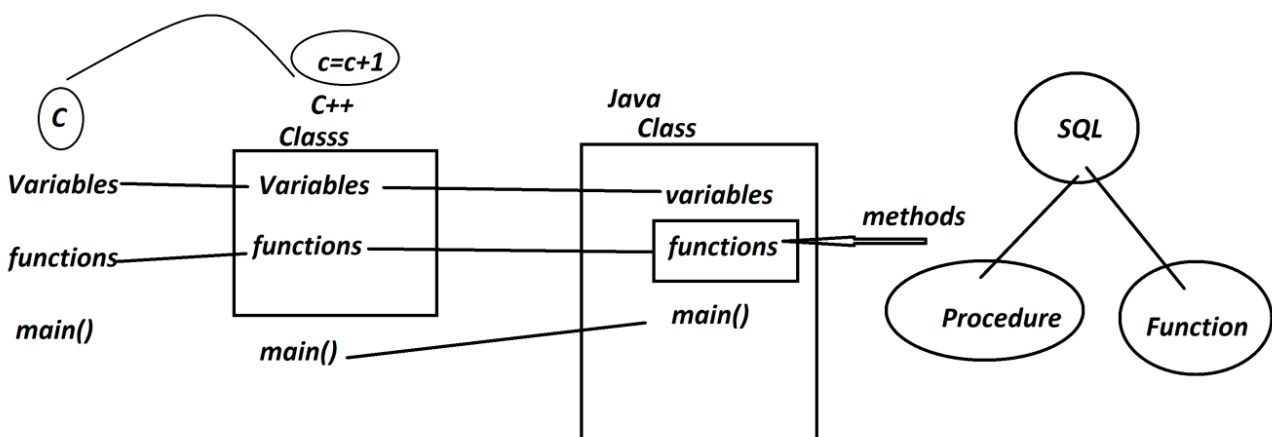
- ⇒ The part of program which is executed outof main()-program is known as 'function' in C-Lang.

#### (ii) member function

- ⇒ The functions which are declared as members of class in C++ are known as member functions.
- ⇒ These member functions can be declared inside the class or Outside the class with class reference.

#### (iii) method

- ⇒ The functions which are declared only inside the class in Java are known as Methods.



## **FAQ**

### **Define 'Procedure'?**

⇒ Procedure is a set-of-queries executed on Database product at-a-time, and after execution Procedure will not return any value.

(Procedure means NonReturn\_type)

*Structure of Procedure:*

```
create or replace procedure Procedure_name
(para_list) is
begin
    query-1
    query-2
    ...
end;
/
```

### **Types of Procedures**

⇒ According to JDBC, Procedures are categorized into three types:

***(a) IN-Parameter Procedure***

***(b) OUT-Parameter Procedure***

***(c) IN-OUT-Parameter Procedure***

#### **(a) IN-Parameter Procedure**

⇒ The Procedures which take the data from Java-Program and sent to database product are known as IN-Parameter Procedures.

#### **(b) OUT-Parameter Procedure**

⇒ The Procedures which take the data from database product and sent to Java-Program are known as OUT-Parameter Procedures.

#### **(c) IN-OUT-Parameter Procedure**

- ⇒ The procedures which perform both operations are known as IN-OUT-Parameter Procedures.

## **FAQ**

### **Define 'Function'?**

- ⇒ Function is a set-of-queries executed on Database product at-a-time, and after execution it will return the value.
- ⇒ we use 'return' statement to return the value from Function.

### *Structure of Function*

```
create or replace Function Function_name
(para_list) return data_type as var data_type;
begin
    queries
    return var;
end;
/
```

-----

## **\*imp**

### **Construct Application to demonstrate Procedure**

#### **Step-1 : Create the following tables**

```
EmpData72(eid,ename,edesg);
EmpAddress72(eid,hno,sname,city,state,pincode);
EmpContact72(eid,mid,phno);
EmpSalary72(eid,bsal,hra,da,totsal);
```

-----

```
create table EmpData72(eid varchar2(10),ename varchar2(15),edesg
varchar2(10), primary key(eid));
```

-----

```
create table EmpAddress72(eid varchar2(10), hno varchar2(15), sname
varchar2(15), city varchar2(15), state varchar2(15), pincode
number(10), primary key(eid));
```

-----



```
create table EmpContact72(eid varchar2(10), mid varchar2(25), phno
number(15), primary key(eid));
```

-----

```
create table EmpSalary72(eid varchar2(10), bsal number(10), hra
number(10,2), da number(10,2), totsai number(10,2), primary key(eid));
```

-----

**Step-2 :** *Construct Procedure to perform Insert operation to Employee tables*

```
create or replace procedure InsertEmployee72
(id varchar2, en varchar2, ed varchar2, hn varchar2, sn varchar2, cty
varchar2, st varchar2, pcode number, md varchar2, pno number, bs
number, h number, d number, ts number) is
begin
    insert into EmpData72 values(id, en, ed);
    insert into EmpAddress72 values(id, hn, sn, cty, st, pcode);
    insert into EmpContact72 values(id, md, pno);
    insert into EmpSalary72 values(id, bs, h, d, ts);
end;
/
```

-----

**Step-3 :** *Construct JDBC Application to execute Procedure*

**Program: DBCon11.java**

```
package test;
import java.util.*;
import java.sql.*;
public class DBCon11 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try(s){
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:xe","system","tiger");
            CallableStatement cs = con.prepareCall
```

```
("{call InsertEmployee72(?,?,?,?,?,?,?,?,?,?,?,?,?,?)}");
System.out.println("Enter the Emp-Id:");
String eId = s.nextLine();
System.out.println("Enter the Emp-Name:");
String eName = s.nextLine();
System.out.println("Enter the Emp-Desg:");
String eDesg = s.nextLine();
System.out.println("Enter the Emp-HNo:");
String hNo = s.nextLine();
System.out.println("Enter the Emp-SName:");
String sName = s.nextLine();
System.out.println("Enter the Emp-City:");
String city = s.nextLine();
System.out.println("Enter the Emp-State:");
String state = s.nextLine();
System.out.println("Enter the Emp-PinCode:");
int pinCode = Integer.parseInt(s.nextLine());
System.out.println("Enter the Emp-MailId:");
String mId = s.nextLine();
System.out.println("Enter the Emp-PhNo:");
long phNo = s.nextLong();
System.out.println("Enter the Emp-bSal:");
int bSal = s.nextInt();
float hra = 0.93F*bSal;
float da = 0.61F*bSal;
float totSal = bSal+hra+da;
cs.setString(1, eId);
cs.setString(2, eName);
cs.setString(3, eDesg);
cs.setString(4, hNo);
cs.setString(5, sName);
cs.setString(6, city);
cs.setString(7, state);
cs.setInt(8, pinCode);
cs.setString(9, mId);
```

```
        cs.setLong(10, phNo);
        cs.setInt(11, bSal);
        cs.setFloat(12, hra);
        cs.setFloat(13, da);
        cs.setFloat(14, totSal);
        cs.execute();//Execute Procedure
        System.out.println("Employee added Successssfully....");
        con.close();
    }catch(Exception e) {
        e.printStackTrace();
    }
}
```

---

**o/p**

Enter the Emp-Id:

T121

Enter the Emp-Name:

Alex

Enter the Emp-Desg:

TE

Enter the Emp-HNo:

12-34/h

Enter the Emp-SName:

SRN

Enter the Emp-City:

Hyd

Enter the Emp-State:

TG

Enter the Emp-PinCode:

506112

Enter the Emp-MailId:

alex@gmail.com

Enter the Emp-PhNo:

9898981234

Enter the Emp-bSal:

45000

Employee added Successssfully....

---

*Date: 13/03/2025 (Day-14)*