CHAPTER 5

EXCEPTION HANDLING AND MULTITHREADED PROGRAMMING

CHAPTER OUTLINE

5.1 DIFFERENT SOURCES OF ERRORS

Different types of errors are occurred during compilation and running a java program. The basic types as follows :

1. **Syntax Error / Compile time Error :** Syntax errors are caused by violations of the syntax of Java. Such as missing semicolon at end of the statement.

   Mismatched parenthesis,unclosed string literal, illegal start of expressions etc.,

```
class A
{
public static void main(String args[])
{
int a=10
int b=20;
int c=a +b;
System.out.println(c);
}
}
```

2. **Run time Error :** While program is under the execution, Run time errors are occurred because of invalid input is given to the program. For example divided by zero error, array out of bounds error etc.,

```
class A
{
public static void main(String args[])
 {
int a=10;
int b=0;
int c=a/b;
System.out.println(c);
 }
}
```

3. **Linker error :** It occurs due to the absence of method definition in a program.

```
class triangle
{
double base ;
double height;
triangle (double b, double h)
 {
   base = b;
   height = h;
 }
public static void main(String args[])
 {
   triangle t1=new triangle(10.52,20.72);
   t1.findarea();
 }
}
```


4. **Logical Error :** Logical error is a bug in a program that causes it to operate incorrectly, but not terminate abnormally. A logical error produces undesired output.

   For example the let us consider statement a+b/2.It performs division operation before addition operation.

```
class A
{
public static void main(String args[])
{
int a=10;
int b=20;
int c=a+b/2;
System.out.println(c);
}
}
```

ADDITIONAL INFORMATION

Introduction : The languages like C and Pascal does not support exception handling. Hence the applications of C and Pascal are not said to be Robust applications.

The languages like java and .NET supports exception handling. Hence the applications of java and .NET are said to be robust applications.

5.2 HOW TO DEAL WITH EXCEPTIONS AND TYPES OF EXCEPTIONS

WHAT IS AN EXCEPTION?

Run time errors of java are called ***System Error Messages.***

Run time errors of java are called ***Exceptions.***

An unwanted event disturbs the normal flow of execution in a program is called an ***Exception.***

System Error Messages of java are called ***Run time errors*** which are nothing but Exceptions.

The process of converting System Error Messages into User friendly error messages is called ***Exception Handling.***

The process of handling run time errors of java is called ***Exception Handling.***

The exception handling in java is one of the powerful mechanisms to handle the run time errors so that normal flow of the application can be maintained.

The core advantage of exception handling is to maintain the normal flow of the application.

Some of **advantages of Exception Handling** in Java are as follows.

- Provision to Complete Program Execution.
- Easy Identification of Program Code and Error-Handling Code.
- Propagation of Errors.
- Meaningful Error Reporting.
- Identifying Error Types.

Exception normally disrupts the normal flow of the application that is why we use exception handling. Let us consider the following code to understand need of an exception handling.

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
```

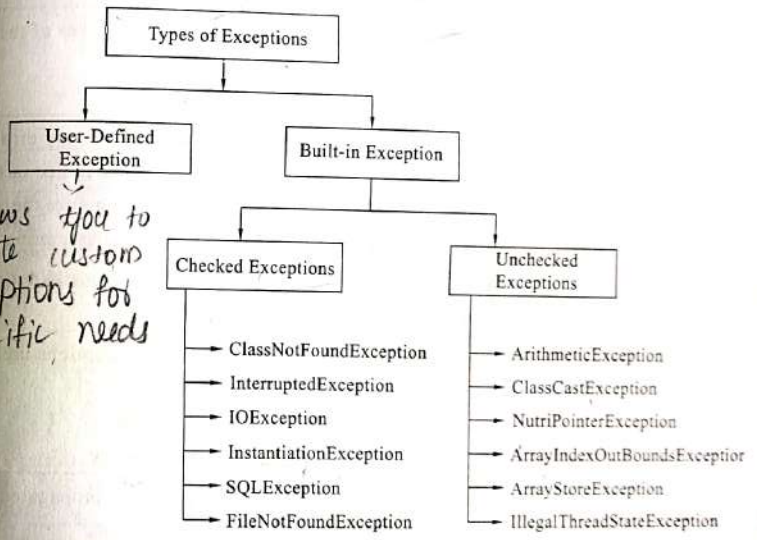


```
statement 7;
statement 8;
statement 9;
statement 10;
```

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 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.

TYPES OF EXCEPTION IN JAVA

Exceptions can be categorized into two ways. They are Built-in Exceptions and User-Defined Exceptions. Built-in Exceptions are again classified into two types. They are Checked exceptions and Unchecked exceptions.



allows you to create custom exceptions for specific needs

BUILT-IN EXCEPTION

Exceptions that are already available in Java libraries are referred to as built-in exceptions. They can be categorized into two broad categories, i.e., checked exceptions and unchecked exception.

CHECKED EXCEPTIONS

Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. The compiler ensures whether the programmer

handles the exception (or) not. The programmer should have to handle the exception; otherwise, the system will show a compilation error.

UNCHECKED EXCEPTIONS

The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle (or) declare it, the program would not give a compilation error. Usually, it occurs when the user provides bad data during the interaction with the program.

Difference Between Checked and Unchecked Exception :

S.No.	Checked Exception	Unchecked Exception
1.	These exceptions are checked at compile time. These exceptions are handled at compile time too.	These exceptions are just opposite to the checked exceptions. These exceptions are not checked and handled at compile time.
2.	These exceptions are direct subclasses of exception but not extended from RuntimeException class.	They are the direct subclasses of the RuntimeException class.
3.	The code gives a compilation error in the case when a method throws a checked exception. The compiler is not able to handle the exception on its own.	The code compiles without any error because the exceptions escape the notice of the compiler. These exceptions are the results of user-created errors in programming logic.
4.	These exceptions mostly occur when the probability of failure is too high.	These exceptions occur mostly due to programming mistakes.
5.	Common checked exceptions include IOException, DataAccessException, InterruptedException, etc.,	Common unchecked exceptions include ArithmeticException, InvalidClassException, NullPointerException, etc.,
6.	These exceptions are propagated using the throws keyword.	These are automatically propagated.
7.	It is required to provide the try-catch and try-finally block to handle the checked exception.	In the case of unchecked exception it is not mandatory.

ArithmeticException ArrayIndexOutOfBoundExceptions, ClassNotFoundExceptions etc., are come in the category of Built-in Exception. Sometimes, the built-in exceptions are not sufficient to explain (or) describe certain situations. For describing these situations, we have to create our own exceptions by creating an exception class as a

subclass of the Exception class. These types of exceptions come in the category of User-Defined Exception.

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

EXAMPLES OF BUILT-IN EXCEPTION

1. **Arithmetic exception :** It is thrown when an exceptional condition has occurred in an arithmetic operation.

```
Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
public static void main(String args[])
  {
    try {
      int a = 30, b = 0;
      int c = a / b; // cannot divide by zero
      System.out.println("Result = " + c);
    }
    catch (ArithmeticException e)
{
      System.out.println("Can't divide a number by 0");
    }
  }
}
```

Output :

```
Can't divide a number by 0
```

2. **ArrayIndexOutOfBounds Exception :** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative (or) greater than (or) equal to the size of the array.

```
// Java program to demonstrate ArrayIndexOutOfBoundException
class ArrayIndexOutOfBound_Demo
{
```

```
public static void main(String args[])
  {
    try {
      int a[] = new int[5];
      a[6] = 9; // accessing 7th element in an array of
      // size 5
    }
    catch (ArrayIndexOutOfBoundsException e)
{
      System.out.println("Array Index is Out Of Bounds");
    }
  }
}
```

Output :

```
Array Index is Out Of Bounds
```

3. **FileNotFoundException :** This Exception is raised when a file is not accessible (or) does not open.

```
// Java program to demonstrate
// FileNotFoundException
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class File_notFound_Demo
{
public static void main(String args[])
{
try {
// Following file does not exist
      File file = new File("E:// file.txt");
      FileReader fr = new FileReader(file);
    }
catch (FileNotFoundException e)
```



```
{
System.out.println("File does not exist");
    }
  }
}
```

Output :

```
File does not exist
```

4. **IOException :** It is thrown when an input-output operation failed or interrupted

```
// Java program to illustrate IOException
import java.io.*
;class Geeks
{
public static void main(String args[])
{
FileInputStream f = null;
f = new FileInputStream("abc.txt");
int i;
while ((i = f.read()) != -1)
{
System.out.print((char)i);
    }
      f.close();
    }
}
```

Output :

```
error : unreported exception IOException; must be caught or declared to be
thrown
```

5. **InterruptedException :** It is thrown when a thread is waiting, sleeping (or) doing some processing, and it is interrupted.

```
// Java Program to illustrate
// InterruptedException
```

```
class Geeks
{
public static void main(String args[])
{
    Thread t = new Thread();
t.sleep(10000);
}
   }
```

Output :

```
error : unreported exception InterruptedException; must be caught or declared
to be thrown
```

6. **NumberFormatException :** This exception is raised when a method could not convert a string into a numeric format.

```
// Java program to demonstrate NumberFormatException
class NumberFormat_Demo
{
public static void main(String args[])
  {
    try
{
          int num = Integer.parseInt("anil");
         System.out.println(num);
    }
     catch (NumberFormatException e)
{
         System.out.println("Number format exception");7
    }
   }
}
```

Output :

```
Number format exception
```


SUMMARY

Hierarchy of Java Exception classes :

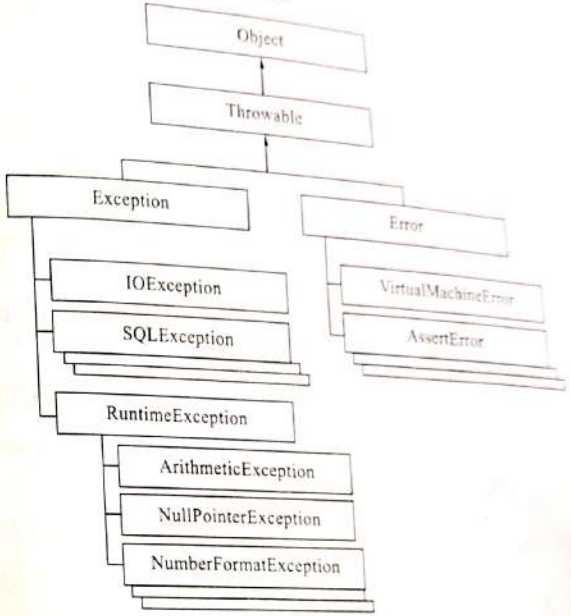


FIG 5.1 :

TYPES OF EXCEPTIONS

There are mainly two types of exceptions. They are.

- Checked Exceptions.
- Unchecked Exceptions.

Where error is considered as unchecked exception.

Difference between Checked and Unchecked Exceptions :

Checked Exceptions : The classes that extend Throwable class except RuntimeException and Error are known as *checked exceptions* e.g. IOException, SQLException.

Checked exceptions are checked at compile-time.

Unchecked Exceptions : The classes that extend RuntimeException are known as unchecked exceptions. **Eg :** ArithmeticException, NullPointerException NumberFormatException, ArrayIndexOutOfBoundsException etc.,

Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

ADDITIONAL INFORMATION

Differences between Errors and Exceptions :

S.No.	Error	Exception
1.	The error indicates trouble that primarily occurs due to the scarcity of system resources.	Exception is an unwanted (or) unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program.
2.	It is not possible to recover from an error.	It is possible to recover from an exception.
3.	In java, all the errors are unchecked.	In java, the exceptions can be both checked and unchecked.
4.	The system in which the program is running is responsible for errors.	The code of the program is accountable for exceptions.
5.	They are described in the java.lang.	They are described in java.lang.Exception package.

5.3 GENERAL FORM OF EXCEPTION-HANDLING BLOCK

Exception handling can be performed by using five keywords. They are.

try
catch
throw
throws
finally

Syntax to perform Exception handling is given below.

```
try
{
Block of state
ments which may arise an exception
}
catch(type of exception1  obj1)
{
block of statements which provides exception handling /user friendly messages;
}
catch(type of exception2  obj2)
{
```



```
block of statements which provides exception handling /user friendly messages;
}
finally
{
block of statements;
}
}
```

1. try block contains some set of statements in which any of one of statements may arise an exception.
2. Whenever an exception occurs in try block, execution is terminated abnormally and the appropriate catch block is executed.
3. Every try block is immediately followed by one (or) more number of catch blocks.
4. Every catch block is contains set of statements which are used to provide user friendly message(exception handling code).
5. If no exception occurs in try block then no catch block will be executed.
6. The finally block will be executed irrespective of whether exception occurs in try block (or) not.

Note : Writing finally block is optional. finally block is used to close the resources which are obtained in try block.

EXAMPLE

Write a java program by using exception handling concept.

```
class div
{
public static void main(String args[])
{
try
{
intx,y,z;
x=10;
y=0;
z=x/y;
```

```
System.out.println("Division of numbers is "+z);
}
catch(ArithmeticException  x )
{
System.out.println("Donot take zero for denominator");
}
finally
{
System.out.println("I am in finally block");
}
}
}
```

Output :

```
Do not take zero for denominator
I am in finally block
```

5.4 CONCEPT OF MULTI-CATCH STATEMENTS WITH EXAMPLE PROGRAMS

Multiple Catch Blocks : Catching Multiple Exceptions.

In Java, a single try block can have multiple catch blocks in exception handling. When statements in a try block generate multiple exceptions, we require multiple catch blocks to handle different types of exceptions. This mechanism is called multi-catch block in java.

Each catch block is capable of catching a different exception. That is each catch block must contain a different exception handler.

The syntax for using a single try with more than one catch block in java is as follows :

Syntax :

```
try
{
Block of statements which may raise exceptions
}
catch(type of exception-1  obj1)
{
```



```
block of statements which provides exception handling /user friendly  messages;
}
catch (type of exception-2  obj2)
{
block of statements which provides exception handling /user friendly  messages;
}
catch (type of exceptioin-Nobj-N)
{
block of statements which provides exception handling /user friendly  messages;
}
```

Example: Write a java program to catch multiple exceptions

```
class ExceptionExample
{
public static void main(String args[])
{
int num1 = 10;
int num2 = 0;
int result = 0;
inta[] = new int[5];
try
{
a[0] = 10;
    a[1] = 20;
    a[2] = 30;
    a[3] = 40;
    a[4] = 50;
    a[5] = 60;
System.out.println("Result of Division : " + result);
}
catch(ArithmeticException e)
{
    System.out.println("Error: Divided by Zero");
}
```

result = num1 / num2;

```
catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("Error: Array Out of Bound");
}
```

finally { s.o.pln("finally block is always executed"); }

In the above example we have two lines that might throw an exception i.e

a[5] = 60;

above statement can cause array index out of bound exception and

result = num1 / num2;

this can cause arithmetic exception.

To handle these two different types o f exception we have included two catch blocks for single try block.

```
catch (ArithmeticException e)
{
    System.out.println("Err: Divided by Zero");
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("Err: Array Out of Bound");
}
```

When an exception is thrown inside the try block then type of the exception thrown is compared with the type of exception of each catch block.

If type of exception thrown is matched with the type of exception from catch then it will execute corresponding catch block.

Notes :

- **At a time only single catch block can be executed.** After the execution of catch block control goes to the next statement after the try block.
- **At a time only single exception can be handled.**

ADDITIONAL INFORMATION

JAVA FINALLY BLOCK

Java finally block is a block that is used *to execute important code* such as closing connection, closing a file etc.,

Java finally block is always executed whether exception is handled (or) not.

Java finally block must be followed by try (or) catch block.

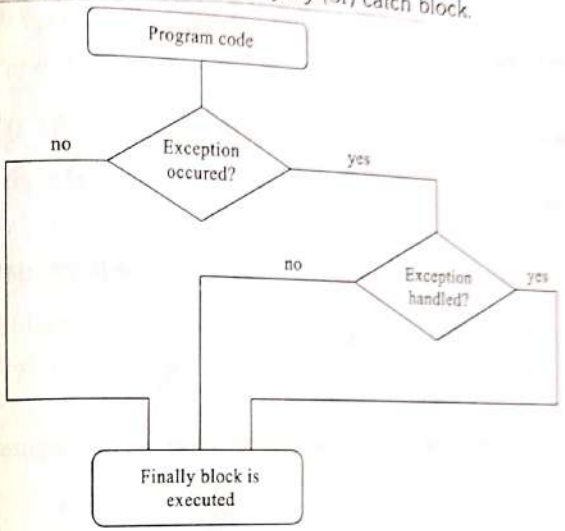


FIG 5.2 :

Finally block in java can be used to utilize resources such as closing a file, closing connection etc.,

Let's see the different cases where java finally block can be used.

Case 1 : Let's see the java finally example where exception doesn't occur.

```
class TestFinallyBlock
{
public static void main(String args[])
{
  try
{
   int data=25/5;
   System.out.println(data);
  }
  catch(ArithmeticException e)
{
System.out.println(e);
}
```

```
finally
{
System.out.println("finally block is always executed");
}
System.out.println("rest of the code...");
 }
}
```

Output :

```
5
finally block is always executed
rest of the code...
```

Case 2 : Let's see the java finally example where exception occurs and handled.

```
public class TestFinallyBlock2
{
 public static void main(String args[])
 {
 try
{
  int data=25/0;
  System.out.println(data);
 }
 catch(ArithmeticException e)
{
System.out.println(e);
}
 finally
{
System.out.println("finally block is always executed");
}
  System.out.println("rest of the code...");
}
}
```


Output :

```
Exception in thread main java.lang.ArithmeticException:/ by zero
finally block is always executed
rest of the code...
```

5.5 NESTED TRY STATEMENTS

Java Nested Try Block : The try block within a try block is known as *nested try block in java.*

NEED OF NESTED TRY BLOCK

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax of nested try block is given below.

```
....
try
{
    statement 1;
    statement 2;
    try
    {
        statements;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
```

EXAMPLE

Write a java program to illustrate nested try block.

```
class Excep9
{
 public static void main(String args[])
{
  try
{
    try
    {
      System.out.println("going to divide");
      int b =39/0;
    }
catch(ArithmeticException e)
    {
        System.out.println(e);
    }

    try
    {
    int a[]=new int[5];
    a[0]=10;
    a[1]=20;
    a[2]=30;
    a[3]=40;
    a[4]=50;
    a[5]=90;
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
      System.out.println(e);
    }
    System.out.println("other statement");
  }
```



```
    catch(Exception e)
    {
      System.out.println("handeled");}
    System.out.println("normal flow..");
    }
    }
```

Output :

```
going to divide
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException:5
other statement
normal flow..
```

5.6 THROW AND THROWS CLAUSES

Java throw exception/Java throw keyword : The Java throwkeyword is used to explicitly throw an exception. We can throw either checked (or) unchecked exception in java by throw keyword.

The syntax of java throw keyword is given below.

```
throw exception;
```

Let's see the example of throw IOException.

```
throw new IOException("sorry... device error);
```

The throw keyword is used to throw an exception within a method. When a throw statement is encountered in a program, execution of the current method is stopped and returned to the caller.

EXAMPLE

Write a java program to illustrate throw keyword.

```
public class Throw1
{
  static void validate(int age)
  {
    if(age<18)
    throw new ArithmeticException("you are not allowed to voting");
    else
```

```
        System.out.println("welcome to vote");
      }
      public static void main(String args[])
    {
        Throw1.validate(13);
        System.out.println("rest of the code...");
      }
    }
```

Output :

```
Exception in thread main java.lang.ArithmeticException:
you are not  allowed to voting
```

In the above program, we have created the validate method that takes integer value as a parameter. If the age is less than 18, the ArithmeticException is thrown otherwise message welcome to vote is printed.

JAVA THROWS KEYWORD

The java throws keyword is used to declare an exception. It gives information to the programmer that there may occur an exception so it is better to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions.

Syntax of java throws keyword is shown below.

```
return_type method_name() throws exception_class_name
{
//method code
}
Advantage of Java throws keyword
```

It provides information to the caller of the method about the exception.

The throws keyword is used to declare a method that may throw one (or) more exceptions. The caller has to catch the exceptions.

These two keywords are usually used together as depicted the following form:

```
void Method() throws Exception1, Exception2
{
```



```
// statements...
if(an exception occurs)
{
throw new Exception1();
}
// statements...

if(another exception occurs)
{
throw new Exception2();
}
}
```

EXAMPLE

Write a java program to illustrate throws keyword.

Let's see the example of java throws describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1
{
  void m()throws IOException
  {
    throw new IOException("dev err");//checked exception
  }
  void p()
  {
    try
    {
      m();
```

```
    }
  catch(Exception e)
    {
  System.out.println("exception handled");
    }
  }
   public static void main(String args[])
  {
    Testthrows1 obj=new Testthrows1();
    obj.p();
    System.out.println("normal flow..");
  }
}
```

Output :

```
exception handled
normalflow..
```

Difference between Throw and Throws in Java : There are many differences between throw and throws keywords. A list of differences between throw and throws are given below :

S.No.	Throw	Throws
1.	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2.	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3.	Throw is followed by an instance.	Throws is followed by class.
4.	Throw is used within the method.	Throws is used with the method signature.
5.	You cannot throw multiple exceptions.	You can declare multiple exceptions. **Eg :** public void method()throws IOException,SQLException.

ADDITIONAL INFORMATION

Difference Between Final, Finally and Finalize : There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below :

S.No.	Final	Finally	Finalize
1.	Final is used to apply restrictions on class, method and variable. Final class can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled (or) not.	Finalize is used to perform clean up processing just before object is garbage collected.
2.	Final is a keyword.	Finally is a block.	Finalize is a method.

5.7 CREATION OF USER DEFINED EXCEPTIONS

Java provides us the facility to create our own exceptions which are basically derived classes of Exception. Creating our own Exception is known as a custom exception (or) user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user needs.

User Defined Exception (or) custom exception is creating your own exception class and throws that exception using throw keyword. This can be done by extending the class Exception.

Following are a few of the reasons to use custom exceptions :

- To catch and provide specific treatment to existing Java exceptions.
- **Business logic exceptions :** These are the exceptions related to business logic and workflow. It is useful for the users (or) the developers to understand the exact problem.

In order to create a custom exception, we need to extend the Exception class that belongs to java.lang package.

EXAMPLE

Write a java program to illustrate creation of user defined Exceptions.

In the below program we pass the string to the constructor of the superclass Exception which is obtained using the "getMessage()" function on the object created.

```
// A Class that represents use-defined exception
class MyException extends Exception
{
    public MyException(String s)
    {
        // Call constructor of parent Exception
        super(s);
```

```
    }
}
// A Class that uses above MyException
public class Main
{
    public static void main(String args[])
    {
        try
        {
            // Throw an object of user defined exception
                throw new MyException("God");
    }
    catch(MyException ex)
        {
                System.out.println("Caught");
                System.out.println(ex.getMessage());
    }
    }
}
```

Output :

```
Caught
God
```

In the above code, the constructor of MyException requires a string as its argument. The string is passed to the parent class Exceptions constructor using super(). The constructor of the Exception class can also be called without a parameter and the call to super is not mandatory.

EXAMPLE

Write a java program to illustrate creation of user defined Exceptions.

```
// A Class that represents use-defined exception
class MyException extends Exception
{

}
// A Class that uses above MyException
```



```
public class setText
{
    // Driver Program
    public static void main(String args[])
    {
            try
    {
                    //Throw an object of user defined exception
                    throw new MyException();
            }
            catch (MyException ex)
    {
                    System.out.println("Caught");
                    System.out.println(ex.getMessage());
            }
    }
}
```

Output :

```
Caught
null
```

EXAMPLE

Write a java program to illustrate creation of user defined Exceptions.

Let's see a simple example of Java custom exception. In the following code, constructor of InvalidAgeException takes a string as an argument. This string is passed to constructor of parent class Exception using the super() method. Also the constructor of Exception class can be called without using a parameter and calling super() method is not mandatory.

```
TestCustomException1.java
// class representing custom exception
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
```

```
        super(str);
    }
}
// class that uses custom exception InvalidAgeException
public class TestCustomException1
{
    static void validate (int age) throws InvalidAgeException
{
    if(age < 18)
{
     // throw an object of user defined exception
     throw new InvalidAgeException("age is not valid to vote");
    }
    else
    {
      System.out.println("welcome to vote");
    }
  }
  public static void main(String args[])
  {
    try
    {
        validate(13);
    }
    catch (InvalidAgeException ex)
    {
      System.out.println("Caught the exception");
      // printing the message from InvalidAgeException object
      System.out.println("Exception occured: " + ex);
    }
    System.out.println("rest of the code...");
  }
}
```


Output :

```
Caught the exception
Exception occured: InvalidAgeException: age is not valid to vote
rest of the code...
```

5.8 THREAD AND LIFE CYCLE OF A THREAD

1. Thread is basically a lightweight sub-process, a smallest unit of processing.
2. Multithreading in java is a process of executing multiple threads simultaneously.
3. Threads share a common memory area.
4. Java Multithreading is mostly used in games, animation etc.,

Advantage of Java Multithreading :

(i) ***Enhancing Efficiency*** **:** It makes the application very efficient by making the optimal use of the CPUs.

(ii) ***Sharing Memory :*** The best part about multithreading is that the thread it uses does not occupy the memory but uses it on a shared basis. Hence, it also makes efficient use of memory by sharing it appropriately.

(iii) ***Reduces Run Time :*** It usually reduces the time utilized by an application to run the particular program. The time saved by threads could be used for other programs to minimize program execution time.

(iv) ***Used in Complex Applications :*** Threads in application development make it easy to create the modules in the application.

(v) Threads are independent so one thread doesn't affect other threads.

Life Cycle of Thread /States of Thread /Thread model of Java.

SOLUTION-1 :

Life cycle of a thread is classified into five states. They are

1. New state.
2. Ready state.
3. Running state.
4. Non runnable state.
5. Terminated/Dead/Halted state.

Life Cycle of Thread /States of Thread is given in below Fig. 5.3.

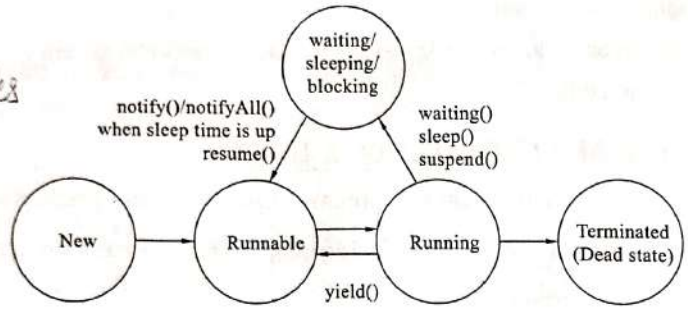


FIG 5.3 :

1. **New State :** A new state is one in which the thread is about to enter into main memory. **(or)** A new thread is being created by java virtual machine (JVM).
2. **Ready state :** Ready state is one in which the thread is under the control of main memory.
3. **Running state :** A running state is one in which the thread is under the control of CPU.
4. **Waiting/Blockedstate :** A thread is said to be in the Non-Runnable (Blocked) state if and only if it satisfies any one of the following conditions.
   (i) Making currently executing thread to wait for a long time.
   (ii) Making currently executing thread to sleep for a long time.
5. **Terminated state :** A Terminated/Dead/Halted state is one in which a thread completes its execution.

SOLUTION-2 :

Life cycle of a thread is classified into 5 states. They are.

1. **New State :** A new thread is being created.
2. **Runnable State :** A thread is in the in runnable state by calling start() method.
3. **Running State :** A Thread is executing that means Thread is in running stage. A thread is in the in running state by calling run() method.
4. **Terminated/Dead State :** A thread is in terminated state (or) dead state when its run() method exits.
5. **Waiting/Blockedstate :** This is the state when the thread is still alive, but it is currently not in running state.

ADDITIONAL INFORMATION

Thread scheduler is responsible for the thread scheduling.

When the multiple threads are waiting to get the chance of execution then in which order thread will be executed is decided by thread scheduler.

5.9 THREAD PRIORITIES

Every Java thread has a priority.

Thead priority helps the thread scheduler to determine the order in which threads are scheduled. Thread priority is used to decide when to switch from one running thread to another thread.

Java priorities are integer values in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

S.No.	Thread Priorities	Values
1.	public static final int MAX_PRIORITY	10
2.	public static final int MIN_PRIORITY	1
3.	public static final int NORMAL_PRIORITY	5

The following instance methods are used for understanding priorities of threads.

1. **setPriority() :** To set the priority of any thread we have to use predefined method setPriority().

   java.lang.Thread.setPriority() method changes the priority of thread to the value priority. This method throws IllegalArgumentException if value of parameter priority goes beyond minimum(1) and maximum(10) limit.The syntax of above method is given below.

   ```
   public final void setPriority(int priority)
   ```

2. **getPriority() :** To get the the priority of current running thread we have to use method getPriority().

   java.lang.Thread.getPriority() method returns priority of given thread. The syntax of above method is given below.

   ```
   public final int getPriority()
   ```

3. **setName() :** This method is used to set the new name to a thread. The syntax of above method is given below.

   ```
   public final void setName(String name)
   ```

4. **getName() :** is used to get the name of a thread. The syntax of above method is given below.

```
public final String getName()
currentThread()
```

5. The **currentThread()** method of thread class is used to return a reference to the currently executing thread object. The syntax of above method is given below.

```
public static Thread.currentThread().
It returns the currently executing thread.
```

EXAMPLE

Write a java program to illustrate priorities of threads.

```
// Java Program to Illustrate Priorities with help of getPriority() and setPriority()
// Importing required classes
importjava.lang.*;
class ThreadDemo  extends Thread
{
    public void run()
    {
        System.out.println("Inside run method");
    }
    public static void main(String[] args)
    {
        // Creating random threads
        ThreadDemo  t1 = new ThreadDemo();
        ThreadDemo  t2 = new ThreadDemo();
        ThreadDemo  t3 = new ThreadDemo();
        // Display the priority of thread using getPriority()
System.out.println("t1 thread priority"+t1.getPriority());
        // Display the priority of thread using getPriority()
System.out.println("t2 thread priority"+ t2.getPriority());
        // Display the priority of thread using getPriority()
System.out.println("t3 thread priority"+t3.getPriority());
```



```
        // Setting priorities of above threads
        t1.setPriority(2);
        t2.setPriority(5);
        t3.setPriority(8);
        // t3.setPriority(21); //throwIllegalArgumentException
System.out.println("t1 thread priority"+t1.getPriority());
System.out.println("t2 thread priority"+ t2.getPriority());
System.out.println("t3 thread priority"+t3.getPriority());
System.out.println("Currently Executing Thread : "
            + Thread.currentThread().getName());
        System.out.println("Main thread priority : "
            + Thread.currentThread().getPriority());
        // Main thread priority is set to 10
        Thread.currentThread().setPriority(10);
        System.out.println("Main thread priority : "
            + Thread.currentThread().getPriority());
    }
}
```

Output :

```
t1 thread priority : 5
t2 thread priority : 5
t3 thread priority : 5
t1 thread priority : 2
t2 thread priority : 5
t3 thread priority : 8
Currently Executing Thread : main
Main thread priority : 5
Main thread priority : 10
```

Output explanation:

- Thread with the highest priority will get an execution chance prior to other threads. Suppose there are 3 threads t1, t2, and t3 with priorities 4, 6, and 1. So, thread t2 will execute first based on maximum priority 6 after that t1 will execute and then t3.

The default priority for the main thread is always 5, it can be changed later. The default priority for all other threads depends on the priority of the parent thread.

EXAMPLE-1

Write a java program using thread priorities.

```
class TestMultiPriority1 extends Thread
{
 public void run()
{
//System.out.println("thread name is:"+Thread.currentThread().getName());
 System.out.println("thread priority is:"+Thread.currentThread().getPriority());
 }
 public static void main(String args[])
{
 TestMultiPriority1 m1=new TestMultiPriority1();
 TestMultiPriority1 m2=new TestMultiPriority1();
 m1.setPriority(1);      //Thread.MIN_PRIORITY);
 m2.setPriority(10);     //Thread.MAX_PRIORITY);
 m1.start();
 m2.start();
 }
}
```

Output :

```
thread priority is:1
thread priority is:10
```

EXAMPLE-2

Write a java program for naming a thread.

```
class TestMultiNaming extends Thread
{
 public void run()
 {
   System.out.println("running...");
```



```
 }
 public static void main(String args[])
 {
 TestMultiNaming   t1=new TestMultiNaming();
 TestMultiNaming   t2=new TestMultiNaming();
 System.out.println("Name of t1:"+t1.getName());
 System.out.println("Name of t2:"+t2.getName());
  t1.start();
  t2.start();
  t1.setName("first");
 System.out.println("After changing name of t1:"+t1.getName());
 t2.setName("second");
 System.out.println("After changing name of t2:"+t2.getName());
 }
}
```

Output :

```
Name of t1:Thread-0
Name of t2:Thread-1
After changing name of t1:first
After changing name of t2:second
running...
running...
```

5.10 PROCESS OF CREATING THREAD USING THREAD CLASS AND RUNNABLE INTERFACE

NUMBER OF WAYS TO CREATE A THREAD

"Thread" class is super class for all newly created threads. It is a predefined class in java.

Thread is created in two ways. They are :

- Thread creation by extending Thread class.
- Thread creation by implementing Runnable Interface.

Method-1 : Thread creation by extending Thread class.

- One way of creating a thread is to create a new class that extends the Thread class.

- We must need to give the definition of run() method.
- This run() method is the entry point for the thread and thread will be alive till run method finishes its execution.
- Basically start() method calls run() method implicitly for the execution of a thread.

Thread is a super class for all newly created threads.

Syntax ofthread creation by extending Thread class is given below.

```
class name_of_thread extends Thread
{
    void run()
    {
        ——
        Statements ;
        ——
    }
}
```

EXAMPLE-1

Write a java program to create multiple threads using Thread class.

```
class Multi extends Thread
{
public void run()
{
for(int i=1;i<10;i++)
System.out.println("thread is running..."+i);
}
public static void main(String args[])
{
        Multi t1=new Multi();
        t1.start();
Multi t2=new Multi();
        t2.start();
  }
}
```


When newly created class is extending the "Thread" class then class object will be considered as a thread object.

Note : start() internally invokes run() method.

void run():run() is used for providing the logic of the thread /task of the thread.

Method-2 : Thread creation by implementing Runnable Interface.

- The second way of creating a thread is to create a class that implements the Runnable interface. We need to give the definition of run() method.
- This run() is the entry point for the thread and thread will be alive till run method finishes its execution.
- Basically start() method calls run() method implicitly.

Syntax ofthread creation by implementing Runnable Interface is given below.

```
class name_of_thread implements Runnable
{
void run()
{
——
Statements ;
——
}
}
```

EXAMPLE-2

Write a java program to create multiple threads using Runnableinterface.

```
class Multi_imp implements Runnable
{
public void run()
{
   for(int i=1;i<10;i++)
   System.out.println("thread is running...");
}
public static void main(String args[])
{
```

```
Multi_imp  m1=new Multi_imp();
Thread t1 =new Thread(m1);
Multi_imp  m2=new Multi_imp();
Thread t2=new Thread(m2);
t1.start();
t2.start();
 }
}
```

When newly created class is not extending the "Thread" class then class object will not be considered as a thread object.

So you need to explicitly create Thread class object. We have to pass the object of newly created class that implements Runnable as a parameter to "Thread"class, so that our class run() method will be executed.

ADDITIONAL INFORMATION

How to perform single task by multiple threads ?

If you want to perform single task by many threads, then you have to include only one run() method but you have to create multiple thread objects.

For Example : Program of performing single task by multiple threads

First Method : by extending Thread class

```
class TestMultitasking1 extends Thread
{
 public void run()
 {
   System.out.println("task one");
 }
 public static void main(String args[])
 {
   TestMultitasking1 t1=new TestMultitasking1();
   TestMultitasking1 t2=new TestMultitasking1();
   t1.start();
```



```
   t2.start();
 }
}
```

Output :

```
task one
task one
```

Second Method : by implementing Runnable Interface.

```
class TestMultitasking2 implements Runnable
{
 public void run()
 {
 System.out.println("task one");
 }
 public static void main(String args[])
 {
    TestMultitasking2 t1=new TestMultitasking2();
    Thread m1=new Thread(t1);
    TestMultitasking2 t2=new TestMultitasking2();
    Thread m2=new Thread(t2);
    m1.start();
    m2.start();
  }
}
```

Output :

```
task one
task one
```

5.11 CREATION OF MULTIPLE THREADS

Creating more than one thread and executing them simultaneously is called multi threading.

How to perform multiple tasks by multiple threads (multitasking in multithreading)?

If you want to perform multiple tasks by multiple threads then we have to include multiple run() methods.

Program of Performing Multiple (two) Tasks by Multiple (two) Threads :

First Method : by extending Thread class.

```
class Simple1 extends Thread
{
 public void run()
 {
   System.out.println("task one");
 }
}
class Simple2 extends Thread
{
 public void run()
 {
   System.out.println("task two");
 }
}
class TestMultitasking3
{
public static void main(String args[])
{
  Simple1 t1=new Simple1();
  Simple2 t2=new Simple2();
  t1.start();
  t2.start();
 }
}
```

Output :

```
task one
task two
```


Second Method : By implementing Runnable Interface.

```
class Simple1 implements Runnable
{
public void run()
 {
   System.out.println("task one");
 }
}
class Simple2 implements Runnable
{
 public void run()
 {
   System.out.println("task two");
 }
}
 class TestMultitasking4
 {
  public static void main(String args[])
  {
    Simple1 t1=new Simple1();
    Thread m1=new Thread(t1);
    Simple2 t2=new Simple2();
    Thread m2=new Thread(t2);
    m1.start();
    m2.start();
  }
 }
```

Output :

```
task one
task two
```

5.12 CONCEPT OF SYNCHRONIZATION

Synchronization in java is the capability to control the access of multiple threads to any shared resource.

Synchronization means a thread must wait for another thread to enter critical section (or) to leave critical section.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

What is need of synchronization ? The synchronization is mainly used to :

1. To prevent thread interference.
2. To prevent consistency problem.

THREAD SYNCHRONIZATION

There are two types of thread synchronization. They are mutual exclusive and inter-thread communication.

1. **Mutual Exclusive :**
   - Synchronized method.
   - Synchronized block.
   - Static synchronization.
2. **Cooperation (Inter-thread communication in java) :** Mutual Exclusive.

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java :

(i) By synchronized method.

(ii) By synchronized block.

(iii) By static synchronization.

CONCEPT OF LOCK IN JAVA

Synchronization is built around an internal entity known as the lock (or) monitor. Every object has an lock associated with it. Whenever a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Understanding the problem without Synchronization In this example, there is no synchronization, so output is inconsistent.

Let's see the example.

```
class Table
{
void printTable(int n)  //method not synchronized
{
  for(int i=1;i<=10;i++)
  {
    System.out.println(n*i);
     try
     {
       Thread.sleep(400);
     }
catch(Exception e)
{
System.out.println(e);
}
  }
}
}
class MyThread1 extends Thread
{
   Table t;
   MyThread1(Table t)
   {
       this.t=t;
   }
    public void run()
    {
        t.printTable(5);
    }
}
```

```
class MyThread2 extends Thread
{
   Table t;
   MyThread2(Table t)
   {
      this.t=t;
   }
   public void run()
   {
     t.printTable(100);
   }
}
class TestSynchronization1
{
  public static void main(String args[])
 {
  Table obj = new Table();                    //only one object
  MyThread1 t1=new MyThread1(obj);
  MyThread2 t2=new MyThread2(obj);
  t1.start();
  t2.start();
 }
}
```

Output :

```
5   100  10   200  15   300  20   400  25   500  30   600  35   700  40
800  45  900  50  1000
```

Synchronized is the modifier applicable only for methods and blocks but not for classes and variables. If multiple threads are trying to access simultaneously on the same java object then there may be a chance of data inconsistency problem.

To overcome this problem we should use "synchronized" keyword before method name.

If a method declared as synchronized then at a time only one thread is allowed to execute that method on the given object so that data inconsistency problem will be resolved.

Advantage :

- Data inconsistency problem is resolved.

Disadvantages :

- It increases waiting time of threads and creates performance problems, hence if there is no specific requirement then it is not recommended to use "synchronized" keyword.
- Internally synchronization is implemented by using lock. Every object in java has a unique lock.
- If a thread wants to execute synchronized method on the given object first it has to get "lock" of object.
- Once thread got lock then it is allowed to execute any synchronized method on that object.
- After method execution is completed automatically thread releases the lock.
- Acquiring the lock and releasing the lock internally performed by JVM.

JAVA SYNCHRONIZED METHOD

If you declare any method as synchronized, it is known as *synchronized method.*

In method synchronization which method you want to synchronize that will be declared as synchronized method with keyword synchronized.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

//Example of java synchronized method

```
class Table
{
 synchronized void printTable(int n)        //synchronized method
 {
   for(int i=1;i<=5;i++)
   {
     System.out.println(n*i);
     try
```

```
{
    Thread.sleep(400);
   }catch(Exception e){System.out.println(e);}
  }
 }
}
class MyThread1 extends Thread
{
Table t;
MyThread1(Table t)
{
  this.t=t;
}
public void run()
{
  t.printTable(5);
}

}
class MyThread2 extends Thread
{
Table t;
MyThread2(Table t)
{
this.t=t;
}
public void run()
{
  t.printTable(100);
}
}
public class TestSynchronization2
```



```
{
 public static void main(String args[])
 {
  Table obj = new Table();//only one object
  MyThread1 t1=new MyThread1(obj);
  MyThread2 t2=new MyThread2(obj);
  t1.start();
  t2.start();
 }
}
```

Output :

5	10	15	20	25	30	35	40	45	50
100	200	300	400	500	600	700	800	900	1000

5.13 ISALIVE() JOIN() SUSPEND() RESUME() METHODS

The isAlive() method of thread class tests if the thread is alive. A thread is considered alive when the start() method of thread class has been called and the thread is not yet dead. This method returns true if the thread is still running and not finished.

Syntax :

```
public final boolean isAlive()
```

This method returns true if the thread is alive otherwise returns false.

EXAMPLE-1

Write a java program to illustrate isAlive().

```
public class JavaIsAliveExp extends Thread
{
  public void run()
  {
    try
    {
      Thread.sleep(300);
      System.out.println("is run() method isAlive "+Thread.currentThread().isAlive());
    }
```

```
            catch (InterruptedException ie)
        {
            }
        }
        public static void main(String[] args)
        {
            JavaIsAliveExp t1 = new JavaIsAliveExp();
            System.out.println("before starting thread isAlive: "+t1.isAlive());
            t1.start();
            System.out.println("after starting thread isAlive: "+t1.isAlive());
        }
    }
```

Output :

```
before starting thread isAlive: false
after starting thread isAlive: true
is run() method isAlive true
suspend() method
```

The suspend() method of thread class puts the thread from running state to waiting state. This method is used if you want to stop the thread execution and start it again when a certain event occurs. This method allows a thread to temporarily cease execution. The suspended thread can be resumed by using the resume() method.

Syntax :

```
public final void suspend()
This method does not return any value.
Example:Write a java program to illustrate suspend()
public class JavaSuspendExp extends Thread
{
    public void run()
    {
        for(int i=1; i<5; i++)
        {
            try
```

```
2
Thread-2
2
Thread-0
3
Thread-2
3
Thread-0
4
Thread-2
4
resume() method
```

The resume() method of thread class is only used with suspend() method. This method is used to resume a thread which was suspended using suspend() method. This method allows the suspended thread to start again.

Syntax :

```
public final void resume()
```

This method does not return any value.

EXAMPLE-2

Write a java program to illustrate resume().

```
public class JavaResumeExp extends Thread
{
  public void run()
  {
    for(int i=1; i<5; i++)
    {
      try
      {
        // thread to sleep for 500 milliseconds
        sleep(500);
        System.out.println(Thread.currentThread().getName());
      }catch(InterruptedException e)
```



```
{
System.out.println(e);
}
        System.out.println(i);
    }
  }
  public static void main(String args[])
  {
    // creating three threads
    JavaResumeExp t1=new JavaResumeExp ();
    JavaResumeExp t2=new JavaResumeExp ();
    JavaResumeExp t3=new JavaResumeExp ();
    // call run() method
    t1.start();
    t2.start();
    t2.suspend(); // suspend t2 thread
    // call run() method
    t3.start();
    t2.resume(); // resume t2 thread
  }
}
```

Output :

```
Thread-0
1
Thread-2
1
Thread-1
1
Thread-0
2
Thread-2
2
```

```
Thread-1
2
Thread-0
3
Thread-2
3
Thread-1
3
Thread-0
4
Thread-2
4
Thread-1
4
join() method
```

The join() method of thread class waits for a thread to die. It is used when you want one thread to wait for completion of another.

Syntax :

1. public final void join()throws InterruptedException
2. public final void join(long millis)throwsInterruptedException
3. public final void join(long millis, int nanos)throws InterruptedException

Parameter

1. **millis :** It defines the time to wait in milliseconds
2. **nanos :** 0-999999 additional nanoseconds to wait

Return : It does not return any value.

Exception

IllegalArgumentException : This exception is thrown if the value of millis is negative, (or) the value of nanos is not in the range 0-999999.

InterruptedException : This exception is thrown if any thread has interrupted the current thread.

EXAMPLE-3

Write a java program to illustrate join().

```
public class JoinExample1 extends Thread
{
   public void run()
   {
      for(int i=1; i<=4; i++)
      {
         try
         {
            Thread.sleep(500);
         }catch(Exception e){System.out.println(e);}
         System.out.println(i);
      }
   }
   public static void main(String args[])
   {
      // creating three threads
      JoinExample1 t1 = new JoinExample1();
      JoinExample1 t2 = new JoinExample1();
      JoinExample1 t3 = new JoinExample1();
      // thread t1 starts
      t1.start();
      // starts second thread when first thread t1 is died.
      try
      {
         t1.join();
      }catch(Exception e)
       {
         System.out.println(e);
       }
      // start t2 and t3 thread
      t2.start();
```

```
        t3.start();
    }
}
```

Output :

```
1  2  3  4  1  1  2  2  3
3  4  4
```

In the above example 1, when t1 completes its task then t2 and t3 starts execution.

EXAMPLE-4

```
public class JoinExample2 extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            try
            {
                Thread.sleep(500);
            }catch(Exception e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        // creating three threads
        JoinExample2 t1 = new JoinExample2();
        JoinExample2 t2 = new JoinExample2();
        JoinExample2 t3 = new JoinExample2();
        // thread t1 starts
        t1.start();
        // starts second thread when first thread t1 is died.
        try
```



```
        {
            t1.join(1500);
        }catch(Exception e){System.out.println(e);}
        // start t2 and t3 thread
        t2.start();
        t3.start();
    }
}
```

Output :

```
1  2  3  1  1  4  2  2  5
3  3  4  4  5  5
```

In the above example 2, when t1 is completes its task for 1500 milliseconds(3 times) then t2 and t3 starts executing.

5.14 INTER THREAD COMMUNICATION

Inter-thread communication (or) Co-operation is all about allowing synchronized threads to communicate with each other.

Inter thread communication is important when you want to develop an application where two (or) more threads exchange some information.

- Two threads can communicate each other by using wait(),notify() notifyAll().
- wait(),notify()/notifyAll() are present in Object class but not in Thread class.
- We have to call wait(),notify()/notifyAll() from synchronized methods, otherwise run time exception i.e., illegalMonitorStateException is generated.

`void wait()`

Causes the current threadto wait until another thread invokes the notify() method (or) the notifyAll() method for this object.

`void wait(long timeout)`

Causes the current thread to wait until either another thread invokes the notify() method (or) the notifyAll() method for this object (or) a specified amount of time has elapsed.

`void wait(long timeout, int nanos)`

Causes the current thread to wait until either another thread invokes the notify() method (or) the notifyAll() method for this object (or) some other thread interrupts the current thread, (or) a certain amount of real time has elapsed.

Syntax of wait():

```
public final void wait()throws InterruptedException
public final void wait(long timeout)throws InterruptedException
public final void wait(long timeout,int nanos)throws InterruptedException
```

notify():wakes up a single thread that is waiting on this objects monitor.

Syntax of notify() is given below.

```
public final void notify()
```

notifyAll():

Wakes up all threads that are waiting on this object's monitor. Syntax of notify()All is given below.

```
public final void notifyAll()
```

If a thread calls wait() on any object, it immediately releases lock of that particular object and enters into waiting state.

If a thread calls notify() on any object it wakes up the thread which is waiting on object monitor and releases lock of that particular object but may not immediately.

Understanding the process of inter-thread communication.

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() (or) notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

EXAMPLE :

Write a java program to illustrate inter thread communication in java.

```
class Customer
{
int balance=10000;
```



```
synchronized void withdraw(int amount)
{
System.out.println("going to withdraw...");
if(balance<amount)
{
System.out.println("Less balance; waiting for deposit...");
try
{
wait();
}
catch(Exception e){}
}
  balance= balance-amount;
  System.out.println("withdraw completed...");
}
synchronized void deposit(int amount)
{
  System.out.println("going to deposit...");
 balance= balance+amount;
  System.out.println("deposit completed... ");
  notify();
}
}
class Test
{
public static void main(String args[])
{
final Customer c=new Customer();
new Thread()
{
public void run()
{
c.withdraw(15000);
```

```
}
}.start();
new Thread()
{
public void run()
{
c.deposit(10000);
}
}.start();
 }
}
```

Output :

```
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed
```

5.15 DEAD LOCK WITH EXAMPLE PROGRAMS

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

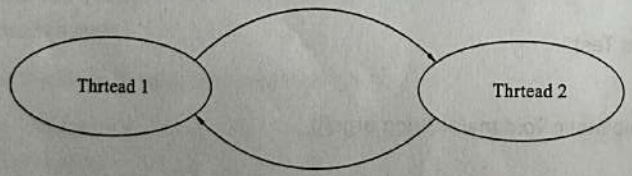


FIG 5.4 :

EXAMPLE-1

Write a java program to illustrate concept of Deadlock in java.

```
class A
{
```



```
public synchronized void d1(B b)
{
System.out.println("thread1 starts execution of d1()");
try{
Thread.sleep(6000);
}
catch(InterruptedException e){}
System.out.println("thread1 trying to call b.last()");
b.last();
}
public synchronized void last()
{
System.out.println("Inside A,this is last() method");
}
}
class B
{
public synchronized void d2(A a)
{
System.out.println("thread2 starts execution of d2()");
try{
Thread.sleep(6000);
}
catch(Exception e){}
System.out.println("thread2 trying to call a.last()");
a.last();
}
public synchronized void last()
{
System.out.println("Inside B,this is last()method");
}
}
class deadlock1 extends Thread
```

```
{
 A a= new A();
 B b= new B();
  public void m1()
 {
  this.start();
   a.d1(b);
 }
public void run()
 {
 b.d2(a);
}
public static void main(String []args)
 {
deadlock1 d=new deadlock1();
d.m1();
 }
}
```

Output :

```
thread1 starts execution of d1()
thread2 starts execution of d2()
thread2 trying to call a.last()
thread1 trying to call b.last()
```

EXAMPLE-2

Write a java program to illustrate concept of Deadlock in java.

```
public class TestDeadlockExample1
{
 public static void main(String[] args)
 {
   final String resource1 = "first";
   final String resource2 = "second";
```



```
// t1 tries to lock resource1 then resource2
Thread t1 = new Thread()
{
 public void run()
 {
   synchronized (resource1)
   {
    System.out.println("Thread 1: locked resource 1");
    try {
                Thread.sleep(1000);
        } catch(Exception e) {}
    synchronized (resource2)
    {
     System.out.println("Thread 1: locked resource 2");
    }
   }
 }
};
// t2 tries to lock resource2 then resource1
Thread t2 = new Thread()
{
 public void run()
 {
  synchronized (resource2)
  {
   System.out.println("Thread 2: locked resource 2");
   try
   {
      Thread.sleep(1000);
   } catch (Exception e) {}
   synchronized(resource1)
   {
    System.out.println("Thread 2: locked resource 1");
```

```
            }
          }
        }
      };
      t1.start();
      t2.start();
    }
  }
```

Output :

```
Thread1:locked resource 1
Thread2:locked resource 2
```

REVIEW QUESTIONS

1 Mark Questions

1. Define compile-time error.
2. What is runtime error ?
3. What is logical error ?
4. What is an exception ?
5. What is exception handling ?
6. List types of exceptions in java.
7. Define built-in exception.
8. Define checked exception.
9. Define Un-checked exception.
10. List any two examples of checked exceptions.
11. List any two examples of Un-checked exceptions.
12. Define thread in Java.
13. What is multithreading.
14. List states in life cycle of thread.

15. List the various thread priorities.
16. What is synchronization ?
17. Define inter-thread communication.
18. Define deadlock.

3 Marks Questions

1. Define compile-time error, run-time error and linker error.
2. Write any three differences between checked exceptions and unchecked exceptions.
3. Draw the hierarchy of java Exception classes.
4. List the keywords used in exception handling.
5. Write the syntax of exception handling block.
6. Write the syntax of multi-catch statement.
7. Write the syntax of nested try statement.
8. Write any three differences between throw and throws.
9. Draw the life cycle of a thread.
10. List the states of life cycle of a thread.
11. Write purpose of isAlive(),suspend0 and resume().
12. List advantages of exception handling.
13. List advantages of multithreading.
14. List any three differences between error and exception.

5 Marks Questions

1. Explain different types of errors.
2. Define an exception and explain different types of exceptions ?
3. Explain about exception-handling block.
4. Write a java program to handle exceptions.
5. Explain finally statement with an example program.
6. Write a java program to illustrate the use of multi-catch statements.

7. Write a java program using Nested-try statements.
8. Write a Java program using throw and throws clauses.
9. Write a java program to create user-defined exceptions.
10. Explain life cycle of a thread.
11. Explain thread priorities with an example program.
12. Explain creating thread using Thread class with suitable example.
13. Explain creating thread using Runnable interface with suitable example.
14. Write a java program for creating multiple threads.
15. Explain the concept of synchronization with an example program.
16. Write a java program to illustrate isAlive().
17. Write a java program to illustrate suspend().
18. Write a java program to illustrate resume().
19. Write a java program to illustrate join().
20. Explain inter-thread communication with example program.
21. Explain how to handle deadlocks with example program.

CHAPTER

6

JDBC AND SERVLETS

CHAPTER OUTLINE

6.1 ABOUT JDBC AND UNDERSTAND JDBC ARCHITECTURE

Java Database Connectivity (JDBC) is an application programming interface (API) between java programming language and database.

(or)

JDBC is an standard API specification developed in order to move data from frontend to backend. This API consists of classes and interfaces written in Java. It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

Why JDBC came into existence ?

In earlier days, the front-end applications are connected to the Databases using the functions provided by the database vendors. For example, the C and C++ applications are connected to the databases using a set of functions given by Oracle Corporation called ***orcl.h header file.*** But, by this, the application becomes database dependent because every DB vendor gives its own set of functions for communication. To overcome this, Microsoft with Simba Technologies has provided us with the ODBC (Open Database Connectivity) community with which we can connect and communicate with Database in an independent manner.

Why is ODBC not used in Java Applications ?

ODBC API is written in C Language with pointers. But Java applications do not contain pointers, so the Java code is being converted to pointers code internally which is time-consuming and poor in performance. Also, ODBC is platform-dependent and database-independent. In order to overcome this problem, Sun Microsystems introduced the JDBC technology to make the Java programs platform and database independent.

JDBC is a Java API which offers a natural Java interface for working with SQL.

Advantages of JDBC Architecture :

1. It can read any database. The only condition for it to do so is that all of the drivers be properly installed.
2. It pulls information from a database and converts it to XML.
3. It does not necessitate the conversion of the content.
4. Software maintenance is centralized with no client settings necessary. Because the driver is built in Java, the JDBC URL (or) a DataSource object has all of the information required to establish a connection.

5. It supports queries and stored procedures completely.
6. The JDBC API contains a DataSource object that can be used to identify and connect to a data source. This improves the codes portability and maintainability.
7. Both synchronous and asynchronous processing is supported.
8. The Java API and the JDBC API work together to make application development simple and cost-effective.
9. Modules are supported.
10. Even if data is housed on various database management systems, businesses can continue to use their installed databases and access information.

JDBC ARCHITECTURE

JDBC supports two types of processing models for accessing database i.e. two-tier and three-tier.

1. **Two-tier Architecture :** This architecture helps java program (or) application to directly communicate with the database. It needs a JDBC driver to communicate with a specific database. Query (or) request is sent by the user to the database and results are received back by the user. The database may be present on the same machine (or) any remote machine connected via a network. This approach is called ***client-server architecture*** (or) configuration.

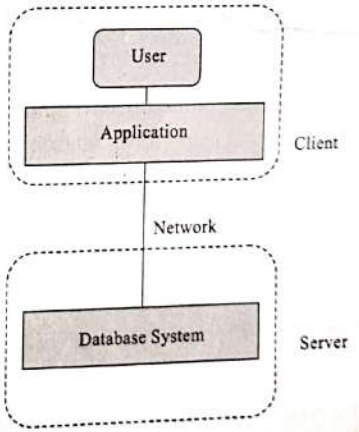


FIG 6.1 : Two-tier Architecture

2. **Three-tier Architecture :** In this, there is no direct communication. Requests are sent to the middle tier i.e. HTML browser sends a request to java application which is then further sent to the database. Database processes the request and sends the result back to the middle tier which then communicates with the user. It increases the performance and simplifies the application deployment.

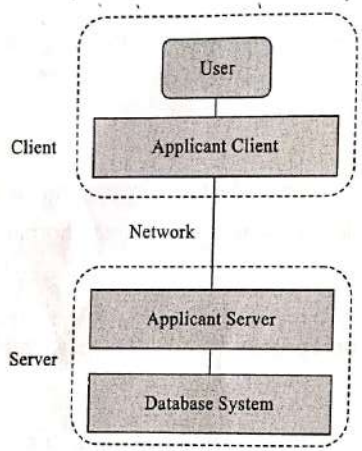


FIG 6.2 : Three-tier Architecture

Components of JDBC Architecture : The components of JDBC Architecture are explained below.

1. **Driver Manager:** It is a class that contains a list of all drivers. When a connection request is received, it matches the request with the appropriate database driver using a protocol called ***communication sub-protocol.*** The driver that matches is used to establish a connection.
2. **Driver :** It is an interface which controls the communication with the database server. DriverManager objects are used to perform communication.
3. **Connection :** It is an interface which contains methods to contact a database.
4. **Statement :** This interface creates an object to submit SQL queries or statements to the database.
5. **ResultSet:** This contains the results retrieved after the execution of the SQL statements (or) queries.
6. **SQLException :** Any errors that occur in database application are handled by this class.

Basic JDBC architectural diagram is shown below with the positioning of all components :

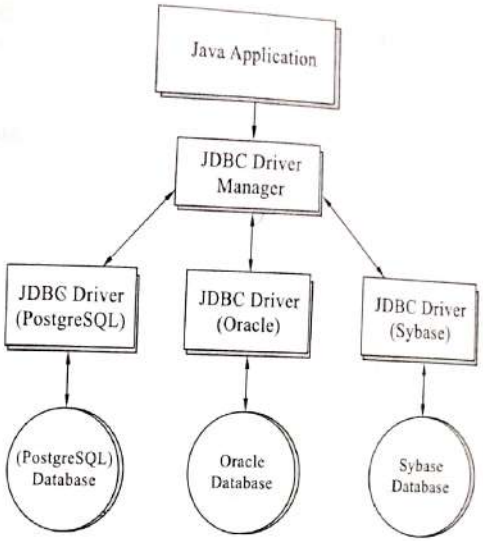


FIG 6.3 :

INTERFACES

The java.sql package consists of many interfaces. Some popular interfaces are mentioned below.

1. **Driver Interface :** This interface allows for multiple database drivers. DriverManager objects are created to communicate with the database. These objects are created by DriverManager.registerDriver();
2. **Connection Interface :** Connection interface establishes the connection i.e. session between java program and the database. It has many methods like rollback(), close() etc.,
3. **Statement Interface :** This interface provides methods for the execution of the SQL queries. It provides factory methods to get a ResultSet object. Some methods of statement interface are executeQuery(), executeUpdate() etc.,
4. **PreparedStatement Interface :** This interface helps when the SQL queries need to implement many times. It accepts input parameters during runtime.
5. **CallableStatement Interface :** This interface is used when stored procedures are to be accessed. It also accepts parameters during run time.
6. **ResultSet Interface :** This interface helps to store the result returned after the execution of the SQL queries.

TYPES OF JDBC DRIVERS

JDBC drivers are classified into four types. There are.

1. **Type-1 Driver (or) JDBC-ODBC Bridge :** Type-1 driver or JDBC-ODBC bridge-driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called ***Universal driver*** because it can be used to connect to any of the databases.

   As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.

   The ODBC bridge driver is needed to be installed in individual client machines.

   Type-1 driver isnt written in java, thats why it isnt a portable driver.

   This driver acts as a bridge between JDBC and ODBC. It is easy to use but execution time is slow.

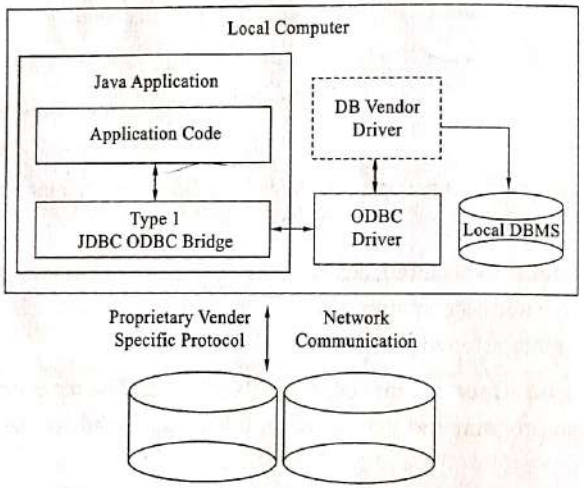


FIG 6.4 : Database Server

2. **Type-2 Driver (or) Native API Partly Java Driver :** The Native API driver uses the client-side libraries of the database. This driver converts JDBC method calls into native calls of the database API. Driver needs to be installed separately in individual client machines. The Vendor client library needs to be installed on client machine.

   Type-2 driver isnt written in java, thats why it isnt a portable driver.

   It is comparatively faster than Type-1 driver but it requires native library and cost of application also increases.

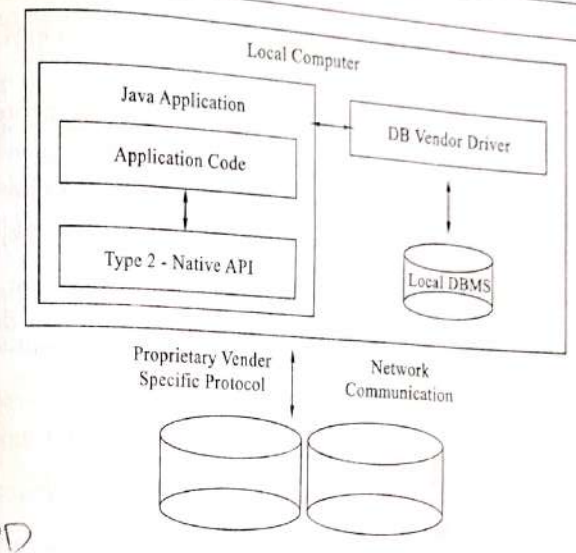


FIG 6.5 : Database Server

3. **Type-3 Driver or Network Protocol Driver :** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, Hence no need of individual client-side installation. Type-3 drivers are fully written in Java, hence they are portable drivers.

   No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.Network support is required on client machine.

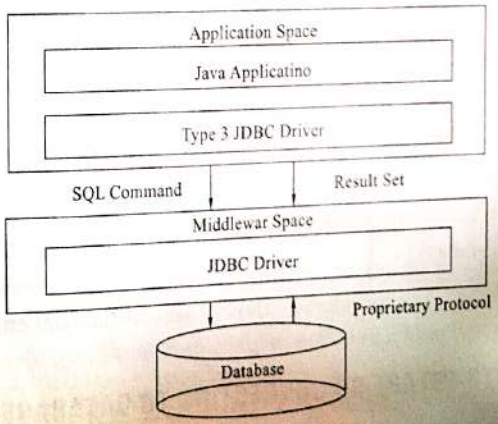


FIG 6.6 : Type 3 JDBC Net Driver

Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4. **Type-4 (or) Thin Driver :** Type-4 driver is also called ***native protocol driver.*** This driver interacts directly with database. It does not require any native database library that is why it is also known as ***thin driver/pure java driver.*** It does not require any native library and middleware server, so no client-side (or) server-side installation.It is fully written in Java language, hence they are portable drivers.

   It has better performance than other drivers but comparatively slow due to an increase in a number of network calls.

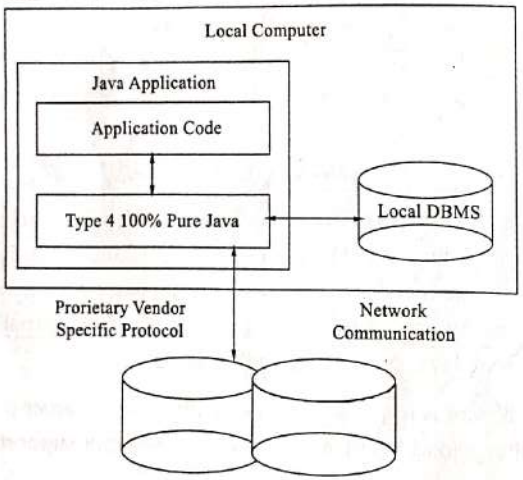


FIG 6.7 : Database Server

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is type-4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

6.2 HOW TO ESTABLISH CONNECTION TO DATABASE

The below steps are to be followed to establish a connection between java program and database.

1. Import the database.
2. Load the drivers using the forName() method/Register the drivers using DriverManager.
3. Establish a connection using the Connection class object.
4. Create a statement.
5. Execute the query.
6. Close the connections.

Let us discuss these steps in brief before implementing by writing suitable code to illustrate connectivity steps for JDBC.

Step-1 : Import the database.

This is for making the JDBC API classes available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used.

```
import java.sql.*;
```

Additionally, The following packages might need to be imported while using the Oracle extensions to JDBC such as using advanced data types such as BLOB, and so on.

```
import oracle.jdbc.driver.*;
import oracle.sql.*;
```

Step-2 : Load the drivers/Register the drivers.

You first need load the driver or register it before using it in the program. You can use following two ways to Load the drivers/Register the drivers.

Class.forName() : Here we load the drivers class file into memory at the runtime. No need of using new or creation of object .The following example uses Class.forName() to load the Oracle driver.

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

DriverManager.registerDriver() : DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time. The following example uses DriverManager.registerDriver()to register the Oracle driver.

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

Step 3 : Establish a connection using the Connection class object.

After loading the driver, establish connections via as shown below as follows :

Connection con =DriverManager.getConnection(url,user,password) :

(i) ***user :*** Username from which your SQL command prompt can be accessed.

(ii) ***password :*** password from which the SQL command prompt can be accessed.

(iii) ***con*** : It is a reference to the Connection interface.

(iv) ***Url :*** Uniform Resource Locator which is created as shown below :

```
String url = " jdbc:oracle:thin:@localhost:1521:xe"
```

Where oracle is the database used, thin is the driver used, @localhost is the IP Address where a database is stored, 1521 is the port number and xe is the service provider. All 3 parameters above are of String type and are to be declared by the programmer before calling the function.

Step-4 : Create a statement.

Once a connection is established, you can interact with the database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database. Use of JDBC Statement is as follows.

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

Note : Here, con is a reference to Connection interface used in previous step.

Step 5 : Execute the query.

Once a Statement object has been constructed, the next step is to execute the query. This is done by using the executeQuery() method of the Statement object. A call to this method takes a SQL SELECT statement as parameter and returns a JDBC ResultSet object. The following line of code illustrates this using the st object created above.

```
ResultSetrset = st.executeQuery
("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");
```

Alternatively, the SQL statement can be placed in a string and then this string passed to the executeQuery() function. This is shown below.

String sql = "SELECT empno, ename, sal, deptno FROM emp ORDER BY ename";

```
ResultSetrset = st.executeQuery(sql);
```


The above statement executes the SELECT statement specified in between the double quotes and stores the resulting rows in an instance of the ResultSet object named rset.

Step-6 : Retrieving and Processing the Results of a Database Query.

Once the query has been executed, there are two steps to be carried out.

- Processing the output resultset to fetch the rows.
- Retrieving the column values of the current row.

The first step is done using the next() method of the ResultSet object. A call to next() is executed in a loop to fetch the rows one row at a time, with each call to next() advancing the control to the next available row. The next() method returns the Boolean value true while rows are still available for fetching and returns false when all the rows have been fetched.

The second step is done by using the getXXX() methods of the JDBC rset object. Here getXXX() corresponds to the getInt(), getString() etc with XXX being replaced by a Java datatype.

The following code demonstrates the above steps :

```
while (rset.next())
 {
System.out.println(str = rset.getInt(1)+ " "+ rset.getString(2)+ " "+rset.getFloat(3)+
" "rset.getInt(4));
 }
```

Here the 1, 2, 3, and 4 in rset.getInt(), rset.getString(), getFloat(), and getInt() respectively denote the position of the columns in the SELECT statement.

Step-7 : Closing the connections.

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the close() method of the Connection class.

By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection. It is as shown below as follows.

```
con.close();
```

6.3 IMPLEMENT SIMPLE APPLICATION AND EXECUTE QUERY

To connect and create a table in the MySQL database through java program by using JDBC, we need to install MySQL Sever.

In Java program, to establish connection with the database, we need hostname (Server name, in case of same system we use localhost with database name, port no, database username and database password.

Here in this example, we are using following details to connect to the database :

Hostname : localhost
Database name : demo
Port number : 3306
Username : root
Password : 123

Now, we need to create an object of Connection class and connect to the database by using above given details using DriverManager.getConnection() method.

Then, we need to create an object of Statement class to prepare MySQL query to be executed. To create an object of Statement class we use :

```
Statement smt=cn.createStatement();
```

Here, Statement is class name, smt is the object name, cn is the object of Connection class and createStatement() is the method which initialize the object of statement class.

After, preparing a query we need to execute it by using executeUpdate() method, which is a method of Statement class.

```
// Java program to create a table
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class CreateTable
{
    public static void main(String[] args)
        {
            try
{
```



```
Class.forName("com.mysql.jdbc.Driver").newInstance();
//serverhost = localhost, port=3306, username=root, password=123
Connection cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/
demo","root","123");
Statement smt=cn.createStatement();
//query to create table Employees with fields
name(empid,empname,dob,city,salary)
String q="create table Employees(empidvarchar(10) primary
key,empnamevarchar(45),dob date,cityvarchar(45),salary varchar(45))";
//to execute the update
smt.executeUpdate(q);
System.out.println("Table Created....");
cn.close();
        }
catch(Exception e)
            {
                System.out.println(e.getMessage());
            }
    }
}
```

Output :

```
Table Created...
```

```
// Java Program to Establish Connection and insert values in database
// Importing database
Import java.sql.*;
// Importing required classes
Import java.util.*;
class Test
{
    public static void main(String a[])
    {
        // Creating the connection using Oracle DB ,url syntax is standard
```

```
String url = "jdbc:oracle:thin:@localhost:1521:xe";
 // Usernamer and password to access DB
String user = "system";
String pass = "12345";
// Entering the data
Scanner k = newScanner(System.in);
 System.out.println("Enter the name");
String name = k.next();
 System.out.println("Enter the roll number");
introll = k.nextInt();
 System.out.println("Enter the class");
String cls = k.next();
 // Inserting data using SQL query
String sql = "insert into student1 values('"+ name+ "',"+ roll + ",'"+ cls + "')";
 // Connection class object
 Connection con = null;
 // Try block to check for exceptions
 try{
    // Registering drivers
    DriverManager.registerDriver(neworacle.jdbc.OracleDriver());
     // Reference to connection interface
    con = DriverManager.getConnection(url,user,pass);
    // Creating a statement
    Statement st = con.createStatement();
     // Executing query
    intm = st.executeUpdate(sql);
    if(m == 1)
       System.out.println(
          "inserted successfully : "+ sql);
    else
       System.out.println("insertion failed");
    // Closing the connections
    con.close();
 }
```



```
    // Catch block to handle exceptions
    catch(Exception ex)
 {
       // Display message when exceptions occurs
       System.err.println(ex);
    }
  }
}
```

Output :

```
Enter the name
RAJU
Enter the roll number
11
Enter the class
6A
Inserted successfully : insert into values(RAJU,11,6A)
```

6.4 DIFFERENT STATEMENTS USED IN JDBC

The statement interface is used to create SQL basic statements in java it provides methods to execute queries with the database. There are different types of statements that are used in JDBC as follows :

1. Create Statement.
2. Prepared Statement.
3. Callable Statement.

1. **Create a Statement :** From the connection interface, you can create the object for this interface. It is generally used for generalpurpose access to databases and is useful while using static SQL statements at runtime.

Syntax :

```
Statement statement = connection.createStatement();
```

Implementation : Once the Statement object is created, there are three ways to execute it.

(a) ***boolean execute(String SQL) :*** If the ResultSet object is retrieved, then it returns true else false is returned. Is used to execute SQL DDL statements (or) for dynamic SQL.

(b) ***intexecuteUpdate(String SQL)*** : Returns number of rows that are affected by the execution of the statement, used when you need a number for INSERT, DELETE or UPDATE statements.

(c) ***ResultSetexecuteQuery(String SQL)*** : Returns a ResultSet object. Used similarly as SELECT is used in SQL.

Java Program illustrating Create Statement in JDBC.

```
// Importing Database(SQL) classes
importjava.sql.*;
class Test
{
  public static void main(String[] args)
  {
    try {
      // Loading driver using forName() method
      Class.forName("com.mysql.cj.jdbc.Driver");
      // Registering driver using DriverManager
      Connection con=DriverManager.getConnection("jdbc:mysql:///world", "root",
        "12345");
      // Create a statement
      Statement statement = con.createStatement();
      String sql = "select * from people";
      // Execute the query
      ResultSet result = statement.executeQuery(sql);
      // Process the results
      while (result.next())
      {
        System.out.println("Name: " + result.getString("name"));
        System.out.println("Age:" + result.getString("age"));
      }
    }
    catch (SQLException e)
    {
      System.out.println(e);
```



```
    }
    // Catching generic ClassNotFoundException if any
    catch (ClassNotFoundException e)
    {
      // Print and display the line number where exception occurred
      e.printStackTrace();
    }
  }
}
```

Output :

```
Name: ASYA
Age:5
Name: NIYA
Age:3
```

2. **Prepared Statement** : Represents a recompiled SQL statement, that can be executed many times. This accepts parameterized SQL queries. In this, "?" is used instead of the parameter, one can pass the parameter dynamically by using the methods of PREPARED STATEMENT at run time.

Illustration : Considering in the people database if there is a need to INSERT some values, SQL statements such as these are used:

```
INSERT INTO people VALUES ("Ayan",25);
INSERT INTO people VALUES("Kriya",32);
```

To do the same in Java, one may use Prepared Statements and set the values in the ?holders, setXXX() of a prepared statement is used as shown:

```
String query = "INSERT INTO people(name, age)VALUES(?, ?)";
Statement pstmt = con.prepareStatement(query);
pstmt.setString(1,"Ayan");
ptstmt.setInt(2,25);
// where pstmt is an object name
```

Implementation : Once the PreparedStatement object is created, there are three ways to execute it.

(a) ***execute():*** This returns a boolean value and executes a static SQL statement that is present in the prepared statement object.

(b) ***executeQuery()*** : Returns a ResultSet from the current prepared statement.

(c) ***executeUpdate()*** : Returns the number of rows affected by the DML statements such as INSERT, DELETE, and more that is present in the current Prepared Statement.

EXAMPLE-1

```
// Java Program illustrating Prepared Statement in JDBC
 // Step 1: Importing DB(SQL here) classes
import java.sql.*;
// Importing Scanner class to
// take input from the user
import java.util.Scanner;
class Test
{
   public static void main(String[] args)
   {
      try
      {   // Step 2: Establish a connection
          // Step 3: Load and register drivers
           // Loading drivers using forName() method
          Class.forName("com.mysql.cj.jdbc.Driver");
           // Scanner class to take input from user
          Scanner sc = new Scanner(System.in);
          // Display message for ease for user
          System.out.println("What age do you want to search?? ");
         // Reading age an primitive datatype from user using nextInt()
          int age = sc.nextInt();
           // Registering drivers using DriverManager
    Connection con = DriverManager.getConnection("jdbc:mysql:///world
", "root", "12345");
          // Step 4: Create a statement
          PreparedStatement ps = con.prepareStatement(
             "select name from world.people where age = ?");
```



```
          // Step 5: Execute the query
          ps.setInt(1, age);
          ResultSet result = ps.executeQuery();
          // Step 6: Process the results
          // Condition check using next() methodto check for element
          while (result.next())
    {
              System.out.println("Name : "+ result.getString(1));
          }
          // Step 7: Closing the connections
          }
        // Catch block to handle database exceptions
       catch (SQLException e)
    {
          // Display the DB exception if any
          System.out.println(e);
       }
       // Catch block to handle class exceptions
       catch (ClassNotFoundException e)
    {
          // Print the line number where exception occurred
          // using printStackTrace() method if any
          e.printStackTrace();
       }
     }
}
```

Output :

```
What age do you want to search??
5
Name:ASYA
```

3. **Callable Statement :** Are stored procedures which are a group of statements that we compile in the database for some task, they are beneficial when we are dealing with

multiple tables with complex scenario and rather than sending multiple queries to the database, we can send the required data to the stored procedure & lower the logic executed in the database server itself. The Callable Statement interface provided by JDBC API helps in executing stored procedures.

Syntax : To prepare a CallableStatement

```
CallableStatement cstmt = con.prepareCall("{call Procedure_name(?, ?}");
```

Implementation : Once the callable statement object is created

- execute()is used to perform the execution of the statement.

EXAMPLE-2

```
// Java Program illustrating Callable Statement in JDBC
 // Step 1: Importing DB(SQL) classes
import java.sql.*;
class Test
{
   public static void main(String[] args)
   {
       try {
         // Step 2: Establish a connection
         // Step 3: Loading and registering drivers
         // Loading driver using forName() method
        Class.forName("com.mysql.cj.jdbc.Driver");
         // Registering driver using DriverManager
        Connection con = DriverManager.getConnection(
          "jdbc:mysql:///world
", "root", "12345");
         // Step 4: Create a statement
        Statement s = con.createStatement();
         // Step 5: Execute the query
        // select * from people
         CallableStatement cs  =con.prepareCall("{call peopleinfo(?,?)}");
        cs.setString(1, "Bob");
        cs.setInt(2, 64);
```



```
        cs.execute();
        ResultSet result= s.executeQuery("select * from people");
       // Step 6: Process the results
        // Condition check using next() methodto check for element
        while (result.next())
{
            System.out.println("Name : "+ result.getString(1));
            System.out.println("Age : " + result.getInt(2));
        }
      }
       catch (SQLException e)
{
        // Print the exception
        System.out.println(e);
      }
     // Catch block for generic class exceptions
      catch (ClassNotFoundException e)
{
       // Print the line number where exception occurred
        e.printStackTrace();
}
   }
```

6.5 RESULTSET

1. **ResultSet Interface :**
   - The result of the query after execution of database statement is returned as table of data according to rows and columns. This data is accessed using the ResultSet interface.
   - A default ResultSet object is not updatable and the cursor moves only in forward direction.

CREATING RESULTSET INTERFACE

To execute a Statement or PreparedStatement, we create ResultSet object.

EXAMPLE-1

```
Statement stmt = connection.createStatement();
ResultSet result = stmt.executeQuery("select * from Students");
   (or)
String sql = "select * from Students";
PreparedStatementstmt = con.prepareStatement(sql);
ResultSet result = stmt.executeQuery();
```

ResultSet Interface Methods :

S.No.	Methods	Description
1.	public boolean absolute(int row)	Moves the cursor to the specified row in the ResultSet object.
2.	public void beforeFirst( )	It moves the cursor just before the first row i.e. front of the ResultSet.
3.	public void afterLast()	Moves the cursor to the end of the ResultSet object, just after the last row.
4.	public boolean first()	Moves the cursor to first value of ResultSet object.
5.	public boolean last( )	Moves the cursor to the last row of the ResultSet object.
6.	public boolean previous ( )	Just moves the cursor to the previous row in the ResultSet object.
7.	public boolean next( )	It moves the curser forward one row from its current position.
8.	public intgetInt(intcolIndex)	It retrieves the value of the column in current row as int in given ResultSet object.
9.	public String getString(intcolIndex)	It retrieves the value of the column in current row as int in given ResultSet object.
10.	public void relative(int rows)	It moves the cursor to a relative number of rows.

Types of ResultSet Interface :

1. **ResultSet.TYPE_FORWARD_ONLY :** TheResultSet can only be navigating forward.
2. **ResultSet.TYPE_SCROLL_INSENSITIVE :** TheResultSet can be navigated both in forward and backward direction. It can also jump from current position to another position. The ResultSet is not sensitive to change made by others.
3. **ResultSet.TYPE_SCROLL_SENSITIVE :** TheResultSet can be navigated in both forward and backward direction. It can also jump from current position to another position. The ResultSet is sensitive to change made by others to the database.

EXAMPLE-2

Program to illustrate ResultSet interface with scrollable.

```
import java.sql.*;
class ResultSetTest
{
   public static void main(String args[])
{
   Connection con = null;
   Statement stmt = null;
   ResultSet rs = null;
try
 {
   Class.forName("oracle.jdbc.driver.OracleDriver");
   con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
   "scott","tiger");
   stmt = con.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
        rs = stmt.executeQuery("Select * from Student");
        rs.absolute(5);    //Accessing 5th row
        System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
        rs.close();
        stmt.close();
        con.close();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
   }
}
```

ResultSetMetaData Interface : ResultSetMetaData is an interface which provides information about a result set that is returned by an executeQuery() method. The ResultSetMetaData extends the Wrapper interface. This interface provides the metadata

about the database. It includes the information about the names of the columns, number of columns etc.,

ResultSetMetaData Interface Methods :

S.No.	Methods	Description
1.	public String getColumnClassName(int column)	It returns the name of the Java class whose instances are created.
2.	public intgetColumnCount() ResultSet object.	It returns the number of columns in the ResultSet oject.
3.	public String getColumnName(int column)	Returns the column name from the ResutlSet object.
4.	public intgetColumnType(int column)	It retrieves the column s which is designated in SQL.
5.	public String getSchemaName(int column)	Return the table s schema which designed with column.
6.	public String getTableName(int column)	Returns the designed SQL table s name.

EXAMPLE-3

Illustrating the ResultSetMetaData Interface.

```
import java.sql.*;
class RSMDTest
{
   public static void main(String args[])
   {
      try
      {
         Class.forName("oracle.jdbc.driver.OracleDriver");
         Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "scott", "tiger");
         PreparedStatement ps = con.prepareStatement("select * from Students");
         ResultSet rs = ps.executeQuery();
         ResultSetMetaData rsmd = rs.getMetaData();
         for (int i = 1; i <= cols; i++)
         {
```



```
            String colName = rsmd.getColumnName(i);
            String colType = rsmd.getColumnTypeName(i);
            System.out.println(colName+" of type "+colType);
         }
         rs.close();
         ps.close();
         con.close();
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

DATABASEMETADATA INTERFACE

The DatabaseMetaData is an interface that tells us the type of driver we are using, database product version, driver name, total number of table etc. It also provides all details about database providers.

DatabaseMetaData Interface Methods :

S.No.	Methods	Description
1.	public Connection getConnection( )	It retrieves the connection that produced the given metadata object.
2.	public intgetDatabaseMajorVersion()	Returns the major version number of the database.
3.	public intgetDatabaseMinorVersion()	Returns the minor version number of the database.
4.	public String getDatabaseProductName( )	Used to retrieve the name of this database product.
5.	public String getDriverName( )	Retrieves the name of the JDBC driver which is used in application.
6.	public String getURL( )	It returns the URL of the current DBMS.
7.	public Connection getConnection( )	It retrieves the connection that produced the given metadata object.

EXAMPLE-4

Different methods of DatabaseMetaData Interface.

```
import java.sql.*;
class DatabaseMetaDataDemo
{
   public static void main(String args[])
   {
      Connection conn = null;
      try
      {
         Class.forName("oracle.jdbc.driver.OracleDriver");
         conn =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
DatabaseMetaData dbmd = conn.getMetaData();
System.out.println("Driver name "+dbmd.getDriverName());
System.out.println("Driver Version "+dbmd.getDriverVersion());
System.out.println("DatabaseProduct name+dbmd.getDatabaseProductName());
System.out.println("Database productversion+dbmd.getDatabaseProductVersion());
System.out.println("Username "+dbmd.getUserName());
Sysetm.out.println("URL of database "+dbmd.getURL());
}
catch(SQLException e)
{
   e.printStackTrace();
}
finally
{
if(conn != null)
{
try
{
conn.close();
```



```
}
catch(SQLException e)
{
e.printStackTrace();
}
}
}
}
}
```

6.6 DDL AND DML PROGRAMS USING JDBC

These basic DDL and DML operations are INSERT, SELECT, UPDATE and DELETE statements in SQL language.

Different types of DDL and DML programs using JDBC are given below.

CONNECTING TO THE DATABASE

The Oracle Database server listens on the default port 1521 at localhost. The following code snippet connects to the database name userid by the user login1 and password pwd1.

```
// Java program to illustrate connecting to the Database
import java.sql.*;
public class connect
{
public static void main(String args[])
   {
try
      {
Class.forName("oracle.jdbc.driver.OracleDriver");
         // Establishing Connection
         Connection con = DriverManager.getConnection(
         "jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
if (con != null)
System.out.println("Connected");
```

```
else
System.out.println("Not Connected");
con.close();
    }
catch(Exception e)
    {
System.out.println(e);
    }
  }
}
```

Output :

```
Connected
```

Note : Here oracle in database URL in getConnection() method specifies SID of Oracle Database. For Oracle database 11g it is orcl and for oracle database 10 g it is xe.

IMPLEMENTING INSERT STATEMENT

Write a java program to illustrate inserting to the Database.

```
import java.sql.*;
public class insert1
{
public static void main(String args[])
  {
     String id = "id1";
     String pwd = "pwd1";
     String fullname = " koti";
     String email = "koti@gmail.com";
 try
     {
 Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection("
jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
        Statement stmt = con.createStatement();
        String q1 = "insert into useridvalues('" +id+ "', '" +pwd+
                    "', '" +fullname+ "', '" +email+ "')";
```



```
int x = stmt.executeUpdate(q1);
if (x > 0)
System.out.println("Successfully Inserted");
else
System.out.println("Insert Failed");
con.close();
    }
catch(Exception e)
    {
System.out.println(e);
    }
  }
}
```

Output :

```
Successfully Inserted
```

IMPLEMENTING UPDATE STATEMENT

Write a java program to illustrate updating the Database.

```
import java.sql.*;
public class update1
{
public static void main(String args[])
  {
     String id = "id1";
     String pwd = "pwd1";
     String newPwd = "newpwd";
 try
     {
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("
jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
        Statement stmt = con.createStatement();
        String q1 = "UPDATE userid set pwd = '" + newPwd +
```

```
" WHERE id = '" +id+ "' AND pwd = '" + pwd + "'";
int x = stmt.executeUpdate(q1);
if (x > 0)
System.out.println("Password Successfully Updated");
else
System.out.println("ERROR OCCURED :(");
con.close();
    }
catch(Exception e)
    {
System.out.println(e);
    }
  }
}
```

Output :

```
Password Successfully Updated
```

IMPLEMENTING DELETE STATEMENT

Write a java program to illustrate deleting from Database.

```
import java.sql.*;
public class delete
{
public static void main(String args[])
  {
    String id = "id2";
    String pwd = "pwd2";
try
    {
Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection("
jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
        Statement stmt = con.createStatement();
```



```
        // Deleting from database
        String q1 = "DELETE from userid WHERE id = '" + id +
            "' AND pwd = '" + pwd + "'";
int x = stmt.executeUpdate(q1);
if (x > 0)
System.out.println("One User Successfully Deleted");
else
System.out.println("ERROR OCCURED :(");
con.close();
    }
catch(Exception e)
    {
System.out.println(e);
    }
  }
}
```

Output :

```
One User Successfully Deleted
```

IMPLEMENTING SELECT STATEMENT

Write a java program to illustrate selecting from Database.

```
import java.sql.*;
public class select
{
public static void main(String args[])
  {
    String id = "id1";
    String pwd = "pwd1";
try
    {
Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection("
jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
```

```
        Statement stmt = con.createStatement();
        String q1 = "select * from userid WHERE id = '" + id +
        "' AND pwd = '" + pwd + "'";
ResultSet rs = stmt.executeQuery(q1);
if (rs.next())
        {
System.out.println("User-Id : " + rs.getString(1));
System.out.println("Full Name :" + rs.getString(3));
System.out.println("E-mail :" + rs.getString(4));
        }
else
        {
System.out.println("No such user id is already registered");
        }
con.close();
    }
catch(Exception e)
    {
System.out.println(e);
    }
  }
}
```

Output :

```
User-Id : id1
Full Name :koti
E-mail :koti@gmail.com
```

6.7 SERVLET AND EXPLAIN THE LIFE CYCLE OF SERVLET

Servlets are the Java programs that run on the Java-enabled web server (or) application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Properties of Servlets are as follows :

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the webserver.

Servlet Architecture is can be depicted from the image itself as provided below as follows :

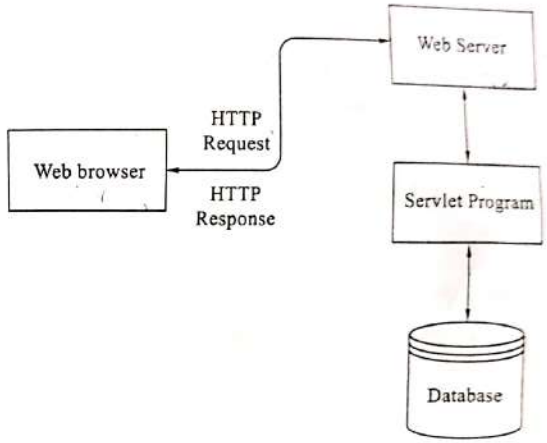


FIG 6.8 :

Execution/Working of Servlets basically involves six basic steps :

1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Now let us do discuss why do we need For Server-Side extensions ?

The server-side extensions are nothing but the technologies that are used to create dynamic Web pages. Actually, to provide the facility of dynamic Web pages, Web pages need a container (or) Web server. To meet this requirement, independent Web server providers offer some proprietary solutions in the form of APIs(Application Programming Interface). These APIs allow us to build programs that can run with a Web server. In

this case, Java Servlet is also one of the component APIs of Java Platform Enterprise Edition which sets standards for creating dynamic Web applications in Java.

The Servlet technology is similar to other Web server extensions such as Common Gateway Interface(CGI) scripts and Hypertext Preprocessor (PHP). However, Java Servlets are more acceptable since they solve the limitations of CGI such as low performance and low degree scalability.

What is CGI ?

CGI is actually an external application that is written by using the programming languages like C (or) C++ and this is responsible for processing client requests and generating dynamic content.

In CGI application, when a client makes a request to access dynamic Web pages, the Web server performs the following operations.

- It first locates the requested web page *i.e* the required CGI application using URL.
- It then creates a new process to service the clients request.
- Invokes the CGI application within the process and passes the request information to the application.
- Collects the response from the CGI application.
- Destroys the process, prepares the HTTP response, and sends it to the client.

Common Gateway Interface (CG) Processing A Clients Request :

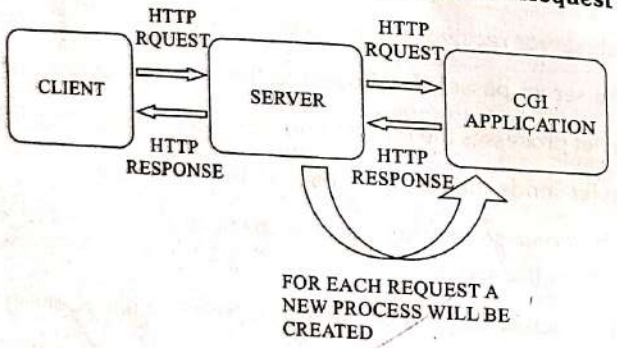


FIG 6.9 :

So, CGI server has to create and destroy the process for every request. Its easy to understand that this approach is applicable for handling few clients but as the number of clients increases, the workload on the server increases and so the time is taken to process requests increases.

Difference Between Servlet and CGI :

S.No.	Servlet	CGI (Comma Gateway Interface)
1.	Servlets are portable and efficient.	CGI is not portable.
2.	In Servlets, sharing data is possible.	In CGI, sharing data is not possible.
3.	Servlets can directly communicate with the webserver.	CGI cannot directly communicate with the webserver.
4.	Servlets are less expensive than CGI.	CGI is more expensive than Servlets.
5.	Servlets can handle the cookies.	CGI cannot handle the cookies.

SERVLETS APIS

Servlets are build from two packages :

- javax.servlet(Basic).
- javax.servlet.http(Advance).

Various classes and interfaces present in these packages are :

S.No.	Component	Type	Package
1.	Servlet	Interface	javax.servlet.*
2.	ServletRequest	Interface	javax.servlet.*
3.	ServletResponse	Interface	javax.servlet.*
4.	GenericServlet	Class	javax.servlet.*
5.	HttpServlet	Class	javax.servlet.http.*
6.	HttpServletRequest	Interface	javax.servlet.http.*
7.	HttpServletResponse	Interface	javax.servlet.http.*
8.	Filter	Interface	javax.servlet.*
9.	ServletConfig	Interface	javax.servlet.*

Advantages of a Java Servlet :

- Servlet is faster than CGI as it doesnt involve the creation of a new process for every new request received.
- Servlets, as written in Java, are platform-independent.
- Removes the overhead of creating a new process for each request as Servlet doesnt run in a separate process. There is only a single instance that handles all requests concurrently. This also saves the memory and allows a Servlet to easily manage the client state.

- It is a server-side component, so Servlet inherits the security provided by the Web server.
- The API designed for Java Servlet automatically acquires the advantages of the Java platforms such as platform-independent and portability. In addition, it obviously can use the wide range of APIs created on Java platforms such as JDBC to access the database.
- Many Web servers that are suitable for personal use or low-traffic websites are offered for free or at extremely cheap costs eg. Java servlet. However, the majority of commercial-grade Web servers are rather expensive, with the notable exception of Apache, which is free.

THE SERVLET CONTAINER

Servlet container, also known as ***servlet engine*** is an integrated set of objects that provide a run time environment for Java Servlet components.

In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

Services Provided by the Servlet Container :

1. **Network Services :** Loads a Servlet class. The loading may be from a local file system, a remote file system (or) other network services. The Servlet container provides the network services over which the request and response are sent.
2. **Decode and Encode MIME-Based Messages :** Provides the service of decoding and encoding MIME-based messages.
3. **Manage Servlet Container :** Manages the lifecycle of a Servlet.
4. **Resource Management** : Manages the static and dynamic resources, such as HTML files, Servlets, and JSP pages.
5. **Security Service :** Handles authorization and authentication of resource access.
6. **Session Management :** Maintains a session by appending a **session ID** to the URL path.

SERVLETS LIFE CYCLE

- The life cycle is the process from the construction till the destruction of any object.
- A servlet also follows a certain life cycle.
- The life cycle of the servlet is managed by the servlet container.

The container performs the following steps :

1. Loads Servlet Class.
2. Creates instance of Servlet.
3. Calls init( ) method.
4. Calls service( ) method.
5. Calls destroy( ) method.

Architecture Diagram of Servlet is given below.

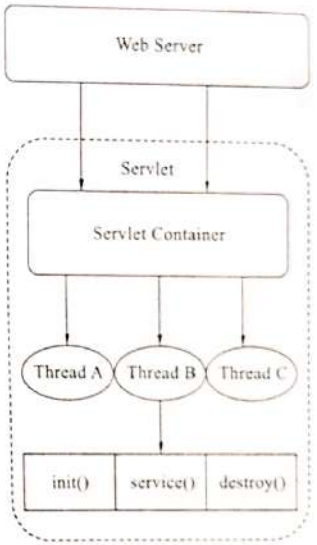


FIG 6.10 : Servlet Life Cycle

1. **Loads the Servlet Class :** The **classloader** is responsible to load the Servlet class into the container. The servlet class is loaded when the first request comes from the web container.
2. **Creates instance of Servlet :** After loading of Servlet class into the container, the web container creates its instance. The instance of Servlet class is created only once in the servlet life cycle.
3. **Calls the init( ) method :** The **init( )** method performs some specific action constructor of a class. It is automatically called by ***Servlet container.*** It is called only once through the ***completeservlet life cycle.***

Syntax :

```
public void init(ServletConfigconfig ) throws ServletException
```

4. **Calls the service( ) method :** The **service( )** method performs actual task of servlet container. The web container is responsible to call the service method each time the request for servlet is received. The service( ) invokes the **doGet( ), doPost( ), doPut ( ), doDelete( )** methods based on the HTTP request.

Syntax :

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

5. **Calls the destroy( ) method :** The **destroy( )** method is executed when the servlet container remove the servlet from the container. It is called only once at the end of the ***life cycle of servlet.***

Syntax :

```
public void destroy( )
```

6.8 JAVA SERVLET DEVELOPMENT KIT

JSDK is the Java Servlet Development Kit. This is an add on to the regular JDK (Java Developers Kit). The JSDK has the additional files needed in order to compile Java servlets. The latest version of the JSDK is version 2.0. Included in the JSDK is a Java Servlet Runner program. The Servlet Runner is a program that runs on your workstation, and allows you to test servlets you have written without running a web server. Other files included are the several Java Servlet examples. Included are the .java and .class files for testing purposes and to help you understand how the Java code is implemented. Another important file that is included is the jsdk.jar file. This file includes the class information necessary to compile the servlets.

INSTALLING JSDK

Let me preface the JSDK installation procedures by saying that the JSDK is an add on to the JDK. If you are trying to compile servlets using JSDK, you will be missing software. The JDK includes the javac.exe program and other java library and class files that are necessary to compile java code. Before you start to install the JSDK, install the JDK, then begin JSDK install.

Installing the JSDK is a very simple procedure. When you run the .exe that you download from the http://java.sun.com web site, it will create some temporary web files on your hard drive. It will then launch the installation script. You will be asked to specify the

directory you wish to have the files copied to. As in the JDK, the recommended directory includes a 'period' in the directory name (ex. C:\JSDK 20). Change this to C:\JSDK20 instead to make it easier when searching for the directory. After the software is installed, you will have a new subdirectory. In order to use both the JDK and JSDK together, the java compiler (javac.exe) needs to know where the class files are located. I found the easiest way to make this work, is to first put the bin directory for the JDK in the path. This will make it easy to find the javac.exe program when compiling code. Second, add the jsdk.jar file to the classpath. This can be done by adding a SET statement to the autoexec.bat file on your workstation (for Windows 95). The SET statement should read SET CLASSPATH = *drive:JSDK install path*\lib\jsdk.jar (ex. SET CLASSPATH = C:\jsdk20\lib\jsdk.jar). Once this is done, you will have no problem compiling your java servlets.

6.9 JAVAX.SERVLET PACKAGE AND SIMPLE SERVLET

The javax.servlet package and javax.servlet package contain a number interfacesand classes for servlet API. The javax.servlet package package contains a number interfaces and classes that are used by servlet container. The javax.servlet package contains a number interfaces and classes that are responsible for http requests only.

INTERFACES IN JAVAX.SERVLET PACKAGE

The javax.servlet package contains a number interfaces and they are given below.

S.No.	Interface	Description
1.	AsyncContext	Class representing the execution context for an asynchronous operation that was initiated on a ServletRequest.
2.	AsyncListener	Listener that will be notified in the event that an asynchronous operation initiated on a ServletRequest to which the listener had been added has completed, timed out, or resulted in an error.
3.	Filter	A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), (or) on the response from a resource, (or) both.
4.	FilterChain	A FilterChain is an object provided by the servlet container to the developer giving a view into the invocation chain of a filtered request for a resource.
5.	FilterConfig	A filter configuration object used by a servlet container to pass information to a filter during initialization.
6.	FilterRegistration	Interface through which a Filter may be further configured.
7.	FilterRegistration.Dynamic	Interface through which a Filter registered via one of the addFilter methods on ServletContext may be further configured.

8.	ReadListener	This class represents a call-back mechanism that will notify implementations as HTTP request data becomes available to be read without blocking.
9.	Registration	Interface through which a Servlet or Filter may be further configured.
10.	Registration.Dynamic	Interface through which a Servlet or Filter registered via one of the addServlet or addFilter methods, respectively, on ServletContext may be further configured.
11.	RequestDispatcher	Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server.
12.	Servlet	Defines methods that all servlets must implement.
13.	ServletConfig	A servlet configuration object used by a servlet container to pass information to a servlet during initialization.
14.	ServletContainerInitializer	Interface which allows a library/runtime to be notified of a web application's startup phase and perform any required programmatic registration of servlets, filters, and listeners in response to it.
15.	ServletContext	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, (or) write to a log file.
16.	ServletContextAttribute Listener	Interface for receiving notification events about ServletContext attribute changes.
17.	ServletContextListener	Interface for receiving notification events about ServletContext lifecycle changes.
18.	ServletRegistration	Interface through which a Servlet may be further configured.
19.	ServletRegistration. Dynamic	Interface through which a Servlet registered via one of the addServlet methods on ServletContext may be further configured.
20.	ServletRequest	Defines an object to provide client request information to a servlet.
21.	ServletRequestAttribute Listener	Interface for receiving notification events about ServletRequest attribute changes.
22.	ServletRequestListener	Interface for receiving notification events about requests coming into and going out of scope of a web application.
23.	ServletResponse	Defines an object to assist a servlet in sending a response to the client.
24.	SessionCookieConfig	Class that may be used to configure various properties of cookies used for session tracking purposes.

25.	SingleThreadModel	Deprecated As of Java Servlet API 2.4, with no direct replacement.
26.	WriteListener	Callback notification mechanism that signals to the developer it's possible to write content without blocking.

CLASSES IN JAVAX.SERVLET PACKAGE

The javax.servlet package contains a number classes and they are given below.

S.No.	Class	Description
1.	AsyncEvent	Event that gets fired when the asynchronous operation initiated on a ServletRequest (via a call to ServletRequest.startAsync() (or) ServletRequest.startAsync(ServletRequest, ServletResponse) has completed, timed out, or produced an error.
2.	GenericFilter	Defines a generic, protocol-independent filter.
3.	GenericServlet	Defines a generic, protocol-independent servlet.
4.	HttpConstraintElement	Java Class representation of an HttpConstraint annotation value.
5.	HttpMethodConstraint Element	Java Class represntation of an HttpMethodConstraint annotation value.
6.	MultipartConfigElement	Java Class represntation of an MultipartConfig annotation value.
7.	ServletContextAttribute Event	Event class for notifications about changes to the attributes of the ServletContext of a web application.
8.	ServletContextEvent	This is the event class for notifications about changes to the servlet context of a web application.
9.	ServletInputStream	Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time.
10.	ServletOutputStream	Provides an output stream for sending binary data to the client.
11.	ServletRequestAttribute Event	This is the event class for notifications of changes to the attributes of the servlet request in an application.
12.	ServletRequestEvent	Events of this kind indicate lifecycle events for a ServletRequest.
13.	ServletRequestWrapper	Provides a convenient implementation of the ServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet.
14.	ServletResponseWrapper	Provides a convenient implementation of the ServletResponse interface that can be subclassed by developers wishing to adapt the response from a Servlet.
15.	ServletSecurityElement	Java Class representation of a ServletSecurity annotation value.

INTERFACES IN JAVAX.SERVLET.HTTP PACKAGE

The javax.servlet.http package contains a number interfacesand they are given below.

S.No.	Interface	Description
1.	HttpServletMapping	Allows runtime discovery of the manner in which the HttpServlet for the current HttpServletRequest was invoked.
2.	HttpServletRequest	Extends the ServletRequest interface to provide request information for HTTP servlets.
3.	HttpServletResponse	Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response.
4.	HttpSession	Provides a way to identify a user across more than one page request (or) visit to a Web site and to store information about that user.
5.	HttpSessionActivation Listener	Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated.
6.	HttpSessionAttribute Listener	Interface for receiving notification events about HttpSession attribute changes.
7.	HttpSessionBinding Listener	Causes an object to be notified when it is bound to (or) unbound from a session.
8.	HttpSessionContext	Deprecated As of Java(tm) Servlet API 2.1 for security reasons, with no replacement.
9.	HttpSessionIdListener	Interface for receiving notification events about HttpSession id changes.
10.	HttpSessionListener	Interface for receiving notification events about HttpSession lifecycle changes.
11.	HttpUpgradeHandler	This interface encapsulates the upgrade protocol processing.
12.	Part	This class represents a part or form item that was received within a multipart/form-data POST request.
13.	PushBuilder	Build a request to be pushed.
14.	WebConnection	This interface encapsulates the connection for an upgrade request.

CLASSES IN JAVAX.SERVLET.HTTP PACKAGE

The javax.servlet.http package contains a number classesand they are given below.

S.No.	Class	Description
1.	Cookie	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.
2.	HttpFilter	Provides an abstract class to be subclassed to create an HTTP filter suitable for a Web site.
3.	HttpServlet	Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
4.	HttpServletRequest Wrapper	Provides a convenient implementation of the HttpServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet.
5.	HttpServletResponse Wrapper	Provides a convenient implementation of the HttpServlet Response interface that can be subclassed by developers wishing to adapt the response from a Servlet.
6.	HttpSessionBinding Event	Events of this type are either sent to an object that implements HttpSessionBindingListener when it is bound (or) unbound from a session, (or) to a HttpSessionAttributeListener that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.
7.	HttpSessionEvent	This is the class representing event notifications for changes to sessions within a web application.

Steps to Create a Servlet Example :

1. Steps to create the servlet using Tomcat server.
2. Create a directory structure.
3. Create a Servlet.
4. Compile the Servlet.
5. Create a deployment descriptor.
6. Start the server and deploy the application.

There are given 6 steps to create a servlet example. These steps are required for all the servers.

The servlet example can be created by three ways :

1. By implementing Servlet interface,
2. By inheriting GenericServlet class (or)
3. By inheriting HttpServlet class.

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.,

Here, we are going to use apache tomcat server in this example. The steps are as follows :

1. Create a directory structure.
2. Create a Servlet.
3. Compile the Servlet.
4. Create a deployment descriptor.
5. Start the server and deploy the project.
6. Access the servlet.

1. **Create a Directory Structures :** The directory structure defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.

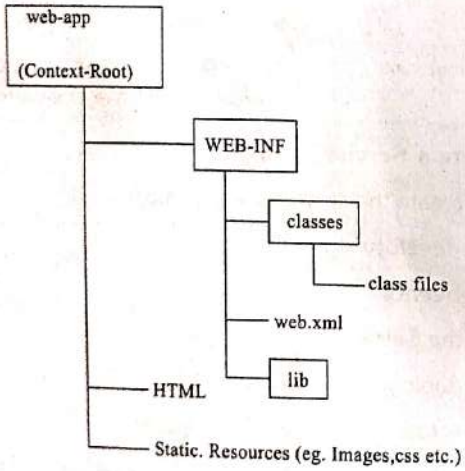


FIG 6.11 :

As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2. **Create a Servlet :** There are three ways to create the servlet.
   - By implementing the Servlet interface.

- By inheriting the GenericServlet class.
- By inheriting the HttpServlet class.
- The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.,

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

```
//DemoServlet.java
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet
{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data
//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");
pw.close();//closing the stream
}
}
```

3. **Compile the Servlet :** For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files.

S.No.	Jar File	Server
1.	servlet-api.jar	Apache Tomcat
2.	weblogic.jar	Weblogic
3.	javaee.jar	Glassfish
4.	javaee.jar	JBoss

Two ways to load the jar file :

(i) set classpath.

(ii) paste the jar file in JRE/lib/ext folder.

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

4. **Create the deployment descriptor (web.xml file) :** The deployment descriptor is an xml file, from which Web Container gets the information about the servet to be invoked.The web container uses the Parser to get the information from the web.xml file.

There are many xml parsers such as SAX, DOM and Pull.There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

```
web.xml file
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

DESCRIPTION OF THE ELEMENTS OF WEB.XML FILE

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows :

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>.This pattern is used at client side to invoke the servlet.

5. **Start the Server and deploy the project :** To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.One Time Configuration for Apache Tomcat Server You need to perform 2 tasks :

(i) set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).

(ii) Change the port number of tomcat.It is required if another server is running on same port (8080).

How to set JAVA_HOME in environment variable ?

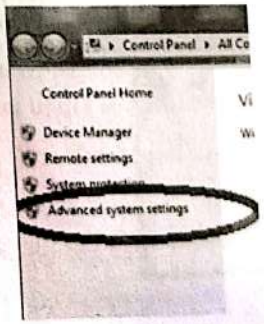
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

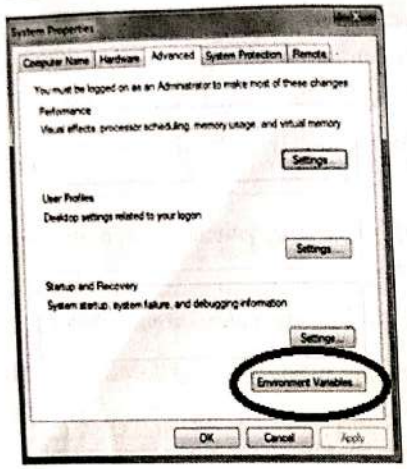
Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path

ofjdk folder in variable value -> ok -> ok -> ok.

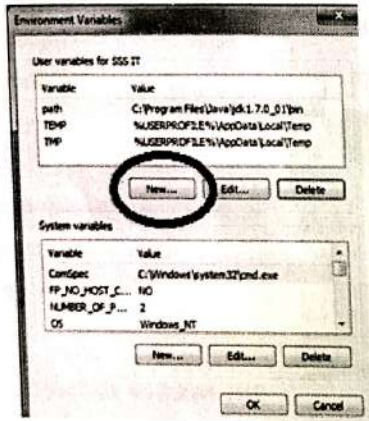
Go to My Computer properties :

Click on advanced system settings tab then environment variables :

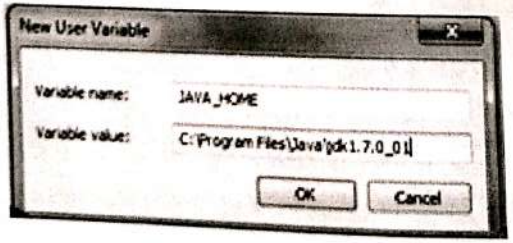




Click on the new tab of user variable or system variable :



Write JAVA_HOME in variable name and paste the path of jdk folder in variable value :



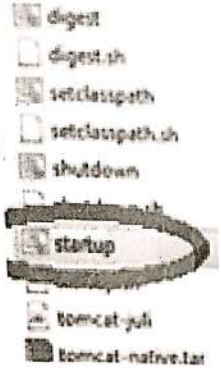
There must not be semicolon (;) at the end of the path.

After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

Note : There are two types of tomcat available :

1. Apache tomcat that needs to extract only (no need to install).
2. Apache tomcat that needs to install.

It is the example of apache tomcat that needs to extract only.



How to change port number of apache tomcat.

Changing the port number is required if there is another server running on the same system with same

portnumber.Suppose you have installed oracle, you need to change the port number of apache tomcat

because both have the default port number 8080.

Open server.xml file in notepad. It is located inside the apache-tomcat/conf directory. Change the

Connector port = 8080 and replace 8080 by any four digit number instead of 8080.

Let us replace it by 9999 and save this file.

6. **How to deploy the servlet project :** Copy the project and paste it in the webapps folder under apache tomcat.

But there are several ways to deploy the project. They are as follows :

- By copying the context(project) folder into the webapps directory.
- By copying the war folder into the webapps directory.
- By selecting the folder path from the server.
- By selecting the war file from the server.

Here, we are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write

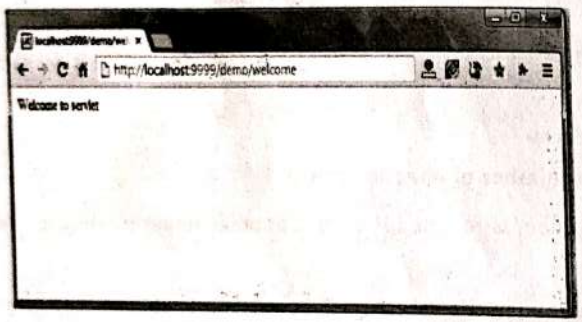
```
projectfolder> jar cvf myproject.war*
```

Creating war file has an advantage that moving the project from one location to another takes less time.

7. **How to access the servlet :**

   Open broser and write http://hostname:portno/contextroot/urlpatternofservlet.

   For example:http://localhost:9999/demo/welcome



6.10 HANDLING HTTP REQUEST AND RESPONSES WITH EXAMPLE PROGRAMS

The HttpServlet class provides specialized methods that handle the various types of HTTP requests. A servlet developer typically overrides one of these methods. These methods are doDelete( ), doGet( ), doHead( ), doOptions( ), doPost( ), doPut( ), and doTrace( ). However, the GET and POST requests are commonly used when handling form input.

Handling HTTP GET Requests.

Here we will develop a servlet that handles an HTTP GET request. The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in ColorGet.html, and a servlet is defined in ColorGetServlet.java. The HTML source code for ColorGet.html is shown in the following listing. It defines a form that contains a select element and a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies a servlet to process the HTTP GET request.


```
<html>
<body>
<center>
<form name="Form1"
action="http://localhost:8080/examples/servlets/servlet/ColorGetServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit"></form>
</body>
</html>
```

The source code for ColorGetServlet.java is shown in the following listing. The doGet() method is overridden to process any HTTP GET requests that are sent to this servlet. It uses the getParameter( ) method of HttpServletRequest to obtain the selection that was made by the user. A response is then formulated.

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;
public class ColorGetServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throwsServletException, IOException
{
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color);
pw.close();
}
}
```

Compile the servlet. Next, copy it to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example :

Start Tomcat, if it is not already running.
Display the web page in a browser.
Select a color.
Submit the web page.

After completing these steps, the browser will display the response that is dynamically generated by the servlet.

One other point: Parameters for an HTTP GET request are included as part of the URL that is sent to the web server. Assume that the user selects the red option and submits the form. The URL sent from the browser to the server is

http://localhost:8080/examples/servlets/servlet/ColorGetServlet?color=Red

The characters to the right of the question mark are known as the ***query string***.

Handling HTTP POST Requests.

Here we will develop a servlet that handles an HTTP POST request. The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in ColorPost.html, and a servlet is defined in ColorPostServlet.java.

The HTML source code for ColorPost.html is shown in the following listing. It is identical to ColorGet.html except that the method parameter for the form tag explicitly specifies that the POST method should be used, and the action parameter for the form tag specifies a different servlet.

```
<html>
<body>
<center>
<form name="Form1" method="post"
action="http://localhost:8080/examples/servlets/servlet/ColorPostServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
```



```
<br><br>
<input type=submit value="Submit"></form>
</body>
</html>
```

The source code for ColorPostServlet.java is shown in the following listing. The doPost() method is overridden to process any HTTP POST requests that are sent to this servlet. It uses the getParameter( ) method of HttpServletRequest to obtain the selection that was made by the user. A response is then formulated.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorPostServlet extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)
throwsServletException, IOException
{
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color);
pw.close();
}
}
```

Compile the servlet and perform the same steps as described in the previous section to test it.

REVIEW QUESTIONS

1 Mark Questions

1. Define JDBC ?
2. List the JDBC drivers.
3. List different types of Statements in JDBC.
4. Write the syntax of create statement.
5. Write the syntax of callable statement.
6. Define servlet.
7. List the components of JDBC architecture.

3 Marks Questions

1. Write any three advantages of JDBC.
2. Write short notes on Type-1 driver.
3. Write short notes on Type-2 driver.
4. Write short notes on Type-3 driver.
5. Write short notes on Type-4 driver.
6. Write short notes on components of JDBC architecture.
7. Why is ODBC not used in Java Applications ?
8. Write the steps to connect any java application with the database using JDBC.
9. Write short notes on create statement.
10. Write short notes on callable statement.
11. Write about Prepared statement.
12. Write the advantages of servlets.
13. Draw the life cycle of servlet.
14. Write any three differences between CGI and servlets.
15. Discuss Java Servlet Development Kit.
16. List any three classes in javax.sevlet package.
17. List any three intertaces in javax.sevlet package.

18. List any three classes in javax.serviet.http package.
19. List any three interfaces in javax.serviet.http package.

5 Marks Questions

1. Explain JDBC Architecture with a diagram.
2. Write the advantages of JDBC architecture.
3. Write short notes on JDBC drivers.
4. Explain the steps to establish a connection to database using JDBC.
5. Write a java program using JDBC to connect to database
6. Explain about Statement, Prepared Statement and Callable Statement.
7. Explain PreparedStatement with an example program.
8. Explain CallableStatement with an example program.
9. Explain about ResultSet in JDBC with an example program.
10. Write a JDBC program to insert records into database.
11. Write a JDBC Program to update a row in a table.
12. Define servlet and explain its life cycle.
13. Explain about creating a simple servlet.
14. Write different classes and interfaces available in javax.servlet package.
15. Write different classes and interfaces available in javax.servlet. http package.
16. Explain the process of handling HTTP request and responses with an example program.