Simple.java

```
class JavaException {
   public static void main(String args[]){
      int d = 0;
      int n = 20;
      int fraction = n/d;
     System.out.println("End Of Main");
   }
}
```

simple.java

```
// Java program to demonstrate how exception is thrown.
class ThrowsExecp{

    public static void main(String args[]){

        String str = null;
        System.out.println(str.length());

    }
}
```

Output :

```
Exception in thread "main" java.lang.NullPointerException
    at ThrowsExecp.main(File.java:8)
```

Exception1.java

```
class Exc2 {
public static void main(String args[]) {
int d, a;
try { // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
} catch (ArithmeticException e) { // catch divide-by-zero error
System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

This program generates the following output:

```
Division by zero.
After catch statement.
```

Exception2.java

```
// Demonstrate multiple catch statements.
class MultiCatch {
public static void main(String args[]) {
try {
int a = args.length;
System.out.println("a = " + a);
int b = 42 / a;
int c[] = { 1 };
c[42] = 99;
} catch(ArithmeticException e) {
System.out.println("Divide by 0: " + e);
} catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array index oob: " + e);
}
System.out.println("After try/catch blocks.");
}
}
```

Exception3.java

```
/* This program contains an error. A subclass must come before its superclass in a series of catch
statements. If not, unreachable code will be created and a compile-time error will result.
*/
class SuperSubCatch {
public static void main(String args[]) {
try {
int a = 0;
int b = 42 / a;
} catch(Exception e) {
System.out.println("Generic Exception catch.");
}
/* This catch is never reached because ArithmeticException is a subclass of Exception. */
catch(ArithmeticException e) { // ERROR - unreachable
System.out.println("This is never reached.");
}
}
}
```

Nestedtry.java

```
// An example of nested try statements.
class NestTry {
public static void main(String args[]) {
try {
int a = args.length;
/* If no command-line args are present, the following statement will generate
a divide-by-zero exception. */
int b = 42 / a;
System.out.println("a = " + a);
try { // nested try block
/* If one command-line arg is used, then a divide-by-zero exception will be generated by the
following code. */
if(a==1) a = a/(a-a); // division by zero
/* If two command-line args are used, then generate an out-of-bounds exception. */
if(a==2) {
int c[] = { 1 };
c[42] = 99; // generate an out-of-bounds exception
}
} catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array index out-of-bounds: " + e);
}
} catch(ArithmeticException e) {
System.out.println("Divide by 0: " + e);
}
}
}
```

Throw.java

```
// Demonstrate throw.
class ThrowDemo {
static void demoproc() {
try {
throw new NullPointerException("demo");
} catch(NullPointerException e) {
System.out.println("Caught inside demoproc.");
throw e; // re-throw the exception
}
}
public static void main(String args[]) {
try {
demoproc();
} catch(NullPointerException e) {
System.out.println("Recaught: " + e);
}
}
}
```

o/p

```
Caught inside demoproc.
Recaught: java.lang.NullPointerException: demo
```

Throws.java

```
class ThrowsDemo {
static void throwOne() throws IllegalAccessException {
System.out.println("Inside throwOne.");
throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
try {
throwOne();
} catch (IllegalAccessException e) {
System.out.println("Caught " + e);
}
}
}
```

Here is the output generated by running this example program:

```
inside throwOne
caught java.lang.IllegalAccessException: demo
```

finally.java

```
class FinallyDemo {
// Through an exception out of the method.
static void procA() {
try {
System.out.println("inside procA");
throw new RuntimeException("demo");
}
finally {
System.out.println("procA's finally");
}
}
// Return from within a try block.
static void procB() {
try {
System.out.println("inside procB");
return;
} finally {
System.out.println("procB's finally");
}}
// Execute a try block normally.
static void procC() {
try {
System.out.println("inside procC");
} finally {
System.out.println("procC's finally");
}}
public static void main(String args[]) {
try {
procA();
} catch (Exception e) {
System.out.println("Exception caught");
}
procB();
procC();
}}
```

Here is the output generated by the preceding program:

```
inside procA
procA’s finally
Exception caught
inside procB
procB’s finally
inside procC
procC’s finally
```

builtinerror.java

```
class StringIndexOutOfBound_Demo
{
   public static void main(String args[])
   {
      try { This is like chipping
         String a = ""; // length is 22
         char c = a.charAt(24); // accessing 25th element
         System.out.println(c);
      }
      catch(StringIndexOutOfBoundsException e) {
         System.out.println("StringIndexOutOfBoundsException");
      }
   }
}
```

Output:

```
StringIndexOutOfBoundsException
```

Builtinerror1.java

```
//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
```

User defined exception

Custom.java

```
// This program creates a custom exception type.
class MyException extends Exception {
private int detail;
MyException(int a) {
detail = a;
}
public String toString() {
return "MyException[" + detail + "]";
}
}
class ExceptionDemo {
static void compute(int a) throws MyException {
System.out.println("Called compute(" + a + ")");
if(a > 10)
throw new MyException(a);
System.out.println("Normal exit");
}
public static void main(String args[]) {
try {
compute(1);
compute(20);
} catch (MyException e) {
System.out.println("Caught " + e);
}
}
}
```

result:

```
Called compute(1)
Normal exit
Called compute(20)
Caught MyException[20]
```

Chainexcept.java

```
// Demonstrate exception chaining.
class ChainExcDemo {
static void demoproc() {
// create an exception
NullPointerException e = new NullPointerException("top layer");
// add a cause
e.initCause(new ArithmeticException("cause"));
throw e;
}
public static void main(String args[]) {
try {
demoproc();
} catch(NullPointerException e) {
// display top level exception
System.out.println("Caught: " + e);
// display cause exception
System.out.println("Original cause: " + e.getCause());
}
}
}
```

The output from the program is shown here:

```
Caught: java.lang.NullPointerException: top layer
Original cause: java.lang.ArithmeticException: cause
```

For example, consider the following Java program that opens file at locatiobn “C:\test\a.txt” and prints first three lines of it. The program doesn’t compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

```
import java.io.*;

class Main {
   public static void main(String[] args) {
      FileReader file = new FileReader("C:\\test\\a.txt");
      BufferedReader fileInput = new BufferedReader(file);

      // Print first 3 lines of file "C:\test\a.txt"
      for (int counter = 0; counter < 3; counter++)
         System.out.println(fileInput.readLine());

      fileInput.close();
   }
}
```

Run on IDE

Output:

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code -
unreported exception java.io.FileNotFoundException; must be caught or declared to be
thrown at Main.main(Main.java:5)
```

To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since *FileNotFoundException* is a subclass of *IOException*, we can just specify *IOException* in the throws list and make the above program compiler-error-free.

```
import java.io.*;

class Main {
   public static void main(String[] args) throws IOException {
      FileReader file = new FileReader("C:\\test\\a.txt");
      BufferedReader fileInput = new BufferedReader(file);

      // Print first 3 lines of file "C:\test\a.txt"
      for (int counter = 0; counter < 3; counter++)
         System.out.println(fileInput.readLine());

      fileInput.close();
   }
}
```

Output: First three lines of file “C:\test\a.txt”

2) Unchecked are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.
In Java exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.

Consider the following Java program. It compiles fine, but it throws ArithmeticException when run. The compiler allows it to compile, because ArithmeticException is an unchecked exception.

```
class Main {
  public static void main(String args[]) {
    int x = 0;
    int y = 10;
    int z = y/x;
  }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Main.main(Main.java:5)
Java Result: 1
```

Base.java

```
#include<iostream>
using namespace std;

class Base {};
class Derived: public Base {};
int main()
{
  Derived d;
  // some other stuff
  try {
     // Some monitored code
     throw d;
  }
  catch(Base b) {
      cout<<"Caught Base Exception";
  }
  catch(Derived d) {  //This catch block is NEVER executed
      cout<<"Caught Derived Exception";
  }
  getchar();
  return 0;
}
```

Derive.java

```
#include<iostream>
using namespace std;

class Base {};
class Derived: public Base {};
int main()
{
  Derived d;
  // some other stuff
  try {
     // Some monitored code
     throw d;
  }
  catch(Derived d) {
      cout<<"Caught Derived Exception";
  }
  catch(Base b) {
      cout<<"Caught Base Exception";
  }
  getchar();
  return 0;
}
```

Derive 1.java

```
//filename Main.java
class Base extends Exception {}
class Derived extends Base  {}
public class Main {
  public static void main(String args[]) {
    try {
       throw new Derived();
    }
    catch(Base b) {}
    catch(Derived d) {}
  }
}
```

Stackcall.java

```
// Java program to demonstrate exception is thrown how the runTime system searches th call
stack to find appropriate exception handler.
class ExceptionThrown
{
        // It throws the Exception(ArithmeticException).
        // Appropriate Exception handler is not found within this method.
        static int divideByZero(int a, int b){

                // this statement will cause ArithmeticException(/ by zero)
                int i = a/b;
                return i;
        }
        // The runTime System searches the appropriate Exception handler in this method also
but couldn't have found. So looking forward on the call stack.
        static int computeDivision(int a, int b) {
                int res =0;
                try
                {
                res = divideByZero(a,b);
                }
                // doesn't matches with ArithmeticException
                catch(NumberFormatException ex)
                {
                System.out.println("NumberFormatException is occured");
                }
                return res;
        }

        // In this method found appropriate Exception handler. i.e. matching catch block.
        public static void main(String args[]){
                int a = 1;
                int b = 0;

                try
                {
                        int i = computeDivision(a,b);
                }

                // matching ArithmeticException
                catch(ArithmeticException ex)
                {
                        // getMessage will print description of exception(here / by zero)
                        System.out.println(ex.getMessage());
                }       }
}
```

Nested try block

```
class Nest{
  public static void main(String args[]){
              //Parent try block
    try{
             //Child try block1
        try{
          System.out.println("Inside block1");
          int b =45/0;
          System.out.println(b);
        }
        catch(ArithmeticException e1){
          System.out.println("Exception: e1");
        }
        //Child try block2
        try{
          System.out.println("Inside block2");
          int b =45/0;
          System.out.println(b);
        }
        catch(ArrayIndexOutOfBoundsException e2){
          System.out.println("Exception: e2");
        }
       System.out.println("Just other statement");
    }
    catch(ArithmeticException e3){
              System.out.println("Arithmetic Exception");
        System.out.println("Inside parent try catch block");
    }
    catch(ArrayIndexOutOfBoundsException e4){
             System.out.println("ArrayIndexOutOfBoundsException");
        System.out.println("Inside parent try catch block");
    }
    catch(Exception e5){
             System.out.println("Exception");
        System.out.println("Inside parent try catch block");
    }
    System.out.println("Next statement..");
  }}
```

Output:

```
Inside block1
Exception: e1
Inside block2
Arithmetic Exception
Inside parent try catch block
Next statement..
```

```
Finally.java
class JavaFinally
{
  public static void main(String args[])
  {
    System.out.println(JavaFinally.myMethod());
  }
  public static int myMethod()
  {
    try {
     return 112;
    }
    finally {
     System.out.println("This is Finally block");
     System.out.println("Finally block ran even after return statement");
    }
  }
}
```

Output of above program:

```
This is Finally block
Finally block ran even after return statement
112
```

Close()

```
try{
   OutputStream osf = new FileOutputStream( "filename" );
   OutputStream osb = new BufferedOutputStream(opf);
   ObjectOutput op = new ObjectOutputStream(osb);
   try{
      output.writeObject(writableObject);
   }
   finally{
      op.close();
   }
}
catch(IOException e1){
    System.out.println(e1);
}
```

Exceptionwithfinally.java

```
class Example3{
  public static void main(String args[]){
    try{
      System.out.println("First statement of try block");
      int num=45/0;
      System.out.println(num);
    }
    catch(ArithmeticException e){
      System.out.println("ArithmeticException");
    }
    finally{
      System.out.println("finally block");
    }
    System.out.println("Out of try-catch-finally block");
  }
}
```

Output:

```
First statement of try block
ArithmeticException
finally block
Out of try-catch-finally block
```

Throw.java

```
public class MyClass {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

The output will be:

```
Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least
18 years old.
        at MyClass.checkAge(MyClass.java:4)
        at MyClass.main(MyClass.java:12)
```

Myexception.java

```
// Java program to demonstrate user defined exception
// This program throws an exception whenever balance amount is below Rs 1000
class MyException extends Exception
{
        //store account information
        private static int accno[] = {1001, 1002, 1003, 1004};
        private static String name[] = {"Nish", "Shubh", "Sush", "Abhi", "Akash"};
        private static double bal[] = {10000.00, 12000.00, 5600.0, 999.00, 1100.55};
        // default constructor
        MyException() { }

        // parametrized constructor
        MyException(String str) { super(str); }

        // write main()
        public static void main(String[] args)
        {
                try {
                // display the heading for the table
                System.out.println("ACCNO" + "\t" + "CUSTOMER" + "\t" + "BALANCE");
                // display the actual account information
                        for (int i = 0; i < 5 ; i++)
                        {
                                System.out.println(accno[i] + "\t" + name[i] + "\t" + bal[i]);
                                // display own exception if balance < 1000
                                if (bal[i] < 1000)
                                {
                        MyException me = new MyException("Balance is less than 1000");
                                        throw me;
                                }       }
                } //end of try

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

RunTime Error

```
 MyException: Balance is less than 1000
   at MyException.main(fileProperty.java:36)
```

Output:

```
ACCNO    CUSTOMER    BALANCE
1001    Nish    10000.0
1002    Shubh    12000.0
1003    Sush    5600.0
1004    Abhi    999.0
```

Checked exception

```
import java.io.*;
class Example {
  public static void main(String args[])
  {
        FileInputStream fis = null;
        /*This constructor FileInputStream(File filename) throws FileNotFoundException which
            is a checked exception        */
      fis = new FileInputStream("B:/myfile.txt");
        int k;

/* Method read() of FileInputStream class also throws a checked exception: IOException       */
        while(( k = fis.read() ) != -1)
        {
                System.out.print((char)k);
        }

        /*The method close() closes the file input stream  It throws IOException*/
        fis.close();
  }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
Unhandled exception type FileNotFoundException
Unhandled exception type IOException
Unhandled exception type IOException
```

Solution
Method 1: Declare the exception using throws keyword.

```
import java.io.*;
class Example {
  public static void main(String args[]) throws IOException
  {
    FileInputStream fis = null;
    fis = new FileInputStream("B:/myfile.txt");
    int k;

    while(( k = fis.read() ) != -1)
    {
          System.out.print((char)k);
    }
    fis.close();
  }
}
```

Output:
File content is displayed on the screen.

Method 2: Handle them using try-catch blocks.

```
import java.io.*;
class Example {
  public static void main(String args[])
  {
        FileInputStream fis = null;
        try{
            fis = new FileInputStream("B:/myfile.txt");
        }catch(FileNotFoundException fnfe){
        System.out.println("The specified file is not " +"present at the given path");
         }
        int k;
        try{
            while(( k = fis.read() ) != -1)
            {
                    System.out.print((char)k);
            }
            fis.close();
        }catch(IOException ioe){
            System.out.println("I/O error occurred: "+ioe);
         }
  }
}
```

Unchecked exception

```
class Example {
   public static void main(String args[]) {
         try{
            int arr[] ={1,2,3,4,5};
            System.out.println(arr[7]);
         }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("The specified index does not exist " +
                  "in array. Please correct the error.");
         }
   }
}
```

Output:

```
The specified index does not exist in array. Please correct the error.
```

Exception Handling with Method Overriding
When Exception handling is involved with Method overriding, ambiguity occurs. The compiler gets confused as which definition is to be followed. Such problems were of two types:

- **Problem 1: If The SuperClass doesn’t declare an exception:**
  In this problem, two cases arise:
  - **Case 1: If SuperClass doesn’t declare any exception and subclass declare checked exception**

```
import java.io.*;

class SuperClass {
   // SuperClass doesn't declare any exception
   void method()
   {
      System.out.println("SuperClass");
   }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

   // method() declaring Checked Exception IOException
   void method() throws IOException
   {
      // IOException is of type Checked Exception so the compiler will give Error
        System.out.println("SubClass");
   }

   // Driver code
   public static void main(String args[])
   {
      SuperClass s = new SubClass();
      s.method();
   }
}
```

Compile Errors:

```
prog.java:16: error:
method() in SubClass cannot override method() in SuperClass
   void method() throws IOException
        ^
  overridden method does not throw IOException
1 error
```

- **Case 2: If SuperClass doesn’t declare any exception and SubClass declare Unchecked exception**

```
import java.io.*;

class SuperClass {

   // SuperClass doesn't declare any exception
   void method()
   {
      System.out.println("SuperClass");
   }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

   // method() declaring Unchecked Exception ArithmeticException
   void method() throws ArithmeticException
   {
// ArithmeticException is of type Unchecked Exception so the compiler won't give any
error

      System.out.println("SubClass");
   }

   // Driver code
   public static void main(String args[])
   {
      SuperClass s = new SubClass();
      s.method();
   }
}
```

Output:

```
SubClass
```

- **Problem 2: If The SuperClass declares an exception:**
  In this problem also, three cases arise:
  - **Case 1: If SuperClass declares an exception and SubClass declares exceptions other than the child exception of the SuperClass declared Exception**

```
import java.io.*;

class SuperClass {
     // SuperClass declares an exception
   void method() throws RuntimeException
   {
      System.out.println("SuperClass");
   }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {
 // SubClass declaring an exception which are not a child exception of
//RuntimeException

  void method() throws Exception
   {
// Exception is not a child exception of the RuntimeException So the compiler
//will give an error
        System.out.println("SubClass");
   }

   // Driver code
   public static void main(String args[])
   {
      SuperClass s = new SubClass();
      s.method();
   }
}
```

Compile Errors:

```
prog.java:16: error:
method() in SubClass cannot override method() in SuperClass
   void method() throws Exception
        ^
  overridden method does not throw Exception
1 error
```

- **Case 2: If SuperClass declares an exception and SubClass declares an child exception of the SuperClass declared Exception.**

```
import java.io.*;

class SuperClass {

  // SuperClass declares an exception
  void method() throws RuntimeException
  {
    System.out.println("SuperClass");
  }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

  // SubClass declaring a child exception of RuntimeException
  void method() throws ArithmeticException
  {
     // ArithmeticException is a child exception of the RuntimeException
    // So the compiler won't give an error
    System.out.println("SubClass");
  }

  // Driver code
  public static void main(String args[])
  {
    SuperClass s = new SubClass();
    s.method();
  }
}
```

Output:

```
SubClass
```

- **Case 3: If SuperClass declares an exception and SubClass declares without exception.**
  Example:

```
import java.io.*;

class SuperClass {

   // SuperClass declares an exception
   void method() throws IOException
   {
      System.out.println("SuperClass");
   }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {
     // SubClass declaring without exception
   void method()
   {
      System.out.println("SubClass");
   }
    // Driver code
   public static void main(String args[])
   {
      SuperClass s = new SubClass();
   try {
      s.method();
   } catch (IOException e) {
      e.printStackTrace();
   }
   }
}
```

Output:

```
SubClass
```

Chained exception

```
// Java program to demonstrate working of chained exceptions
public class ExceptionHandling
{
public static void main(String[] args)
{
	try
	{
	// Creating an exception
	NumberFormatException ex = new NumberFormatException("Exception");
	// Setting a cause of the exception
	ex.initCause(new NullPointerException( "This is actual cause of the exception"));

	// Throwing an exception with cause.
		throw ex;
	}
	catch(NumberFormatException ex)
	{
	// displaying the exception
		System.out.println(ex);
	// Getting the actual cause of the exception
	System.out.println(ex.getCause());
	}
	}
}
```

Thread1.java

```
// Controlling the main Thread.
class CurrentThreadDemo {
public static void main(String args[]) {
Thread t = Thread.currentThread();
System.out.println("Current thread: " + t);
// change the name of the thread
t.setName("My Thread");
System.out.println("After name change: " + t);
try {
for(int n = 5; n > 0; n--) {
System.out.println(n);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println("Main thread interrupted");
}
}
}
```

Output:

```
Current thread: Thread[main,5,main]
After name change: Thread[My Thread,5,main]
5
4
3
2
1
```

Runnable.java

```
// Create a second thread.
class NewThread implements Runnable {
Thread t;
NewThread() {
// Create a new, second thread
t = new Thread(this, "Demo Thread");// constructor thread
System.out.println("Child thread: " + t);
t.start(); // Start the thread
}
// This is the entry point for the second thread.
public void run() {
try {
for(int i = 5; i > 0; i--) {
System.out.println("Child Thread: " + i);
t.sleep(500);
}
} catch (InterruptedException e) {
System.out.println("Child interrupted.");
}
System.out.println("Exiting child thread.");
}
}
class ThreadDemo {
public static void main(String args[]) {
new NewThread(); // create a new thread
try {
for(int i = 5; i > 0; i--) {
System.out.println("Main Thread: " + i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println("Main thread interrupted.");
}
System.out.println("Main thread exiting.");
}
}
```

Output:

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
```

```
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

Extend.java

```
// Create a second thread by extending Thread
class NewThread extends Thread {
NewThread() {
// Create a new, second thread
super("Demo Thread");
System.out.println("Child thread: " + this);
start(); // Start the thread
}
// This is the entry point for the second thread.
public void run() {
try {
for(int i = 5; i > 0; i--) {
System.out.println("Child Thread: " + i);
Thread.sleep(500);
}
} catch (InterruptedException e) {
System.out.println("Child interrupted.");
}
System.out.println("Exiting child thread.");
}
}
class ExtendThread {
public static void main(String args[]) {
new NewThread(); // create a new thread
try {
for(int i = 5; i > 0; i--) {
System.out.println("Main Thread: " + i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println("Main thread interrupted.");
}
System.out.println("Main thread exiting.");
}
}
```

```
class RunnableDemo implements Runnable {
  private Thread t;
  private String threadName;

  RunnableDemo( String name) {
    threadName = name;
    System.out.println("Creating " +  threadName );
  }

  public void run() {
    System.out.println("Running " +  threadName );
    try {
      for(int i = 4; i > 0; i--) {
        System.out.println("Thread: " + threadName + ", " + i);
        // Let the thread sleep for a while.
        Thread.sleep(50);
      }
    } catch (InterruptedException e) {
      System.out.println("Thread " +  threadName + " interrupted.");
    }
    System.out.println("Thread " +  threadName + " exiting.");
  }

  public void start () {
    System.out.println("Starting " +  threadName );
    if (t == null) {
      t = new Thread (this, threadName);
      t.start ();
    }
  }
}

public class TestThread {
  public static void main(String args[]) {
    RunnableDemo R1 = new RunnableDemo( "Thread-1");
    R1.start();

    RunnableDemo R2 = new RunnableDemo( "Thread-2");
    R2.start();
  }
}
```

Output

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
```

```
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }

   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
      } catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}

public class TestThread {

   public static void main(String args[]) {
      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

Output

```
Creating Thread-1
Starting Thread-1
```

```
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Multithread.java

```
// Create multiple threads.
class NewThread implements Runnable {
String name; // name of thread
Thread t;
NewThread(String threadname) {
name = threadname;
t = new Thread(this, name);
System.out.println("New thread: " + t);
t.start(); // Start the thread
}
// This is the entry point for thread.
public void run() {
try {
for(int i = 5; i > 0; i--) {
System.out.println(name + ": " + i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println(name + "Interrupted");
}
System.out.println(name + " exiting.");
}
}
class MultiThreadDemo {
public static void main(String args[]) {
new NewThread("One"); // start threads
new NewThread("Two");
new NewThread("Three");
try {
// wait for other threads to end
Thread.sleep(10000);
} catch (InterruptedException e) {
System.out.println("Main thread Interrupted");
}
System.out.println("Main thread exiting.");
}
}
```

The output from this program is shown here:

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

```
Two: 4
Three: 4
One: 3
Three: 3
Two: 3
One: 2
Three: 2
Two: 2
One: 1
Three: 1
Two: 1
One exiting.
Two exiting.
Three exiting.
Main thread exiting.
```

Isalivejoin.java

```
// Using join() to wait for threads to finish.
class NewThread implements Runnable {
String name; // name of thread
Thread t;
NewThread(String threadname) {
name = threadname;
t = new Thread(this, name);
System.out.println("New thread: " + t);
t.start(); // Start the thread
}
// This is the entry point for thread.
public void run() {
try {
for(int i = 5; i > 0; i--) {
System.out.println(name + ": " + i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println(name + " interrupted.");
}
System.out.println(name + " exiting.");
}
}
class DemoJoin {
public static void main(String args[]) {
NewThread ob1 = new NewThread("One");
NewThread ob2 = new NewThread("Two");
NewThread ob3 = new NewThread("Three");
System.out.println("Thread One is alive: " + ob1.t.isAlive());
System.out.println("Thread Two is alive: " + ob2.t.isAlive());
System.out.println("Thread Three is alive: "+ ob3.t.isAlive());
// wait for threads to finish
try {
System.out.println("Waiting for threads to finish.");
ob1.t.join();
ob2.t.join();
ob3.t.join();
} catch (InterruptedException e) {
System.out.println("Main thread Interrupted");
}
System.out.println("Thread One is alive: " + ob1.t.isAlive());
System.out.println("Thread Two is alive: " + ob2.t.isAlive());
System.out.println("Thread Three is alive: " + ob3.t.isAlive());
System.out.println("Main thread exiting.");
}
}
output
New thread: Thread[One,5,main]
New thread: Thread[Two,5,main]
New thread: Thread[Three,5,main]
Thread One is alive: true
Thread Two is alive: true
Thread Three is alive: true
Waiting for threads to finish.
One: 5
Two: 5
Three: 5
One: 4
Two: 4
Three: 4
One: 3
Two: 3
Three: 3
One: 2
Two: 2
Three: 2
One: 1
Two: 1
Three: 1
Two exiting.
Three exiting.
One exiting.
Thread One is alive: false
Thread Two is alive: false
Thread Three is alive: false
Main thread exiting.
```

Threadpriority.java

```
// Demonstrate thread priorities.
class clicker implements Runnable {
long click = 0;
Thread t;
private volatile boolean running = true;
public clicker(int p) {
t = new Thread(this);
t.setPriority(p);
}
public void run() {
while (running) {
click++;
}
}
public void stop() {
running = false;
}
public void start() {
t.start();
}
}

class HiLoPri {
public static void main(String args[]) {
Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
clicker hi = new clicker(Thread.NORM_PRIORITY + 2);
clicker lo = new clicker(Thread.NORM_PRIORITY - 2);
lo.start();
hi.start();
try {
Thread.sleep(10000);
} catch (InterruptedException e) {
System.out.println("Main thread interrupted.");
}
lo.stop();
hi.stop();
// Wait for child threads to terminate.
try {
hi.t.join();
lo.t.join();
} catch (InterruptedException e) {
System.out.println("InterruptedException caught");
}
System.out.println("Low-priority thread: " + lo.click);
System.out.println("High-priority thread: " + hi.click);
}
}
```

Output

```
Low-priority thread: 4408112
High-priority thread: 589626904
```

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
```

join()

```
public class MyThread extends Thread
{
	public void run()
	{
		System.out.println("r1 ");
		try {
		Thread.sleep(500);
		}catch(InterruptedException ie){ }
		System.out.println("r2 ");
	}
	public static void main(String[] args)
	{
		MyThread t1=new MyThread();
		MyThread t2=new MyThread();
		t1.start();

		try{
			t1.join();	//Waiting for t1 to finish
		}catch(InterruptedException ie){}

		t2.start();
	}
}
```

output

```
r1
r2
r1
r2
```

```
// join(long miliseconds) method
class TestJoinMethod2 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[]){
 TestJoinMethod2 t1=new TestJoinMethod2();
 TestJoinMethod2 t2=new TestJoinMethod2();
 TestJoinMethod2 t3=new TestJoinMethod2();
 t1.start();
 try{
  t1.join(1500);
 }catch(Exception e){System.out.println(e);}

 t2.start();
 t3.start();
 }
}
Output:1-t1
       2 t1
       3 t1
       1 t2
       4 t1
       1 t3
       2 t2
       5 t1--terminate
       2 t3
       3 t2
       3 t3
       4 t2
       4 t3
       5 t2 - terminate
       5 t3 -terminate
```

Nonsynch.java

```
// This program is not synchronized.
class Callme {
void call(String msg) {
// synchronized void call(String msg)
System.out.print("[" + msg);
try {
Thread.sleep(1000);
} catch(InterruptedException e) {
System.out.println("Interrupted");
}
System.out.println("]");
}
}
class Caller implements Runnable {
String msg;
Callme target;
Thread t;
public Caller(Callme targ, String s) {
target = targ;
msg = s;
t = new Thread(this);
t.start();
}
public void run() {
target.call(msg);
}
}
class Synch {
public static void main(String args[]) {
Callme target = new Callme();
Caller ob1 = new Caller(target, "Hello");
Caller ob2 = new Caller(target, "Synchronized");
Caller ob3 = new Caller(target, "World");
// wait for threads to end
try {
ob1.t.join();
ob2.t.join();
ob3.t.join();
} catch(InterruptedException e) {
System.out.println("Interrupted");
}
}
}
```

Here is the output produced by this program:

```
Hello[Synchronized[World]
]
]

Using comment
[Hello]
[Synchronized]
[World]
```

```
class PrintDemo {
  public void printCount() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Counter   ---   "  + i );
      }
    } catch (Exception e) {
      System.out.println("Thread  interrupted.");
    }
  }
}

class ThreadDemo extends Thread {
  private Thread t;
  private String threadName;
  PrintDemo  PD;

  ThreadDemo( String name,  PrintDemo pd) {
    threadName = name;
    PD = pd;
  }

  public void run() {
    synchronized(PD) {
      PD.printCount();
    }
    System.out.println("Thread " +  threadName + " exiting.");
  }

  public void start () {
    System.out.println("Starting " +  threadName );
    if (t == null) {
      t = new Thread (this, threadName);
      t.start ();
    }
  }
}

public class TestThread {
  public static void main(String args[]) {
    PrintDemo PD = new PrintDemo();
    ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );
    ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );
    T1.start();
    T2.start();
    // wait for threads to end
    try {
      T1.join();
      T2.join();
    } catch ( Exception e) {
      System.out.println("Interrupted");
    }
  }
}
```
This produces the same result every time you run this program −

```
Output
Starting Thread - 1
Starting Thread - 2
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 1  exiting.
Counter   ---   5
Counter   ---   4
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 2  exiting.
```

Incorrect.java

```
// An incorrect implementation of a producer and consumer.
class Q {
int n;
synchronized int get() {
System.out.println("Got: " + n);
return n;
}
synchronized void put(int n) {
this.n = n;
System.out.println("Put: " + n);
}
}

class Producer implements Runnable {
Q q;
Producer(Q q) {
this.q = q;
new Thread(this, "Producer").start();
}
public void run() {
int i = 0;
while(true) {
q.put(i++);
}
}
}
class Consumer implements Runnable {
Q q;
Consumer(Q q) {
this.q = q;
new Thread(this, "Consumer").start();
}
public void run() {
while(true) {
q.get();
}
}
}
class PC {
public static void main(String args[]) {
Q q = new Q();
new Producer(q);
new Consumer(q);
System.out.println("Press Control-C to stop.");
}
}
```

Output:

```
Put: 1
Got: 1
Got: 1
Got: 1
Got: 1
Got: 1
Put: 2
Put: 3
Put: 4
Put: 5
Put: 6
Put: 7
Got: 7
```

Correct.java

```
// A correct implementation of a producer and consumer.
class Q {
int n;
boolean valueSet = false;
synchronized int get() {
while(!valueSet)
try {
wait();
} catch(InterruptedException e) {
System.out.println("InterruptedException caught");
}
System.out.println("Got: " + n);
valueSet = false;
notify();
return n;
}
synchronized void put(int n) {
while(valueSet)
try {
wait();
} catch(InterruptedException e) {
System.out.println("InterruptedException caught");
}
this.n = n;
valueSet = true;
System.out.println("Put: " + n);
notify();
}
}
class Producer implements Runnable {
Q q;
Producer(Q q) {
this.q = q;
new Thread(this, "Producer").start();
}
public void run() {
int i = 0;
while(true) {
q.put(i++);
}
}
}
class Consumer implements Runnable {
Q q;
Consumer(Q q) {
```

```
this.q = q;
new Thread(this, "Consumer").start();
}
public void run() {
while(true) {
q.get();
}
}
}
class PCFixed {
public static void main(String args[]) {
Q q = new Q();
new Producer(q);
new Consumer(q);
System.out.println("Press Control-C to stop.");
}
}
```

Output:

```
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
Put: 5
Got: 5
```

Deadlock.java

```
// An example of deadlock.
class A {
synchronized void foo(B b) {
String name = Thread.currentThread().getName();
System.out.println(name + " entered A.foo");
try {
Thread.sleep(1000);
} catch(Exception e) {
System.out.println("A Interrupted");
}
System.out.println(name + " trying to call B.last()");
b.last();
}
synchronized void last() {
System.out.println("Inside A.last");
}
}
class B {
synchronized void bar(A a) {
String name = Thread.currentThread().getName();
System.out.println(name + " entered B.bar");
try {
Thread.sleep(1000);
} catch(Exception e) {
System.out.println("B Interrupted");
}
System.out.println(name + " trying to call A.last()");
a.last();
}
synchronized void last() {
System.out.println("Inside A.last");
}
}
class Deadlock implements Runnable {
A a = new A();
B b = new B();
Deadlock() {
Thread.currentThread().setName("MainThread");
Thread t = new Thread(this, "RacingThread");
t.start();
a.foo(b); // get lock on a in this thread.
System.out.println("Back in main thread");
}
public void run() {
b.bar(a); // get lock on b in other thread.
System.out.println("Back in other thread");
}
public static void main(String args[]) {
new Deadlock();
}
}
```

When you run this program, you will see the output shown here:

```
MainThread entered A.foo
RacingThread entered B.bar
MainThread trying to call B.last()
RacingThread trying to call A.last()
```

Suspend.java

```
// Using suspend() and resume().
class NewThread implements Runnable {
String name; // name of thread
Thread t;
NewThread(String threadname) {
name = threadname;
t = new Thread(this, name);
System.out.println("New thread: " + t);
t.start(); // Start the thread
}
// This is the entry point for thread.
public void run() {
try {
for(int i = 15; i > 0; i--) {
System.out.println(name + ": " + i);
Thread.sleep(200);
}
} catch (InterruptedException e) {
System.out.println(name + " interrupted.");
}
System.out.println(name + " exiting.");
}
}
class SuspendResume {
public static void main(String args[]) {
NewThread ob1 = new NewThread("One");
NewThread ob2 = new NewThread("Two");
try {
Thread.sleep(1000);
ob1.t.suspend();
System.out.println("Suspending thread One");
Thread.sleep(1000);
ob1.t.resume();
System.out.println("Resuming thread One");
ob2.t.suspend();
System.out.println("Suspending thread Two");
Thread.sleep(1000);
ob2.t.resume();
System.out.println("Resuming thread Two");
} catch (InterruptedException e) {
System.out.println("Main thread Interrupted");
}
// wait for threads to finish
try {
System.out.println("Waiting for threads to finish.");
ob1.t.join();
ob2.t.join();
} catch (InterruptedException e) {
System.out.println("Main thread Interrupted");
}
System.out.println("Main thread exiting.");
}
}
```

Sample output from this program is shown here. (Your output may differ based on processor speed and task load.)

```
New thread: Thread[One,5,main]
One: 15
New thread: Thread[Two,5,main]
Two: 15
One: 14
Two: 14
One: 13
Two: 13
One: 12
Two: 12
One: 11
Two: 11
Suspending thread One
Two: 10
Two: 9
Two: 8
Two: 7
Two: 6
Resuming thread One
Suspending thread Two
One: 10
One: 9
One: 8
One: 7
One: 6
Resuming thread Two
Waiting for threads to finish.
Two: 5
One: 5
Two: 4
One: 4
Two: 3
One: 3
Two: 2
One: 2
Two: 1
```

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

Interprocesscommunication.java

```
class Customer{
int amount=10000;

synchronized void withdraw(int amount){
System.out.println("going to withdraw...");
if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=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
```

ThreadClassDemo.java

```
public class ThreadClassDemo {

  public static void main(String [] args) {
    Runnable hello = new DisplayMessage("Hello");
    Thread thread1 = new Thread(hello);
    thread1.setDaemon(true);
    thread1.setName("hello");
    System.out.println("Starting hello thread...");
    thread1.start();

    Runnable bye = new DisplayMessage("Goodbye");
    Thread thread2 = new Thread(bye);
    thread2.setPriority(Thread.MIN_PRIORITY);
    thread2.setDaemon(true);
    System.out.println("Starting goodbye thread...");
    thread2.start();

    System.out.println("Starting thread3...");
    Thread thread3 = new GuessANumber(27);
    thread3.start();
    try {
      thread3.join();
    } catch (InterruptedException e) {
      System.out.println("Thread interrupted.");
    }
    System.out.println("Starting thread4...");
    Thread thread4 = new GuessANumber(75);

    thread4.start();
    System.out.println("main() is ending...");
  }
}
```

This will produce the following result. You can try this example again and again and you will get a different result every time.

Output

```
Starting hello thread...
Starting goodbye thread...
Hello
Hello
Hello
Hello
Hello
Hello
Goodbye
```

Goodbye
Goodbye
Goodbye
Goodbye
....

Daemon thread in java

```
public class TestDaemonThread1 extends Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();

  t1.setDaemon(true);//now t1 is daemon thread

  t1.start();//starting threads
  t2.start();
  t3.start();
 }
}
```

Output

```
daemon thread work
user thread work
user thread work
```

FileOutputStream.java

```
import java.io.FileOutputStream;

public class FileOutputStreamExample {
public static void main(String args[]){

try{
FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
// write a byte
fout.write(65);
// write a string
String s="Welcome to javaTpoint.";
byte b[]=s.getBytes();//converting string into byte array
fout.write(b);

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

Output:

```
Success...
```

FileInputStream.java
// Java FileInputStream Class
Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

```
import java.io.FileInputStream;

public class DataStreamExample {
public static void main(String args[]){

try{

FileInputStream fin=new FileInputStream("D:\\testout.txt");
int i=fin.read();
System.out.print((char)i);

// reading a file
int i=0;
while((i=fin.read())!=-1){
System.out.print((char)i);
}

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

Character.java

Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency, additional functionality, or both. Some of the implementation class are BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

```
// Java Program illustrating that we can read a file in a human readable format using FileReader

import java.io.*;   // Accessing FileReader, FileWriter, IOException
public class GfG
{
public static void main(String[] args) throws IOException
{
FileReader sourceStream = null;
try
{
sourceStream = new FileReader("test.txt");

// Reading sourcefile and writing content to target file character by character.
int temp;
while ((temp = sourceStream.read()) != -1)
System.out.println((char)temp);
}
finally
{
// Closing stream as no longer in use
if (sourceStream != null)
sourceStream.close();
}
}
}
```

Output :
Shows contents of file test.txt

Byte.java

Java FileOutputStream is an output stream used for writing data to a file. If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

```
// Java Program illustrating the Byte Stream to copy contents of one file to another file.
import java.io.*;
public class BStream
{
public static void main(String[] args) throws IOException
{
FileInputStream sourceStream = null;
FileOutputStream targetStream = null;

try
{
sourceStream = new FileInputStream("sorcefile.txt");
targetStream = new FileOutputStream ("targetfile.txt");

// Reading source file and writing content to target file byte by byte
int temp;
while ((temp = sourceStream.read()) != -1)
targetStream.write((byte)temp);
}
finally
{
if (sourceStream != null)
sourceStream.close();
if (targetStream != null)
targetStream.close();
}
}
}
```

SequenceInputStream.java

Java SequenceInputStream class is used to read data from multiple streams. It reads data sequentially (one by one).

```
//Example that reads the data from two files and writes into another file
package com.javatpoint;

import java.io.*;
class Input1{
public static void main(String args[])throws Exception{

FileInputStream fin1=new FileInputStream("D:\\testin1.txt");
FileInputStream fin2=new FileInputStream("D:\\testin2.txt");
FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
SequenceInputStream sis=new SequenceInputStream(fin1,fin2);
int i;
while((i=sis.read())!=-1)
{
fout.write(i);
}
sis.close();
fout.close();
fin1.close();
fin2.close();
System.out.println("Success..");
}
}
```

Output:

```
Succeess...
```

byteArrayOutputStream.java

Java ByteArrayOutputStream class is used to write common data into multiple files. In this stream, the data is written into a byte array which can be written to multiple streams later. The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams. The buffer of ByteArrayOutputStream automatically grows according to data.

```
package com.javatpoint;
import java.io.*;
public class DataStreamExample {
public static void main(String args[])throws Exception{
FileOutputStream fout1=new FileOutputStream("D:\\f1.txt");
FileOutputStream fout2=new FileOutputStream("D:\\f2.txt");

ByteArrayOutputStream bout=new ByteArrayOutputStream();
bout.write(65);
bout.writeTo(fout1);
bout.writeTo(fout2);

bout.flush();
bout.close();//has no effect
System.out.println("Success...");
}
}
```

Output:

```
Success...
```

Bufferenstream.java

Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. For adding the buffer in an OutputStream, use the BufferedOutputStream class.

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast. The important points about BufferedInputStream are:

- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
- When a BufferedInputStream is created, an internal buffer array is created.

```
import java.io.*;
public class BufferedOutputStreamExample{
public static void main(String args[])throws Exception{

    FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
    BufferedOutputStream bout=new BufferedOutputStream(fout);
    String s="Welcome to javaTpoint.";
    byte b[]=s.getBytes();
    bout.write(b);
    bout.flush();
    bout.close();
    fout.close();
    System.out.println("success");
}
}
```

Output:

```
Success
```

Bufferedinput.java

```
import java.io.*;
public class BufferedInputStreamExample{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("D:\\testout.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
    while((i=bin.read())!=-1){
     System.out.print((char)i);
    }
    bin.close();
    fin.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

Bufferereader.java

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.
Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

```
import java.io.*;
public class BufferedReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        BufferedReader br=new BufferedReader(fr);

        int i;
        while((i=br.read())!=-1){
        System.out.print((char)i);
        }
        br.close();
        fr.close();
   }
}
```

Bufferwriter.java

```
import java.io.*;
public class BufferedWriterExample {
public static void main(String[] args) throws Exception {
FileWriter writer = new FileWriter("D:\\testout.txt");
BufferedWriter buffer = new BufferedWriter(writer);
buffer.write("Welcome to javaTpoint.");
buffer.close();
System.out.println("Success");
}
}
```

Create a new File

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative. The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

```
// importing the File class
import java.io.File;

class Main {
  public static void main(String[] args) {

    // create a file object for the current location
    File file = new File("newFile.txt");

    try {

      // trying to create a file based on the object
      boolean value = file.createNewFile();
      if (value) {
        System.out.println("The new file is created.");
      }
      else {
        System.out.println("The file already exists.");
      }
    }
    catch(Exception e) {
      e.getStackTrace();
    }
  }
}
```

Delete a file

```
import java.io.File;

class Main {
  public static void main(String[] args) {

    // creates a file object
    File file = new File("file.txt");

    // deletes the file
    boolean value = file.delete();
    if(value) {
      System.out.println("The File is deleted.");
    }
    else {
      System.out.println("The File is not deleted.");
    }
  }
}
```

```
package com.journaldev.files;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.attribute.PosixFilePermission;
import java.util.HashSet;
import java.util.Set;

public class FilePermissions {

    public static void main(String[] args) throws IOException {
        File file = new File("/Users/pankaj/temp.txt");

        //set application user permissions to 455
        file.setExecutable(false);
        file.setReadable(false);
        file.setWritable(true);

        //change permission to 777 for all the users no option for group and others
        file.setExecutable(true, false);
        file.setReadable(true, false);
        file.setWritable(true, false);

        //using PosixFilePermission to set file permissions 777
        Set<PosixFilePermission> perms = new HashSet<PosixFilePermission>();
        //add owners permission
        perms.add(PosixFilePermission.OWNER_READ);
        perms.add(PosixFilePermission.OWNER_WRITE);
        perms.add(PosixFilePermission.OWNER_EXECUTE);
        //add group permissions
        perms.add(PosixFilePermission.GROUP_READ);
        perms.add(PosixFilePermission.GROUP_WRITE);
        perms.add(PosixFilePermission.GROUP_EXECUTE);
        //add others permissions
        perms.add(PosixFilePermission.OTHERS_READ);
        perms.add(PosixFilePermission.OTHERS_WRITE);
        perms.add(PosixFilePermission.OTHERS_EXECUTE);

        Files.setPosixFilePermissions(Paths.get("/Users/pankaj/run.sh"), perms);
    }

}
```

Create a new directory in Java

```
import java.io.File;

class Main {
  public static void main(String[] args) {

    // creates a file object with specified path
    File file = new File("Java Example\\directory");

    // tries to create a new directory
    boolean value = file.mkdir();
    if(value) {
      System.out.println("The new directory is created.");
    }
    else {
      System.out.println("The directory already exists.");
    }
  }
}
```

File permission

```
File file = new File("/Users/run.sh");

//check file permissions for application user
System.out.println("File is readable? "+file.canRead());
System.out.println("File is writable? "+file.canWrite());
System.out.println("File is executable? "+file.canExecute());

//change file permissions for application user only
file.setReadable(false);
file.setWritable(false);
file.setExecutable(false);

//change file permissions for other users also
file.setReadable(true, false);
file.setWritable(true, false);
file.setExecutable(true, true);
```

Serializationdeserialization.java

```
import java.io.*;

class Demo implements java.io.Serializable
{
   public int a;
   public String b;

   // Default constructor
   public Demo(int a, String b)
   {
      this.a = a;
      this.b = b;
   }

}

class Test
{
   public static void main(String[] args)
   {
      Demo object = new Demo(1, "geeksforgeeks");
      String filename = "file.ser";

      // Serialization
      try
      {
         //Saving of object in a file
         FileOutputStream file = new FileOutputStream(filename);
         ObjectOutputStream out = new ObjectOutputStream(file);

         // Method for serialization of object
         out.writeObject(object);

         out.close();
         file.close();

         System.out.println("Object has been serialized");
       }
       catch(IOException ex)
      {
         System.out.println("IOException is caught");
      }


      Demo object1 = null;
```

```
        // Deserialization
        try
        {
            // Reading the object from a file
            FileInputStream file = new FileInputStream(filename);
            ObjectInputStream in = new ObjectInputStream(file);

            // Method for deserialization of object
            object1 = (Demo)in.readObject();

            in.close();
            file.close();

            System.out.println("Object has been deserialized ");
            System.out.println("a = " + object1.a);
            System.out.println("b = " + object1.b);
        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }

        catch(ClassNotFoundException ex)
        {
            System.out.println("ClassNotFoundException is caught");
        }

    }
}
```
Output :
```
Object has been serialized
Object has been deserialized
a = 1
b = geeksforgeeks
```

```
serialuid.java
import java.io.*;

class Emp implements Serializable {
private static final long serialversionUID = 129348938L;
   transient int a;
   static int b;
   String name;
   int age;

   // Default constructor
public Emp(String name, int age, int a, int b)
   {
      this.name = name;
      this.age = age;
      this.a = a;
      this.b = b;
   }

}

public class SerialExample {
public static void printdata(Emp object1)
   {
      System.out.println("name = " + object1.name);
      System.out.println("age = " + object1.age);
      System.out.println("a = " + object1.a);
      System.out.println("b = " + object1.b);
   }

public static void main(String[] args)
   {
      Emp object = new Emp("ab", 20, 2, 1000);
      String filename = "shubham.txt";

      // Serialization
      try {
          // Saving of object in a file
         FileOutputStream file = new FileOutputStream(filename);
         ObjectOutputStream out = new ObjectOutputStream (file);
          // Method for serialization of object
         out.writeObject(object);
         out.close();
         file.close();

         System.out.println("Object has been serialized\n"+ "Data before Deserialization.");
```

```
            printdata(object);

            // value of static variable changed
            object.b = 2000;
        }
          catch (IOException ex) {
            System.out.println("IOException is caught");
        }

        object = null;
          // Deserialization
        try {

            // Reading the object from a file
            FileInputStream file = new FileInputStream (filename);
            ObjectInputStream in = new ObjectInputStream (file);

            // Method for deserialization of object
            object = (Emp)in.readObject();

            in.close();
            file.close();
            System.out.println("Object has been deserialized\n"   + "Data after Deserialization.");
            printdata(object);

            // System.out.println("a = " + object1.a);
        }

        catch (IOException ex) {
            System.out.println("IOException is caught");
        }

        catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException" +
                                " is caught");
        }
    }
}
```

Output:

```
Object has been serialized
Data before Deserialization.
name = ab
age = 20
a = 2
b = 1000
Object has been deserialized
```

```
Data after Deserialization.
name = ab
age = 20
a = 0
b = 2000
```

Isaserialize.java

```
import java.io.Serializable;
class Person implements Serializable{
int id;
String name;
Person(int id, String name) {
this.id = id;
this.name = name;
}
}

class Student extends Person{
String course;
int fee;
public Student(int id, String name, String course, int fee) {
super(id,name);
this.course=course;
this.fee=fee;
}
}
```

Hasaserialize.java

```
class Address{
String addressLine,city,state;
public Address(String addressLine, String city, String state) {
this.addressLine=addressLine;
this.city=city;
this.state=state;
}
}

import java.io.Serializable;
public class Student implements Serializable{
int id;
String name;
Address address;//HAS-A
public Student(int id, String name) {
this.id = id;
this.name = name;
}
}
```

Since Address is not Serializable, you can not serialize the instance of Student class.

Transient.java

```
import java.io.Serializable;
public class Student implements Serializable{
int id;
String name;
transient int age;//Now it will not be serialized
public Student(int id, String name,int age) {
this.id = id;
this.name = name;
this.age=age;
}
}
```

Now write the code to serialize the object.

```
import java.io.*;
class PersistExample{
 public static void main(String args[])throws Exception{
  Student s1 =new Student(211,"ravi",22);//creating object
  //writing object into file
  FileOutputStream f=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(f);
  out.writeObject(s1);
  out.flush();

  out.close();
  f.close();
  System.out.println("success");
 }
}
```

Output:

```
success
```

Now write the code for deserialization.

```
import java.io.*;
class DePersist{
 public static void main(String args[])throws Exception{
  ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
  Student s=(Student)in.readObject();
  System.out.println(s.id+" "+s.name+" "+s.age);
  in.close();
 }
}
```

```
211 ravi 0
```

As you can see, printing age of the student returns 0 because value of age was not serialized.

Some more Programs

User Defined Exception

```
public class AgeDoesnotMatchException extends Exception{
  (String msg){
    super(msg);
  }

AgeDoesnotMatchException(String msg){
  super(msg);
}

public String toString(){
  return "CustomException[Age is not between 17 and 24]";
}
}

public class Student extends RuntimeException {
  private String name;
  private int age;
  public Student(String name, int age){
    try {
      if (age<17||age>24) {
        String msg = "Age is not between 17 and 24";
        AgeDoesnotMatchException ex = new AgeDoesnotMatchException(msg);
        throw ex;
      }
    }
    catch(AgeDoesnotMatchException e) {
      e.printStackTrace();
    }
    this.name = name;
    this.age = age;
  }

  public void display(){
    System.out.println("Name of the Student: "+this.name );
    System.out.println("Age of the Student: "+this.age );
  }

  public static void main(String args[]) {
    Scanner sc= new Scanner(System.in);
    System.out.println("Enter the name of the Student: ");
    String name = sc.next();
    System.out.println("Enter the age of the Student should be 17 to 24
    (including 17 and 24): ");
    int age = sc.nextInt();
    Student obj = new Student(name, age);
    obj.display();
  }
}
```

User defined Exception

```
class InvalidProductException extends Exception
{
   public InvalidProductException(String s)
   {
      // Call constructor of parent Exception
      super(s);
   }
}

public class Example1
{
  void productCheck(int weight) throws InvalidProductException{
         if(weight<100){
                  throw new InvalidProductException("Product Invalid");
         }
  }

   public static void main(String args[])
   {
         Example1 obj = new Example1();
      try
      {
         obj.productCheck(60);
      }
      catch (InvalidProductException ex)
      {
         System.out.println("Caught the exception");
         System.out.println(ex.getMessage());
      }
   }
}
```

Overloaded Exception

```
public class Main {
  double method(int i) throws Exception {
    return i/0;
  }
  boolean method(boolean b) {
    return !b;
  }
  static double method(int x, double y) throws Exception {
    return x + y ;
  }
  static double method(double x, double y) {
    return x + y - 3;
  }
  public static void main(String[] args) {
    Main mn = new Main();
    try {
      System.out.println(method(10, 20.0));
      System.out.println(method(10.0, 20));
      System.out.println(method(10.0, 20.0));
      System.out.println(mn.method(10));
    } catch (Exception ex) {
      System.out.println("exception occoure: "+ ex);
    }
  }
}
```

Result

```
30.0
27.0
27.0
exception occoure: java.lang.ArithmeticException: / by zero
```

print stack of the Exception in Java.

```
public class Demo {
  public static void main(String[] args) {
    try {
      ExceptionFunc();
    } catch(Throwable e) {
      e.printStackTrace();
    }
  }
  public static void ExceptionFunc() throws Throwable {
    Throwable t = new Throwable("This is new Exception in Java...");

    StackTraceElement[] trace = new StackTraceElement[] {
      new StackTraceElement("ClassName","methodName","fileName",5)
    };
    t.setStackTrace(trace);
    throw t;
  }
}
```

result.

```
java.lang.Throwable: This is new Exception in Java...
        at ClassName.methodName(fileName:5)
```

checked exception using catch block.

```
public class Main {
   public static void main (String args[]) {
      try {
         throw new Exception("throwing an exception");
      } catch (Exception e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Result

```
throwing an exception
```

If base class doesn’t throw any exception but child class throws an unchecked exception.
In this example class Room is overriding the method color(). The overridden method is not throwing any exception however the overriding method is throwing an unchecked exception (NullPointerException). Upon compilation code ran successfully.

```
class Building {
  void color()
  {
     System.out.println("Blue");
  }
}
class Room extends Building{
  //It throws an unchecked exception
  void color() throws NullPointerException
  {
     System.out.println("White");
  }
  public static void main(String args[]){
     Building obj = new Room();
     obj.color();
  }
}
```

Output:

```
White
```

Example 2: If base class doesn’t throw any exception but child class throws an checked exception

```
import java.io.*;
class Building {
  void color()
  {
    System.out.println("Blue");
  }
}
class Room extends Building{
  void color() throws IOException
  {
    System.out.println("White");
  }
  public static void main(String args[]){
    Building obj = new Room();
    try{
      obj.color();
    }catch(Exception e){
      System.out.println(e);
     }
  }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Exception IOException is not compatible with throws clause in Building.color()
```

The above code is having a compilation error: Because the overriding method (child class method) cannot throw a checked exception if the overridden method(method of base class) is not throwing an exception.

Example 3: When base class and child class both throws a checked exception

```
import java.io.*;
class Building {
  void color() throws IOException
  {
     System.out.println("Blue");
  }
}
class Room extends Building{
   void color() throws IOException
   {
      System.out.println("White");
   }
   public static void main(String args[]){
      Building obj = new Room();
      try{
           obj.color();
         }catch(Exception e){
           System.out.println(e);
          }
   }
}
```

Output:

```
White
```

The code ran fine because color() method of child class is NOT throwing a checked exception with scope broader than the exception declared by color() method of base class.

Example 4: When child class method is throwing border checked exception compared to the same method of base class

```
package beginnersbook.com;
import java.io.*;
class Building {
        void color() throws IOException
        {
                System.out.println("Blue");
        }
}
class Room extends Building{
         void color() throws Exception
         {
                System.out.println("White");
         }
         public static void main(String args[]){
                 Building obj = new Room();
                 try{
                 obj.color();
                 }catch(Exception e){
                         System.out.println(e);
                 }
         }
}
```

Output:
Compilation error because the color() method of child class is throwing Exception which has a broader scope than the exception thrown by method color() of parent class.

```
class Count extends Thread
{
  Count()
  {
    super("my extending thread");
    System.out.println("my thread created" + this);
    start();
  }
  public void run()
  {
    try
    {
      for (int i=0 ;i<10;i++)
      {
        System.out.println("Printing the count " + i);
        Thread.sleep(1000);
      }
    }
    catch(InterruptedException e)
    {
      System.out.println("my thread interrupted");
    }
    System.out.println("My thread run is over" );
  }
}
class ExtendingExample
{
  public static void main(String args[])
  {
     Count cnt = new Count();
     try
     {
       while(cnt.isAlive())
       {
        System.out.println("Main thread will be alive till the child thread is live");
        Thread.sleep(1500);
       }
     }
     catch(InterruptedException e)
     {
      System.out.println("Main thread interrupted");
     }
     System.out.println("Main thread's run is over" );
  }
}
```
Output:

```
my thread createdThread[my runnable thread,5,main]
Main thread will be alive till the child thread is live
Printing the count 0
Printing the count 1
Main thread will be alive till the child thread is live
Printing the count 2
Main thread will be alive till the child thread is live
Printing the count 3
Printing the count 4
Main thread will be alive till the child thread is live
Printing the count 5
Main thread will be alive till the child thread is live
Printing the count 6
Printing the count 7
Main thread will be alive till the child thread is live
Printing the count 8
Main thread will be alive till the child thread is live
Printing the count 9
mythread run is over
Main thread run is over
```

```
//This class' shared object will be accessed by threads
class LoopValues implements Runnable{

   @Override
   public void run() {
 System.out.println(Thread.currentThread().getName() +
          " Priority is " + Thread.currentThread().getPriority());
      for (int i = 1; i <= 10; i++) {
         System.out.println(Thread.currentThread().getName() + " : " + i);
      }
   }
}

public class ThreadPriorityDemo {
   public static void main(String[] args) {
      Thread thread1 = new Thread(new LoopValues(), "Thread-1");
      Thread thread2 = new Thread(new LoopValues(), "Thread-2");
      thread1.setPriority(Thread.MAX_PRIORITY);
      thread2.setPriority(Thread.MIN_PRIORITY);
      thread1.start();
      thread2.start();
      try {
         //Wait for the threads to finish
         thread1.join();
         thread2.join();

      } catch (InterruptedException ex) {
         ex.printStackTrace();
      }
      System.out.println("Done with looping values");
   }
}
```
Output

```
Thread-1 Priority is 10
Thread-1 : 1
Thread-1 : 2
Thread-1 : 3
Thread-1 : 4
Thread-1 : 5
Thread-1 : 6
Thread-1 : 7
Thread-1 : 8
Thread-1 : 9
Thread-1 : 10
Thread-2 Priority is 1
```

```
Thread-2 : 1
Thread-2 : 2
Thread-2 : 3
Thread-2 : 4
Thread-2 : 5
Thread-2 : 6
Thread-2 : 7
Thread-2 : 8
Thread-2 : 9
Thread-2 : 10
Done with looping values
```

Thred Intrupt

```
class Intr implements Runnable         //Implementing the Runnable interface
{

public void run()                  //Entry point of new thread
{
for(int i=0;i<5;i++)
{
        System.out.println(i);
        if (Thread.interrupted())  // Checking interrupt status
                System.out.println("Thread was interrupted");
        }
}

public static void main(String... ar)
{
Intr newTh= new Intr();
Thread th= new Thread(newTh,"Thread2");     //Calling Thread's constructor & passing the
object of class that  implemented  Runnable interface & the name of new thread.
th.start();          //Starts the thread.
th.interrupt();   //Interrupts the thread while it is running
}
}
```

Output-

```
0
Thread was interrupted
1
2
3
4
```

```
package threadpriority;



class MyThread1 extends Thread {
	public void run() {
		System.out.println(Thread.currentThread().getPriority());
	}
}

public class ThreadPriorityExample {
	public static void main(String[] args) {
		MyThread1 thread1 = new MyThread1();
		MyThread1 thread2 = new MyThread1();
		MyThread1 thread3 = new MyThread1();
		thread1.setPriority(6);
		thread2.setPriority(7);
		thread3.setPriority(8);
		thread1.start();
		thread2.start();
		thread3.start();
	}
}
```
Output is –

```
6
7
8
```

```
package multithreading;

class Thread1 extends Thread {
	public void run() {
		for (int i = 0; i < 5; i++) {

System.out.println("Thread1 created with name: - " + Thread.currentThread().getName());
		Thread.yield();
	}

	}
}

public class ThreadYieldExample1 {
	public static void main(String[] args) {
		Thread1 t1 = new Thread1();
		t1.start();

		for (int i = 0; i < 5; i++) {
			System.out.println("Main created with name: - " +
Thread.currentThread().getName());

		}
	}
}
```

What is expected output here? As we discussed in the beginning since we are calling Thread.yeild() for t1, t1 will request to operating system hey can you ask main thread to execute. So the expected output is –

```
Thread1 created with name: – Thread-0
Main created with name: – main
Main created with name: – main
Main created with name: – main
Main created with name: – main
Main created with name: – main
Thread1 created with name: – Thread-0
Thread1 created with name: – Thread-0
Thread1 created with name: – Thread-0
Thread1 created with name: – Thread-0
```

```
// Import the File class
import java.io.File;

// Import this class for handling errors
import java.io.FileNotFoundException;

// Import the Scanner class to read content from text files
import java.util.Scanner;

public class GFG {
        public static void main(String[] args)
        {
                try {
                        File Obj = new File("myfile.txt");
                        Scanner Reader = new Scanner(Obj);
                        while (Reader.hasNextLine()) {
                                String data = Reader.nextLine();
                                System.out.println(data);
                        }
                        Reader.close();
                }
                catch (FileNotFoundException e) {
                        System.out.println("An error has occurred.");
                        e.printStackTrace();
                }
```

```
          }
}

// Import the FileWriter class
import java.io.FileWriter;

// Import the IOException class for handling errors
import java.io.IOException;

public class GFG {
          public static void main(String[] args)
          {
                    try {
                              FileWriter Writer
                                        = new FileWriter("myfile.txt");
                              Writer.write(
                                        "Files in Java are seriously good!!");
                              Writer.close();
                              System.out.println("Successfully written.");
                    }
                    catch (IOException e) {
                              System.out.println("An error has occurred.");
                              e.printStackTrace();
                    }
          }}
```

Create statement/execute query/

Example.java

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/EMP";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

  public static void main(String[] args) {
  Connection conn = null;
  Statement stmt = null;
  try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");
```

```
//Class.forName(JDBC_DRIVER);



        //STEP 3: Open a connection
        System.out.println("Connecting to database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
        // conn = DriverManager.getConnection(DB_URL, “”, “”);

        //STEP 4: Execute a query
        System.out.println("Creating statement...");
        stmt = conn.createStatement();
        String sql = "UPDATE Employees set age=30 WHERE id=103";

        // Let us check if it returns a true Result Set or not.
        Boolean ret = stmt.execute(sql);
        System.out.println("Return value is : " + ret.toString() );

        // Let us update age of the record with ID = 103;
        int rows = stmt.executeUpdate(sql);
        System.out.println("Rows impacted : " + rows );
```

```
          // Let us select all the records and display them.

          sql = "SELECT id, first, last, age FROM Employees";

          ResultSet rs = stmt.executeQuery(sql);


Int age, id;

String first, last;

          //STEP 5: Extract data from result set

          while(rs.next()){

          //Retrieve by column name

          id  = rs.getInt("id");

             age = rs.getInt("age");

          first = rs.getString("first");

          last = rs.getString("last");


          //Display values

          System.out.print("ID: " + id);

          System.out.print(", Age: " + age);

          System.out.print(", First: " + first);

          System.out.println(", Last: " + last);

          }

          //STEP 6: Clean-up environment
```

```
        rs.close();
        stmt.close();
        conn.close();
}catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
}catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
}finally{
        //finally block used to close resources
        try{
        if(stmt!=null)
        stmt.close();
        }catch(SQLException se2){
        }// nothing we can do
        try{
        if(conn!=null)
        conn.close();
        }catch(SQLException se){
        se.printStackTrace();
        }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

Output

```
Connecting to database...
Creating statement...
Return value is : false
Rows impacted : 1
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 30, First: Sumit, Last: Mittal
Goodbye!
```

Create statement execute update query

```
//STEP 1. Import required packages
import java.sql.*;
```

```
public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/EMP";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

public static void main(String[] args) {
  Connection conn = null;
  Statement stmt = null;
  try{
        //STEP 2: Register JDBC driver
     Class.forName("com.mysql.jdbc.Driver");

        //STEP 3: Open a connection
        System.out.println("Connecting to database...");
        conn = DriverMana ger.getConnection(DB_URL,USER,PASS);
```

```
//STEP 4: Execute a query to create statement with required arguments for RS
example.

        System.out.println("Creating statement...");

        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,

                        ResultSet.CONCUR_READ_ONLY);

        String sql;

        sql = "SELECT id, first, last, age FROM Employees";

        ResultSet rs = stmt.executeQuery(sql);


        // Move cursor to the last row.

        System.out.println("Moving cursor to the last...");

        rs.last();


        //STEP 5: Extract data from result set

        System.out.println("Displaying record...");

        //Retrieve by column name

        String id  = rs.getString("id");

System.out.println("id is “+id);

 //Int id1=Integer.parseInt(id);

        int age = rs.getInt("age");

        String first = rs.getString("first");

        String last = rs.getString("last");
```

```
//Display values
System.out.print("ID: " + id);
System.out.print(", Age: " + age);
System.out.print(", First: " + first);
System.out.println(", Last: " + last);

// Move cursor to the first row.
System.out.println("Moving cursor to the first row...");
rs.first();

//STEP 6: Extract data from result set
System.out.println("Displaying record...");
//Retrieve by column name
id  = rs.getInt("id");
age = rs.getInt("age");
first = rs.getString("first");
last = rs.getString("last");

//Display values
System.out.print("ID: " + id);
System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);

// Move cursor to the first row.


System.out.println("Moving cursor to the next row...");

rs.next();


//STEP 7: Extract data from result set

System.out.println("Displaying record...");

id  = rs.getInt("id");

age = rs.getInt("age");

first = rs.getString("first");

last = rs.getString("last");


//Display values

  System.out.print("ID: " + id);

System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);
```

```
     //STEP 8: Clean-up environment

     rs.close();

     stmt.close();

     conn.close();

}catch(SQLException se){

     //Handle errors for JDBC

     se.printStackTrace();

}catch(Exception e){

     //Handle errors for Class.forName

     e.printStackTrace();

}finally{

     //finally block used to close resources

     try{

     if(stmt!=null)

     stmt.close();

     }catch(SQLException se2){

     }// nothing we can do

     try{

     if(conn!=null)

     conn.close();

     }catch(SQLException se){
         se.printStackTrace();
         }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

Output

```
Connecting to database...
Creating statement...
Moving cursor to the last...
Displaying record...
ID: 103, Age: 30, First: Sumit, Last: Mittal
Moving cursor to the first row...
Displaying record...
ID: 100, Age: 18, First: Zara, Last: Ali
Moving cursor to the next row...
Displaying record...
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
Goodbye!
```

Exampleupdate.java

```
//STEP 1. Import required packages

import java.sql.*;


public class JDBCExample {

   // JDBC driver name and database URL

   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";

   static final String DB_URL = "jdbc:mysql://localhost/EMP";


   //  Database credentials

   static final String USER = "username";

   static final String PASS = "password";


 public static void main(String[] args) {

   Connection conn = null;

   try{

         //STEP 2: Register JDBC driver

      Class.forName("com.mysql.jdbc.Driver");


         //STEP 3: Open a connection
         System.out.println("Connecting to database...");
         conn = DriverManager.getConnection(DB_URL,USER,PASS);


         //STEP 4: Execute a query to create statment with
         // required arguments for RS example.
         System.out.println("Creating statement...");
         Statement stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_UPDATABLE);


 //STEP 5: Execute a query
         String sql = "SELECT id, first, last, age FROM Employees";
         ResultSet rs = stmt.executeQuery(sql);
         System.out.println("List result set for reference....");


         printRs(rs);


         //STEP 6: Loop through result set and add 5 in age
         //Move to BFR postion so while-loop works properly
         rs.beforeFirst();
         //STEP 7: Extract data from result set
         while(rs.next()){
```

```
    //Retrieve by column name
    int newAge = rs.getInt("age") + 5;
    rs.updateDouble( "age", newAge );
    rs.updateRow();
    }
    System.out.println("List result set showing new ages...");
    printRs(rs);
    // Insert a record into the table.
    //Move to insert row and add column data with updateXXX()
    System.out.println("Inserting a new record...");
    rs.moveToInsertRow();
    rs.updateInt("id",104);
rs.updateString("first","John");
rs.updateString("last","Paul");
    rs.updateInt("age",40);
    //Commit row
    rs.insertRow();
    System.out.println("List result set showing new set...");
    printRs(rs);
    // Delete second record from the table. Set position to second record first
    rs.absolute( 2 );
```

```
System.out.println("List the record before deleting...");

//Retrieve by column name

int id  = rs.getInt("id");

int age = rs.getInt("age");

String first = rs.getString("first");

String last = rs.getString("last");

//Display values

System.out.print("ID: " + id);

System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);


//Delete row

rs.deleteRow();

System.out.println("List result set after deleting one records...");

printRs(rs);


//STEP 8: Clean-up environment

rs.close();

stmt.close();

conn.close();
```

```
   }catch(SQLException se){
         //Handle errors for JDBC
         se.printStackTrace();
   }catch(Exception e){
         //Handle errors for Class.forName
         e.printStackTrace();
   }finally{
         //finally block used to close resources
         try{
         if(conn!=null)
         conn.close();
         }catch(SQLException se){
         se.printStackTrace();
         }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main

   public static void printRs(ResultSet rs) throws SQLException{
         //Ensure we start with first row
         rs.beforeFirst();
```

```
        while(rs.next()){
        //Retrieve by column name
        int id  = rs.getInt("id");
        int age = rs.getInt("age");
        String first = rs.getString("first");
        String last = rs.getString("last");

        //Display values
        System.out.print("ID: " + id);
        System.out.print(", Age: " + age);
        System.out.print(", First: " + first);
       System.out.println(", Last: " + last);
        }
        System.out.println();
  }//end printRs()
}//end JDBCExample
```

Output:

```
Connecting to database...
Creating statement...
```

```
List result set for reference....

ID: 100, Age: 33, First: Zara, Last: Ali

ID: 101, Age: 40, First: Mahnaz, Last: Fatma

ID: 102, Age: 50, First: Zaid, Last: Khan

ID: 103, Age: 45, First: Sumit, Last: Mittal


List result set showing new ages...

ID: 100, Age: 38, First: Zara, Last: Ali

ID: 101, Age: 45, First: Mahnaz, Last: Fatma

ID: 102, Age: 55, First: Zaid, Last: Khan

ID: 103, Age: 50, First: Sumit, Last: Mittal


Inserting a new record...

List result set showing new set...

ID: 100, Age: 38, First: Zara, Last: Ali

ID: 101, Age: 45, First: Mahnaz, Last: Fatma

ID: 102, Age: 55, First: Zaid, Last: Khan

ID: 103, Age: 50, First: Sumit, Last: Mittal

ID: 104, Age: 40, First: John, Last: Paul


List the record before deleting...
```

```
ID: 101, Age: 45, First: Mahnaz, Last: Fatma

List result set after deleting one records...

ID: 100, Age: 38, First: Zara, Last: Ali

ID: 102, Age: 55, First: Zaid, Last: Khan

ID: 103, Age: 50, First: Sumit, Last: Mittal

ID: 104, Age: 40, First: John, Last: Paul


Goodbye!
```

Resultmetadata.java

```
import java.sql.*;

class Rsmd{

public static void main(String args[]){

try{

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

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


PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();

ResultSetMetaData rsmd=rs.getMetaData();


System.out.println("Total columns: "+rsmd.getColumnCount());

System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));

System.out.println("Column Type Name of 1st column:
"+rsmd.getColumnTypeName(1));


con.close();

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

}

}
```

```
Output: Total columns: 2

        Column Name of 1st column: ID

        Column Type Name of 1st column: NUMBER
```

Example1.java

```
import java.sql.*;

class OracleCon{

public static void main(String args[]){

try{

//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");


//step2 create  the connection object

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


//step3 create the statement object

Statement stmt=con.createStatement();


//step4 execute query

ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next())

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));


//step5 close the connection object

con.close();


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


}

}
```

Mysql create statement

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/db_name","root","root");
//here db_name is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

Connection with Access withoutdsn.java

```
import java.sql.*;
class Test{
public static void main(String ar[]){
 try{
   String database="student.mdb";//Here database exists in the current directory

   String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
                 DBQ=" + database + ";DriverID=22;READONLY=true";

   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection c=DriverManager.getConnection(url);
   Statement st=c.createStatement();
   ResultSet rs=st.executeQuery("select * from login");

   while(rs.next()){
          System.out.println(rs.getString(1));
   }
}catch(Exception ee){System.out.println(ee);}

}}
```

Connection with Access Withdsn.java

```
import java.sql.*;
class Test{
public static void main(String ar[]){
 try{
   String url="jdbc:odbc:mydsn";
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection c=DriverManager.getConnection(url);
   Statement st=c.createStatement();
   ResultSet rs=st.executeQuery("select * from login");

   while(rs.next()){
          System.out.println(rs.getString(1));
   }
}catch(Exception ee){System.out.println(ee);}

}}
```

Statementinterface.java

```
import java.sql.*;

class FetchRecord{

public static void main(String args[])throws Exception{

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

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

Statement stmt=con.createStatement();


int result=stmt.executeUpdate("delete from emp765 where id=33");

System.out.println(result+" records affected");

con.close();

}}
```

Resultset.java

```
import java.sql.*;

class FetchRecord{

public static void main(String args[])throws Exception{


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

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

Statement  stmt
=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCU
R_UPDATABLE);

ResultSet rs=stmt.executeQuery("select * from emp765");


//getting the record of 3rd row

rs.absolute(3);

System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));


con.close();
}}
```

Preparedstatement.java

```
create table emp(id number(10),name varchar2(50));



import java.sql.*;

class InsertPrepared{

public static void main(String args[]){

try{

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


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


PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");

stmt.setInt(1,101);

//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");


int i=stmt.executeUpdate();

System.out.println(i+" records inserted");


con.close();


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


}

}
```

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where
id=?");

stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name

stmt.setInt(2,101);


int i=stmt.executeUpdate();

System.out.println(i+" records updated");
```

insert records until user press n

```
import java.sql.*;

import java.io.*;

class RS{

public static void main(String args[])throws Exception{

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

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


PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");


BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

Int id;

String name, s;

Float salary;

do{

System.out.println("enter id:");
id=Integer.parseInt(br.readLine());

System.out.println("enter name:");

name=br.readLine();

System.out.println("enter salary:");

salary=Float.parseFloat(br.readLine());


ps.setInt(1,id);

ps.setString(2,name);

ps.setFloat(3,salary);

int i=ps.executeUpdate();

System.out.println(i+" records affected");


System.out.println("Do you want to continue: y/n");

s=br.readLine();

if(s.startsWith("n")){

break;

}

}while(true);


con.close();

}}
```

Try catch block

```
importjava.sql.*;

importjava.util.*;

class Main

{

        public static void main(String a[])

        {

        //Creating the connection

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

        String user = "system";

        String pass = "12345";


        //Entering the data

        Scanner k = new Scanner(System.in);

        System.out.println("enter name");

        String name = k.next();

        System.out.println("enter roll no");

        int roll = k.nextInt();
```

```
System.out.println("enter class");

String cls =  k.next();


//Inserting data using SQL query

String sql = "insert into student1 values('"+name+"',"+roll+",'"+cls+"')";

Connection con=null;

try

{

DriverManager.registerDriver(new oracle.jdbc.OracleDriver());


//Reference to connection interface

con = DriverManager.getConnection(url,user,pass);


Statement st = con.createStatement();

int m = st.executeUpdate(sql);

if (m == 1)

    System.out.println("inserted successfully : "+sql);

else

    System.out.println("insertion failed");

con.close();

}
```

```
    catch(Exception ex)
    {
    System.err.println(ex);
    }
    }
}
```

Prepare and Create

```
import java.sql.*;

public class jdbcConn {
  public static void main(String[] args) throws Exception {
    Class.forName("org.apache.derby.jdbc.ClientDriver");
       Connection con = DriverManager.getConnection (
      "jdbc:derby://localhost:1527/testDb","name","pass");
       PreparedStatement updateemp = con.prepareStatement(
          "insert into emp values(?,?,?)");


          updateemp.setInt(1,23);
       updateemp.setString(2,"Roshan");
          updateemp.setString(3, "CEO");
          updateemp.executeUpdate();


          Statement stmt = con.createStatement();
          String query = "select * from emp";
          ResultSet rs =  stmt.executeQuery(query);
            System.out.println("Id Name   Job");


          while (rs.next()) {
          int id = rs.getInt("id");
          String name = rs.getString("name");
          String job = rs.getString("job");
          System.out.println(id + "   " + name+"    "+job);
       }
    }
 }
```

Result

The above code sample will produce the following result. The result may vary.

```
Id Name      Job
23  Roshan   CEO
```

Callable:

```
create or replace procedure "INSERTR"
(id IN NUMBER, name IN VARCHAR2)  is
begin
insert into user420 values(id,name);
end;
/
```

```
import java.sql.*;
public class Proc {
public static void main(String[] args) throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
stmt.setInt(1,1011);
stmt.setString(2,"Amit");
stmt.execute();

System.out.println("success");
}
}
```

Callable statement

```
CallableStatement callableStatement =
        connection.prepareCall("{call calculateStatistics(?, ?)}",
        ResultSet.TYPE_FORWARD_ONLY,
        ResultSet.CONCUR_READ_ONLY,
        ResultSet.CLOSE_CURSORS_OVER_COMMIT
        );
```

Callable statement with parameter

```
CallableStatement callableStatement =
        connection.prepareCall("{call calculateStatistics(?, ?)}");

callableStatement.setString(1, "param1");
callableStatement.setInt  (2, 123);
```

Batch Updates

```
CallableStatement callableStatement =
        connection.prepareCall("{call calculateStatistics(?, ?)}");

callableStatement.setString(1, "param1");
callableStatement.setInt   (2, 123);
callableStatement.addBatch();

callableStatement.setString(1, "param2");
callableStatement.setInt   (2, 456);
callableStatement.addBatch();

int[] updateCounts = callableStatement.executeBatch();
```

OUT Parameters

```
CallableStatement callableStatement =
        connection.prepareCall("{call calculateStatistics(?, ?)}");

callableStatement.setString(1, "param1");
callableStatement.setInt (2, 123);

callableStatement.registerOutParameter(1, java.sql.Types.VARCHAR);
callableStatement.registerOutParameter(2, java.sql.Types.INTEGER);

ResultSet result = callableStatement.executeQuery();
while(result.next()) { ... }

String out1 = callableStatement.getString(1);
int    out2 = callableStatement.getInt   (2);
```

Simple example of transaction management in jdbc using Statement

```
import java.sql.*;

class FetchRecords{

public static void main(String args[])throws Exception{

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

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

con.setAutoCommit(false);


Statement stmt=con.createStatement();

stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");

stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");


con.commit();

con.close();

}}
```

Example of batch processing

```
import java.sql.*;

class FetchRecords{

public static void main(String args[])throws Exception{
```

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

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

con.setAutoCommit(false);


Statement stmt=con.createStatement();

stmt.addBatch("insert into user420 values(190,'abhi',40000)");

stmt.addBatch("insert into user420 values(191,'umesh',50000)");


stmt.executeBatch();//executing the batch


con.commit();

con.close();

}}
```

Example of batch processing using PreparedStatement

```
import java.sql.*;

import java.io.*;

class BP{
```

```
public static void main(String args[]){

try{


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

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


PreparedStatement ps=con.prepareStatement("insert into user420 values(?,?,?)");


BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

while(true){


System.out.println("enter id");

String s1=br.readLine();

int id=Integer.parseInt(s1);


System.out.println("enter name");

String name=br.readLine();


System.out.println("enter salary");

String s3=br.readLine();
int salary=Integer.parseInt(s3);


ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);


ps.addBatch();
System.out.println("Want to add more records y/n");
String ans=br.readLine();
if(ans.equals("n")){
break;
}


}
ps.executeBatch();


System.out.println("record successfully saved");


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

}}

//out of syllabus

Rowset in Recordset

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
        public static void main(String[] args) throws Exception {
                Class.forName("oracle.jdbc.driver.OracleDriver");

        //Creating and Executing RowSet
        JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
   rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        rowSet.setUsername("system");
```

```
    rowSet.setPassword("oracle");


        rowSet.setCommand("select * from emp400");

        rowSet.execute();


        //Adding Listener and moving RowSet

        rowSet.addRowSetListener(new MyListener());


            while (rowSet.next()) {

            // Generating cursor Moved event

              System.out.println("Id: " + rowSet.getString(1));

              System.out.println("Name: " + rowSet.getString(2));

              System.out.println("Salary: " + rowSet.getString(3));

            }


        }

}


class MyListener implements RowSetListener {

        public void cursorMoved(RowSetEvent event) {

                System.out.println("Cursor Moved...");
        }
        public void rowChanged(RowSetEvent event) {
                System.out.println("Cursor Changed...");
        }
        public void rowSetChanged(RowSetEvent event) {
                System.out.println("RowSet changed...");
        }
}
```

The output is as follows:

```
Cursor Moved...
Id: 55
Name: Om Bhim
Salary: 70000
Cursor Moved...
Id: 190
Name: abhi
Salary: 40000
Cursor Moved...
Id: 191
Name: umesh
```

```
Salary: 50000

Cursor Moved...
```