Unit-2 : Introducing Classes, Objects and methods

2.1 Class fundamentals

A class is a blueprint for creating an objects. Class encapsulates data and methods to manipulate the data. Class is a combination of data members and member functions.

Syntax :

```
Class ClassName
{
  Data_members;
  member_functions()
  { function_body;
  }
}
```

Note:- Methods and Member functions are same term.

Example :

```
Class person
{
  String Name;
  int age;
  void display_details()
  {
    System.out.println("Name of the person is: "
                       + Name);
    System.out.println("Age of the person is:" +Age);
  }
}public class Rajesh
  public static void main(String args[])
```

```
{
    Person p = new person();
    P.Name = "Rajesh parajuli";
    P.Age = 27;
    P.display_details();
}
}
```

To compile: javac Rajesh.java

To Run: java Rajesh

output: Name of the person is: Rajesh parajuli
Age of the person is: 27

Example 2: To Represent Class in Java program

Java program to find given number is even or odd.

```
import java.util.Scanner;
public class Evenodd
{
    public static void main(String args[])
    {
        Scanner scan = new Scanner(System.in)
        System.out.println("Enter the number:");
        int num = scan.nextInt();
        findEvenodd(num); // Calling function
    }
    public static void findEvenodd(int num);
    {
        if (num%.2 == 0)
            System.out.println("even number");
        else
            System.out.println("odd number");
    }
}
```

2.2. Object Creation

An instance of a class is called an object.
Objects are created using the new keyword.

Syntax:

```
Class_Name object_Name = new Class_Name();
```

Example:

```
Class Animal
{
  String name;
  void display()
  {
    System.out.println("Animal name is: "+name);
  }
}
public Class Demo
{
  public static void main(String[] args)
  {
    //object creation
    Animal dog = new Animal();
    dog.name = "Dog";
    dog.display();
  }
}
```

To compile : javac Demo.java
To Run : java Demo

output: Animal name is : Dog

2.3 methods

Methods define the behavior of a class. methods are member functions that have blocks of code to perform a task.

Syntax:

```
return_type method_Name (parameters)
{
   // block of codes or method body;
}
```

Example:

```
class Calculator
{
  int add (int a, int b)
  {
    return a+b;
  }
}

public class Demo
{
  public static void main (String [] args)
  {
    Calculator calc = new Calculator ();
    int sum = calc.add (5,10);
    System.out.println ("sum is: " +sum);
  }
}
```

output: Sum is: 15

2.4 Command Line Arguments

The java Command line argument is an argument that is passed to the main method at the time of running (Execution) of the java program

Syntax:

```
public static void main(String args[])
{
  // args stores Command-line arguments
}
```

Example:

```
class CommandLine
{
  public static void main(String args[])
  {
    System.out.println("your first argument is:"
                       + args[0]);
  }
}
```

To compile: javac CommandLine.java
To Run : java CommandLine Rajesh

output: your first argument is: Rajesh

Example2: "program that prints all the values"

```
class A
{
  public static void main(String args[])
  {
    for(int i=0; i<args.length; i++)
      System.out.println(args[i]);
  }
}
```

To compile: javac A.java
To Run : java A Rajesh prakash suman kabi
output: Rajesh
prakash
suman
kabi

2.5 Constructors

* A Constructor is a special type of member function that initializes an object when it is created.
* Constructors are automatically called when an instance of class is created. The constructor has the same name as the class name and does not have return type even void.

Syntax:

```
class class_Name
{
    class_Name ( ) // Constructor
    {
        // Constructor body
    }
}
```

Types of Constructor in java

(i) Default Constructor
(ii) Parameterized Constructor

(i) Default Constructor :- The constructor having no arguments and provides the default values to the objects like 0, null etc. depending on the data type of the member variable of the class is called Default Constructor. If there is no any constructor in the class java automatically create the default constructor.

Syntax:

```
class class_Name
{   class_Name ( )
    {
    }
}
```

Program Example

```
Class A
{ int a;
  String name;
  A()
  {
    a=0;
    name=null;
  }
  void show()
  {
    System.out.println("the value of a and name
                        is" +a+" "+ name);
  }
}
Class Demo
{
  public static void main(String args[])
  {
    A obj = new A(); //Automatically execute
                     Constructor
    obj.show(); // showing function execute to
                show the output.
  }
```

Output:

the value of a and name is 0 null

(2) parameterized Constructor:- The Constructor having parameters or arguments is known as parameterized Constructor. this keyword is used to represent the current members of the class.

Syntax:

```
class class_name{
class_name(int A, int B)
{ ----
}
```

Example:

```
Class Student
{
  String name;
  int age;
  Student (String n, int a)
  {
    this.name = n;
    this.age = a;
  }
  void display ( )
  {
    System.out.println("Name is: "+name);
    System.out.println("Age is: "+age);
  }
}
public Class Demo //Driver class for running program
{
  public static void main (String args[])
  {
    Student s = new Student ("Rajesh", 27);
    s.display();
  }
}
```

output: Name is: Rajesh
Age is: 27

2.6 Garbage Collection

Garbage Collection is java's process of automatically freeing up memory by destroying unused objects.

Automatically runtime memory management where java reclaims memory from objects no longer in use.

In other words, Garbage Collection is a way to destroy the unused objects. To do so, we were using free() function in C language and delete () in C++. But in java it is performed automatically.

→ Java memor efficient because gc removes the unreferenced objects from heap memory.

→ It is automatically done by the garbage collector (a part of JVM) so we don't need to make extra efforts.

→ finalize method is to perform clean up processing

(How can an object be unreferenced (destroyed)?

There are different ways to destroy the objects in java:

(i) By nulling the reference
(ii) By assigning a reference to another
(iii) By anonymous object etc.

(i) By nulling the reference

```
Student St = new Student();
St = null;
```

→ The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

(ii) By assigning a reference to another:

```
Student St1= new Student( );
Student st2 = new Student();
st1 = st2; // now st1 is available for
              garbage collecton.
```

(iii) By anonymous Object

```
new Student();
```

Note: Neither finalization nor garbage collection is guaranteed. finalize method is invoked each time before the object is garbage collected. It is used to perform cleanup processing. This method is defined in object class as:

```
protected void finalize()        // Also public but not private
{
}system.out.print(" object is destroyed");
```

Simple program example of garbage collection in java.

```
class GarbageCollection
{ protected void finalize()
  {system.out.println("object is destroyed!");
  }
  public static void main(String args[])
  {
    GarbageCollection g1 = new GarbageCollection();
    GarbageCollection g2 = new GarbageCollection();
     g1=null
     g2=null
     System.gc();
  }}
```

output:

```
object is destroyed
object is destroyed
```

2.7. This Keyword

This keyword can be used within any method (constructor, user-defined method) to represent the current object of the class.

The main purpose is to used this keyword is to avoid naming conflicts of class Data members and constructor parameters.

```
Class person
{
  String name; //instanse variable
  person(String name) // Parameter of constructor
  {
    this.name = name; // this refers to the current
  }                   instance variable/object.
   void display()
   {
    System.out.println("Name is " +this.name);
  }}
  public class Demo
  {
    public static void main(String args[])
    {
      person p = new person("Rajesh");
      P.display();
  }}
```

output: Name is Rajesh

Syntax:

```
Class Exampleofthis
{
    int x;
    Exampleofthis(int x)
    {
      this.x = x; // 'this' refers to the instance variable
    }
}
```

this keyword can be used for different following purposes.

→ this keyword is used to refer to current object.
→ this is always a reference to the object on which method was called.
→ this can be used to call current class constructor.
→ this can be passed as an argument to another method.

Example of reference variable & this.

```
Class A
{
  void display()
  {
    System.out.println(this);
  }
  public static void main(String args[])
  {
    A obj = new A();
    System.out.println(obj);
    obj.display();
  }
}
```

output: print the same referenced ID two times.

Example of calling the class method with "this"

```
class A
{
  void print()
  {
    this.show();
    System.out.println("print method");
  }
  void show()
  {
    System.out.println("show method");
  }
}
class Demo {
  public static void main(String args[])
  {
    A obj = new A();
    obj.print();
  }
}
```

output: show method
print method

2.8 Static fields and methods

The static keyword is used for memory management. we can apply static keywords with java variables, methods, blocks and nested class.

Static fields : Static fields are static variables. Static variable refer the common property of all objects.

Ex: Company name of employees
College name of Students

Static variable will get the memory only once, and retain its value if any object changes the value of the static variables

Examples:

without Static variable

```
Class Counter
{
  int count=0;
  Counter()
  { count++;
    System.out.println(count);
  }
  public static void main(String
  args[])
  { Counter C1= new Counter();
    Counter C2 =new Counter();
    Counter C3 =new Counter();
  }
}
```

output:

1
1
1

with Static variable

```
Class Counter
{
static int count=0;
  Counter()
  { count++;
    System.out.println(count);
  }
public static void main(String
args[])
{ Counter C1 = new Counter();
  Counter C2 = new Counter();
  Counter C3 = new Counter();
}
}
```

output:

1
2
3

Example 2

program showing use of static variable

```
class student
{
  int student_rollno;
  String student_name;
  static String college_name = "amc";
  student(int r, String n)
  {
    student_rollno = r;
    student_name = n;
  }
  void display()
  {
    System.out.println(student_rollno + " " + student_name + " " + college_name);
  }
  public static void main(String args[])
  {
    student S1 = new student(10, "Alina");
    student S2 = new student(11, "sunil");
    student S3 = new student(12, "Laxmi");
    student S4 = new student(13, "surakxya");
    S1.display();
    S2.display();
    S3.display();
    S4.display();
  }
}
```

output:

```
10 Alina amc
11 sunil amc
12 Laxmi amc
13 surakxya amc
```

Static methods

A method is static when you used the static keyword with it. A static method belongs to the class rather than object of class.

→ A static method can be called without creating an object of a class so the main method in java must be static method.

→ Static method can access static data members and can change the value of it.

Example

```
class Calculator
{
  static int add(int a, int b)
  {
    return a+b;
  }
  void display()
  {
    System.out.println("This is a non
                        static method.");
  }
  public class Main{
    public static void main(String args[]){
      int result;
      result = calculator.add(10,20)
      System.out.println(result);
      Calculator calC = new Calculator();
      Calc.display();
    }
  }
```

output: 30
This is a non-static method.

Example 2: Access static data and change the value of it

```
class student
{
  int rollno;
  String name;
  Static String college = "Rastriya";
  Static void change()
  {
    college = "cmc";
  }
  Student(int r, String n)
  {
    rollno = r;
    name = n;
  }
  Void display()
  {
    System.out.println(rollno + " " + name + " " + college);
  }
  public static void main(String args[])
  {
    Student.change();
    Student S1 = new Student(10, "Alina");
    Student S2 = new Student(11, "Laxmi");
    Student S3 = new Student(12, "Sunil");
    S1.display();
    S2.display();
    S3.display();
  }
}
```

output:

```
10 Alina cmc
11 Laxmi cmc
12 Sunil cmc
```

2.10 variable Length Arguments (varargs)

The varargs allows the method to accept zero or multiple arguments. It is used when you are creating a function and you don't know in advance that how many arguments will be passed. Internally, the arguments are treated as an array.

Syntax:

```
return type methodName ( datatype ... varName)
{
    // method body
}
```

→ The ... (three dots) is used to indicate that the method accepts variable arguments.

→ Inside the method, the varargs behave like an array.

Program Example

```
Class varargsExample {
static void display_Numbers (int... numbers)
{
  System.out.println("Numbers of arguments: "
                     + numbers.length);
    for (int num : numbers)
      {System.out.println(num);
    }}
public class main
{public static void main(String[] args)
  {
```

```
varargsExample.displayNumbers(1,2,3,4,5);
varargsExample.displayNumbers(10,20);
    }
}
```

output:

```
Number of arguments: 5
1
2
3
4
5
Number of arguments: 2
10
20
```

3.2 Inheritance and Constructors

⇒ In java, when a subclass / child class is instantiated, the constructor of the superclass is called first. If no explicit call to the superclass constructor is made, the default constructor of the superclass is automatically invoked.

When an object is created of the derived class, it will call the both of the constructor method from superclass as well as child class. but always call the constructor of the parent class first.

Example:-

```
class Base
{
  Base()
  {
    System.out.println("Base class Constructor called.");
  }
}
class Derived extends Base
{
  Derived()
  {
    System.out.println("Derived class Constructor called.");
  }
}
public class inherit {
   public static void main(String[] args){
     Derived d = new Derived();
} }
```

output:

```
Base class Constructor called.
Derived class Constructor called
```

In this example, creating an instance of 'Derived' results in the 'Base' constructor being called first, followed by the 'Derived' constructor.

3.3 'super' keyword

The 'super' keyword in java is used within a subclass to refer to its immediate superclass. super keyword can be used to call

1. superclass methods
2. Access superclass fields, and
3. Call superclass constructors.

1. Superclass methods

```
class Base{
  void B()
  {
    System.out.println("Base class method called.");
  }
}
class Derived extends Base{
  void B()
  {
    System.out.println("Derived class method called.");
  }
  void display()
  {
    super.B(); // call the superclass method
    B();
  }
}
  public class inherit{
    Derived d = new Derived();
    d.display();
}}
```

output:

Base class method called.
Derived class method called.

(2) Accessing superclass Constructors

```
class Animal{
  Animal()
  {
    System.out.println("Animal constructor called");
  }
}
class Dog extends Animal
{
  Dog()
  {
    super(); // call the superclass constructor
    System.out.println("Dog constructor called");
  }
}
public class Inherit{
  public static void main(String[] args)
  {
    Dog dog = new dog();
  }
}
```

output:
Animal Constructor called
Dog constructor called

(3). Accessing Superclass fields

```
Class A
{
  String name = "Rajesh Parajuli";
}
Class B extend A
{
  String name = "Rajesh";
  Void Display_Names()
  {
  System.out.println("class A Name:  " + super.name
  System.out.println("Class B Name: " + name);
  }
  public class inherit {
   public static void main(String[] args)
   {
     B bb = new B();
     bb.Display_Names();
   }
  }
```

Output:

```
Class A Name : Rajesh Parajuli
Class B Name : Rajesh
```

3.4 Method overriding :- Member function overriding

The process of redefining the inherited method of the super class in the derived class is called method overriding. The return type and the parameters must be same in both the super class and the subclass. The '@override' annotation is often used to indicate that a method is being overridden.

The overridden method in the subclass should have same access modifier or less restrictive than that of super class. for example, if the overridden method in super class is protected, then it must be either protected or public in the subclass but not private.

example

```
Class A {
String name = "class A";
Void display() {
System.out.println("Display from "+name);
}
Class B extends A{
 String name = "Class B";
 @Override
  Void display()
  {
  System.out.println ("Display from "+ name);
  }
   Void displayBoth()
  {
   System.out.println("superclass name:"+super.name);
  } System.out.println("Subclass name:"+name);
```

```
public class Inherit{
  public static void main (String args[])
  {
      B b = new B();
      A a = new A();

      a.display(); // Display from class A
      b.display(); // Display from class B
      b.displayBoth(); // Displays from Both class
  }
```

output:

```
Display from class A
Disply from class B
Superclass name : class A
Subclass name : class B
```

Unit4: Exception Handling and Multithreading

An Exception can be anything that interrupts the normal flow of the program. When an exception occurs program gets terminated and doesn't continue further. It is an object which is thrown at runtime.

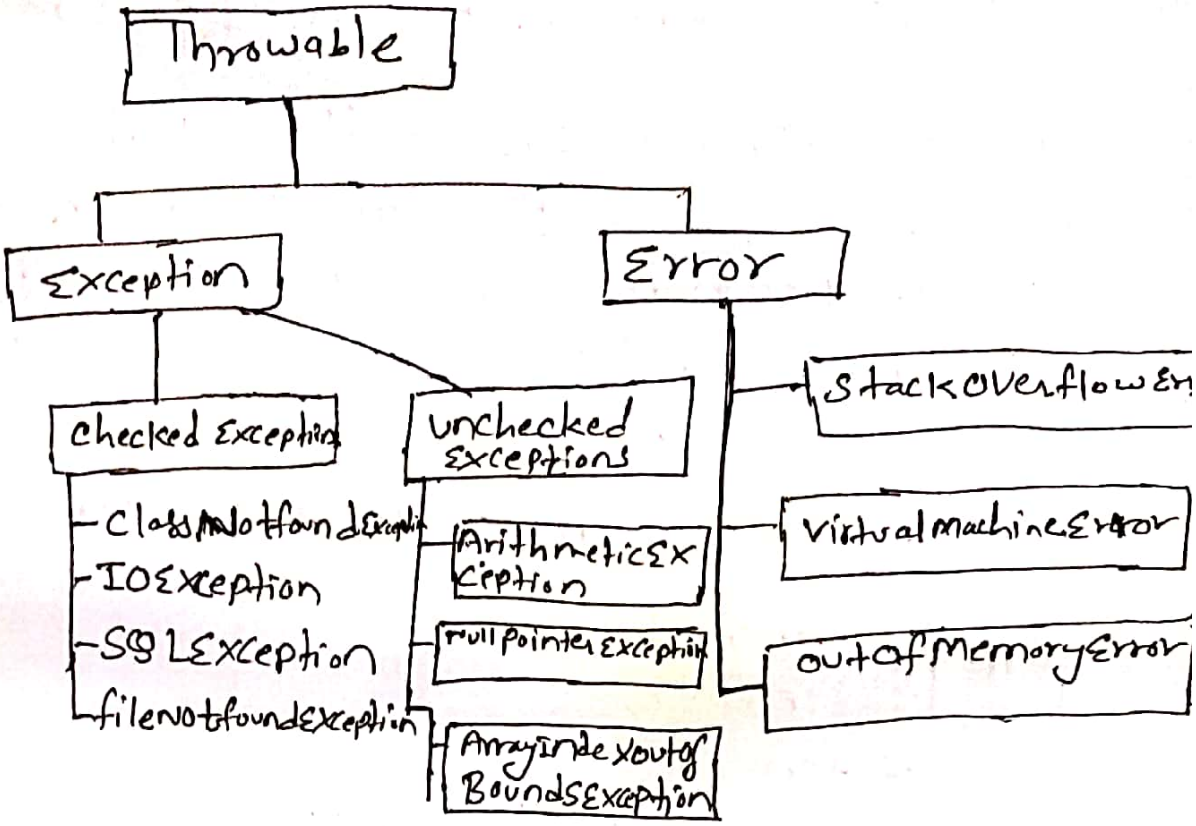
Let's understand from a scenario:

```
Statement1;
Statement2;
Statement3;
statement4; // exception occurs
Statement5;
Statement6;
  :
  :
Statementn;
```

Suppose there are 10 statements in java program and an exception occurs at statement 4, the rest of the code will not be executed, i.e., Statements 5 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in java.

4.1 The Exception Hierarchy

In java, exceptions are represented in a class hierarchy rooted at 'java.lang.Throwable', which is divided into two branches:



1 Exception : Represents exceptions that can be caught and handled by the program.

(1) Checked Exception: Exceptions that are checked at compile time.
IOException, SQLException are example of Checked Exceptions that must be either caught or declared in the method.

(2) unchecked Exception: An exception that occurs at the time of (Runtime) execution.
Also known as Runtime Exceptions.
NullPointerException, ArrayIndex-outofBoundsException, ArithmeticExce-ption are the Example of unchecked Exception.

2 Error: Represents serious system errors (e.g outofmemory Error, stackoverflowError) that usually cannot be handled.

Errors are the critical conditions that occur due to the lack of the system resources, and it cannot be handled by the code of the program. The code is not responsible for such errors. The result of the occurrence of the error is that the program gets terminated abnormally.

4.2 Exception Handling fundamentals

Exception Handling is a mechanism to handle the exceptions. Java provides five keywords that are used to handle the exception.
try, catch, finally, throw, throws.

1. Syntax of try-catch-finally

```
try
{
  //code that might throw an exception
}
catch(ExceptionType1 e1)
{ Code to handle ExceptionType1
}
catch(ExceptionType2 e2)
{ //code to handle ExceptionType2
}
finally
{ //code that will always execute (optional, but used for cleanup
```

2. Using throw to manually Throw an exception

Syntax:

```
throw new ExceptionType("Exception message");
```

3. Method Declaration with throws

Syntax:

```
returntype methodName() throws ExceptionType1,
                                ExceptionType2
{
  // method code that might throw exceptions
}
```

All keywords using syntax!

```
public return_type method_Name() throws Exception
Type1, ExceptionType2
{
  try
  {
    //code that might throw an exception
    throw new ExceptionType1("exception message");
  }
  Catch(ExceptionType1 e1)
  {
    // Handle ExceptionType1
  }
  catch(ExceptionType2 e2)
  {
    //Handle Exception Type2
  }
  finally
  {
    // Code that will always run(e.g. resource clean up)
  }
}
```

Divide by zero program occurs Arithmetic Exception

```
Class ExceptionExample
{
  public static void main (String args[])
  {
     int a=10;
     int b=5;
     int c=5;
     int res = a/(b-c); // exception occurs
     System.out.println (res);
  }
}
```

Now, we are handling this Arithmetic exception using Try, catch blocks.

```
Class ExceptionHandle
{
  public static void main(String args[])
  { int a=10,b=5, c=5;
    try
    {
      int res = a/(b-c);
    }
    catch(ArithmeticException e)
    {
      System.out.println(" Division by zero");
    }
```

```
finally
{
    System.out.println("This block is always
                        executed");
}
}
}
```

output: Division by zero
This block is always executed

Example program using throw keyword:

→ throw keyword is used to throw an exception message explicitly.

```
public class ThrowAE
{
    public static int divide(int x, int y)
    {
        if(y==0)
        {
            throw new ArithmeticException("cannot
            divide by zero");
        }
        return x/y;
    }
    public static void main(String args[])
    {
        try
        {
            int x = 10;
            int y = 0;
            int result = divide(x, y);
            System.out.println(result);
        }
```

```
Catch (ArithmeticException e)
    System.out.println("Error: " + e.getMessage());
}
}
```

Output:

Error: Cannot divide by zero.

Example program for ArrayIndexOutOfBoundsException Handling:

```
public class ArrayOutOfBoundExample
{
public static void main(String args[])
{
try
{
int[] numbers = {1,2,3,4,5};
System.out.println("Element at index 10: "+numbers[10]);
}
Catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("Error: " + e.getMessage());
System.out.println("Array index is out of bounds.
                    Please Check the index value.");
}
finally
{
System.out.println("Array access attempt completed.");
}
}
}
```

output:

~~Error: Element at index 10:~~

Error: Index 10 out of bounds for length 5

Array index is out of bounds. please check the index value.

Array access attempt Completed.

Keywords Concepts:

try: Block of code where exceptions might occur.

Catch: Block of code that handles the exception.

finally: Block of code that always executes, regardless of whether an exception was thrown or caught.

throws: used to declare that a method may throw an exception.

Example:

```
try
{
  int result = 10/0;
}
catch (ArithmeticException e)
{
  System.out.println("error: "+ e.getMessage());
}
finally
{
  System.out.println("End of Exception handling");
}
```

output: error: / by zero
End of Exception handling.

4.3 Throwing, Re-throwing, and Catching Exceptions

↳ Throwing an exception: you can use the throw keyword to manually throw an exception.

Syntax: `throw new ArithmeticException("custom division by zero error");`

↳ Re-throwing an exception: After catching an exception you can rethrow it to let the calling method handle it.

Syntax:

```
try
{
  // code
}
catch (Exception e)
  {
    throw e;
  }
```

↳ Cathing exceptions: Exceptions are caught using the catch block, which sould match the type of the thrown exception.

```
try
{
  // code that might throw an exception
}
catch (IOException e)
{
  //handle the specific exception
}
```

4.5 Multithreading fundamentals

Process and Thread are two basic units of java program execution. A program in an execution is process. process can be seen as a program or application.

Thread is a segments of process. It is lightweight process. Thread requires less resources to create and exists in the process. Thread shares the process resources

multithreading in java is a process of executing multiple processes simultaneously.

A program is divided into two or more subprograms, which can be implemented at the same time in parallel.

multiprocessing & multithreading both are used to achieve multitasking.

Threads are independent so it doesn't affect other threads.

Multithreading can perform many operations together So it saves time.

Threads are implemented in the form of objects.

↳ The run() and start() are two inbuilt methods which helps to thread implementation.

↳ The run() method can be initiating by the calling of start() method.

Thread is a class found in java.lang package.

Thread can be Created in two ways:

(i) By extending Thread class

(ii) By implementing Runnable interface.

Multithreading is a process of executing multiple threads concurrently. A thread is a lightweight process, and java supports multithreading to make better use of cpu resources.

Benefits

→ faster execution by performing multiple tasks simultaneously
→ Better utilization of cpu resources.

1. By Extending Thread class

```
class multi extends Thread
{
  public void run()
  {
    System.out.println("thread is running");
  }
  public static void main(String args[])
  {
    multi t1 = new multi(); // object initiated
    t1.start(); // run() method called through
                start()
  }
}
```

output: thread is running

2. By implementing Runnable interface

Example

```
class multi implements Runnable
{
  public void run()
  {
    System.out.println("thread is running");
  }
  public static void main(String args[])
  {
    multi m1 = new multi();
    Thread t1 = new Thread(m1);
    t1.start();
  }
}
```

output: thread is running.

JDBC

Java Database Connectivity. It is an API which is used to connect java application with the database and execute queries.

JDBC provides a standard method for ~~creating~~ Connecting to a database, executing SQL queries, and retrieving results.

There are five (5) steps to connect database in java.

(i) Register the Driver class : The drive class will be different for different database. Class.forName() method is used to register the Driver class ~~forer~~

(ii) Creating Connection object :- Establishing a connection to the database using DriverManager.getConnection().

DriverManager.getConnection("url", "user_name", "password");

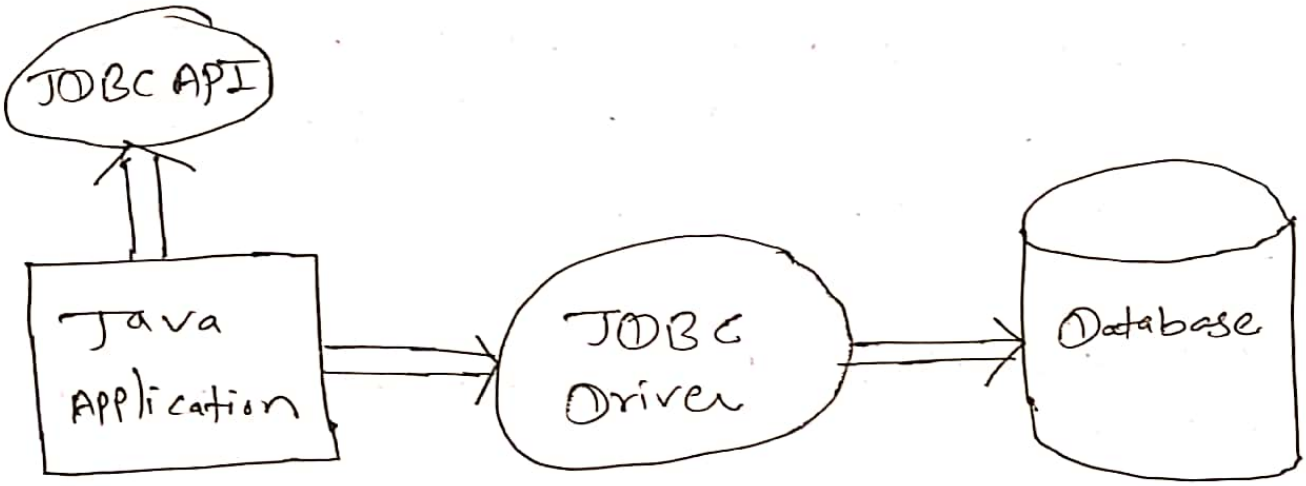
(iii) Creating statement object :- Creating statement object for sending SQL statements to the database. using Connection.createStatement().

eg Connection.createStatement();

(iv) Executing queries :- Executing queries that returns a ResultSet. Statement.executeQuery(sql);

(v) Closing Connection :- Closing the connection using Connection.close(); function.

figure: Java Database Connectivity
JDBC Connection



JDBC API → provides classes and interfaces to connect java application with database.

JDBC Driver → Software component that enables java applications to interact with a specific database.

1. JDBC-ODBC Bridge Driver
2. Native-API Driver
3. Network protocol Driver
4. Thin Driver (pure java)

Java Application → program written in java that uses JDBC API to connect to the Database, execute SQL statements and handle the results.

Database: Collection of relational data eg mysql, SQL server, oracle .. etc.

JDBC Connection program in java (Notepad file) using Command prompt.

```
import. java.sql.*; //Import the SQL package
Public class database
{
  Public static void main (String args[] throws
   Exception
  {
    try{ //Load the mysql JDBC driver
        Class.forName ("com.mysql.cj.jdbc.Driver");

    //Establish a connection to the database "db_name" on
      localhost.

    Connection con = DriverManager.getConnection("
                    jdbc:mysql://localhost:3306/
                    testdb", "root", " ");

    System.out.println("Connection to database successfully
                        Established");

    //Create a statement object to execute SQL query
    Statement stmt = Con.createStatement();

    //Execute query to select all records from the
     'users' table_name table.
     ResultSet rs = stmt.executeQuery("select *
                                from users");

     while(rs.next())
     {
       System.out.println(rs.getInt("id") + " " +
                          rs.getString("name");

     }
```

```
        //close the resources
        stmt.close();
        con.close();
    }
    catch (Exception e)
    {
        System.out.println(e);
    }
}
}
```

To Compile : javac -cp driver.jar;. database.java

To Run : java -cp driver.jar;. database