

Java Internal Lab 2

Programs on Exception Handling :

Exception Handling:

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

Program 1:(Demo.java):

```
class Demo{

    public static void main(String args[]){

        try{

            int a=args.length;

            System.out.println(a);

            int b=42/a;

            int c[]={1};

            c[42]=99;

        }

        catch(ArithmeticException ae){

            System.out.println(ae);

        }

        catch(ArrayIndexOutOfBoundsException ae1){

            System.out.println(ae1);

        }

        catch(Exception e){
```

```

        System.out.println(e);
    }

    System.out.println("After try catch block");

}

}

```

Output:

>javac Demo.java

```

C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java Demo Firstarg
1
java.lang.ArrayIndexOutOfBoundsException: Index 42 out of bounds for length 1
After try catch block

C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java Demo
0
java.lang.ArithmeticException: / by zero
After try catch block

```

NestedTry:

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, the **inner try block** can be used to handle **ArrayIndexOutOfBoundsException** while the **outer try block** can handle the **ArithmeticException** (division by zero).

Program:(NestedTry.java):

```

class NestedTry{

    public static void main(String args[]){

        try{

```

```
int a=args.length;

int b=42/a;

System.out.println("a = "+a);

try{

if(a==1){

a=a/(a-a);

}

if(a==2){

int c[]={1};

c[42]=99;

}

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println(e);

}

}

catch(ArithmeticException e1){

System.out.println(e1);

}

}

}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac NestedTry.java

C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java NestedTry Firstarg Secondarg
a = 2
java.lang.ArrayIndexOutOfBoundsException: Index 42 out of bounds for length 1

C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java NestedTry
java.lang.ArithmeticException: / by zero
```

Threads in java:

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

Thread Priority programs:

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses. Note that not only JVM a Java programmer can also assign the priorities of a thread explicitly in a Java program.

Program:(ThreadPriorityExample.java):

```
import java.lang.*;

public class ThreadPriorityExample extends Thread
{
    public static void main(String argsv[])
    {
        ThreadPriorityExample th1 = new ThreadPriorityExample();
```

```
ThreadPriorityExample th2 = new ThreadPriorityExample();

ThreadPriorityExample th3 = new ThreadPriorityExample();


System.out.println("Priority of the thread th1 is : " + th1.getPriority());

System.out.println("Priority of the thread th2 is : " + th2.getPriority());

System.out.println("Priority of the thread th2 is : " + th2.getPriority());


th1.setPriority(6);

th2.setPriority(3);

th3.setPriority(9);


System.out.println("Priority of the thread th1 is : " + th1.getPriority());

    System.out.println("Priority of the thread th2 is : " + th2.getPriority());

System.out.println("Priority of the thread th3 is : " + th3.getPriority());

    System.out.println("Currently Executing The Thread : " + Thread.currentThread().getName());

    System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority());

    Thread.currentThread().setPriority(10);

    System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority());

}

}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac ThreadPriorityExample.java
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java ThreadPriorityExample
Priority of the thread th1 is : 5
Priority of the thread th2 is : 5
Priority of the thread th2 is : 5
Priority of the thread th1 is : 6
Priority of the thread th2 is : 3
Priority of the thread th3 is : 9
Currently Executing The Thread : main
Priority of the main thread is : 5
Priority of the main thread is : 10
```

Program:(ThreadPriority.java):

```
public class ThreadPriority extends Thread
{
    public void run ()
    {
        System.out.println ("running thread name is:" + Thread.currentThread ().getName ());
        System.out.println ("running thread priority is:" + Thread.currentThread ().getPriority ());
    }

    public static void main (String args[])
    {
        ThreadPriority m1 = new ThreadPriority ();

        ThreadPriority m2 = new ThreadPriority ();

        m1.setPriority (Thread.MIN_PRIORITY);

        m2.setPriority (Thread.MAX_PRIORITY);

        m1.start ();

        m2.start ();

    }
}
```

```
}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac ThreadPriority.java
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java ThreadPriority
running thread name is:Thread-0
running thread priority is:1
running thread name is:Thread-1
running thread priority is:10
```

Inter Thread Communications:(very very important):

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

Program:(InterThreadCommunication.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:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac InterThreadCommunication.java
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java Test
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed...
```

Generics in Java :

Generics means parameterized types. The idea is to allow type (Integer, String, ... etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Programs on generics :

Program1:(GenricFirstProgram.java):

```
// Java program to show working of user defined
```

```
// Generic classes
```

```
// We use < > to specify Parameter type
```

```
class Test<T> {
```

```
// An object of type T is declared
```

```
T obj;
```

```
Test(T obj) { this.obj = obj; } // constructor
```

```
public T getObject() { return this.obj; }
```

```
}
```

```
// Driver class to test above
```

```
class Main {
```

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

```
{
```

```
// instance of Integer type
```

```
Test<Integer> iObj = new Test<Integer>(15);
```

```
System.out.println(iObj.getObject());
```

```
// instance of String type
```

```
Test<String> sObj
```

```
= new Test<String>("GeeksForGeeks");

System.out.println(sObj.getObject());

}

}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac GenricFirstProgram.java
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java Main
15
GeeksForGeeks
```

Program2(Generic2.java):

```
// Java program to show multiple
// type parameters in Java Generics

// We use < > to specify Parameter type

class Test<T, U>

{

    T obj1; // An object of type T

    U obj2; // An object of type U
```

```
// constructor
```

```
Test(T obj1, U obj2)
```

```
{
```

```
    this.obj1 = obj1;
```

```
    this.obj2 = obj2;
```

```
}
```

```
// To print objects of T and U
```

```
public void print()
```

```
{
```

```
    System.out.println(obj1);
```

```
    System.out.println(obj2);
```

```
}
```

```
}
```

```
// Driver class to test above
```

```
class Main
```

```
{
```

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

```
{
```

```
Test <String, Integer> obj =  
new Test<String, Integer>("GfG", 15);  
  
obj.print();  
  
}  
  
}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac Generic2.java  
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java Main  
GfG  
15
```

Program3:(Generic_Functions.java):

```
// Java program to show working of user defined
```

```
// Generic functions
```

```
class Test {
```

```
// A Generic method example
```

```
static <T> void genericDisplay(T element)
```

```
{
```

```
System.out.println(element.getClass().getName())
```

```
+ " = " + element);  
  
}  
  
// Driver method  
  
public static void main(String[] args)  
{  
  
    // Calling generic method with Integer argument  
  
    genericDisplay(11);  
  
    // Calling generic method with String argument  
  
    genericDisplay("GeeksForGeeks");  
  
    // Calling generic method with double argument  
  
    genericDisplay(1.0);  
  
}  
  
}
```

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac Generic_Functions.java  
  
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java Test  
java.lang.Integer = 11  
java.lang.String = GeeksForGeeks  
java.lang.Double = 1.0
```

Collections in Java:

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](#), Vector, [LinkedList](#), [PriorityQueue](#), HashSet, LinkedHashSet, TreeSet)

ArrayList:

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.

Program:(TestJavaCollection1.java):

```
import java.util.*;

class TestJavaCollection1{
    public static void main(String args[]){
        ArrayList<String> list=new ArrayList<String>();
        list.add("Honey");
        list.add("Pavan");
        list.add("Ben10");
        list.add("Doremon");
        Iterator itr=list.iterator();
        while(itr.hasNext()){
```

```
System.out.println(itr.next());  
}  
}  
}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac TestJavaCollection1.java  
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java TestJavaCollection1  
Honey  
Pavan  
Ben10  
Doremon
```

LinkedList:

LinkedList implements the collection interface it uses a doubly linked list internally to store the elements.it can store the duplicate elements.it maintains the insertion order and is not synchronized.in LinkedList,the manipulation is fast because no shifting is required.

Program:(TestJavaCollection2.java):

```
import java.util.*;  
  
class TestJavaCollection2{  
  
    public static void main(String args[]){  
  
        LinkedList<String> al=new LinkedList<String>();  
  
        al.add("Honey");  
  
        al.add("queen");  
  
        al.add("drone");  
  
    }  
}
```



```
al.add("Hive");

Iterator<String> itr=al.iterator();

while(itr.hasNext()){

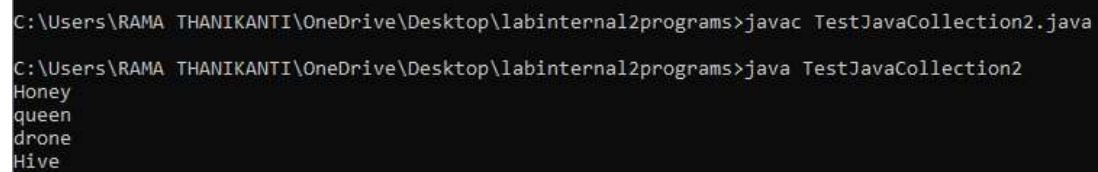
System.out.println(itr.next());

}

}

}
```

Output:



```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac TestJavaCollection2.java
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java TestJavaCollection2
Honey
queen
drone
Hive
```

Vector:

Vector uses a dynamic array to Store the data elements .it is similar to ArrayList However,it is synchronized and contains many metho that are not the part of collection framework

Program:(TestJavaCollection3.java):

```
import java.util.*;
```

```
class TestJavaCollection3{

public static void main(String args[]){

Vector<String> v=new Vector<String>();

v.add("Honey");

v.add("queen");

v.add("drone");

v.add("Hive");

Iterator<String> itr=v.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac TestJavaCollection3.java
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java TestJavaCollection3
Honey
queen
drone
Hive
```

Stack:

The Stack is the subclass of vector. it implements the last in first out data structure .i.e., Stack. The stack contains all of the methods of vector class and also provides its methods like boolean push(), boolean pop(),

Program:(TestJavaCollecton4.java):

```
import java.util.*;

public class TestJavaCollection4{

    public static void main(String args[]){

        Stack<String> stack=new Stack<String>();

        stack.push("arcade");

        stack.push("stereo hearts");

        stack.push("avicii");

        stack.push("Lovely");

        stack.push("Heat waves");

        stack.pop();

        Iterator<String> itr=stack.iterator();

        while(itr.hasNext()){

            System.out.println(itr.next());

        }

    }

}
```

```
}
```

```
}
```

Output:

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>javac TestJavaCollection4.java
```

```
C:\Users\RAMA THANIKANTI\OneDrive\Desktop\labinternal2programs>java TestJavaCollection4  
arcade  
stereo hearts  
avicii  
Lovely
```