Chapter Four

Exception Handling

By: Sinodos G.

Introduction

- Exception is a problem that arises during the execution of a program.
- When an exception occurs, the normal execution flow of the program will be interrupted.
- Java provides programmers with the capability to handle runtime exceptions.
- Using the capability of exception handling, you can develop robust programs for mission-critical computing.

Cont'd …

- A program that does not provide code for catching and handling exceptions will terminate abnormally, and may cause serious problems.
  - Example:- if your program attempts to transfer money from a savings account to a checking account, but because of a runtime error is terminated after the money is drawn from the savings account and before the money is deposited in the checking account, the customer will lose money.
- Exceptions occur for various reasons. the:-
  - User may enter an invalid input,
  - Program may attempt to open a file that doesn't exist
  - Network connection may hang up, or
  - Program may attempt to access an out-of-bounds array element.

Example:-

```
import java.util.Scanner;

public class Program1 {
    public static void main(String aa[]){

        Scanner sc=new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int number=sc.nextInt();
        System.out.println("the sum of number: "+number);

    }
}
```

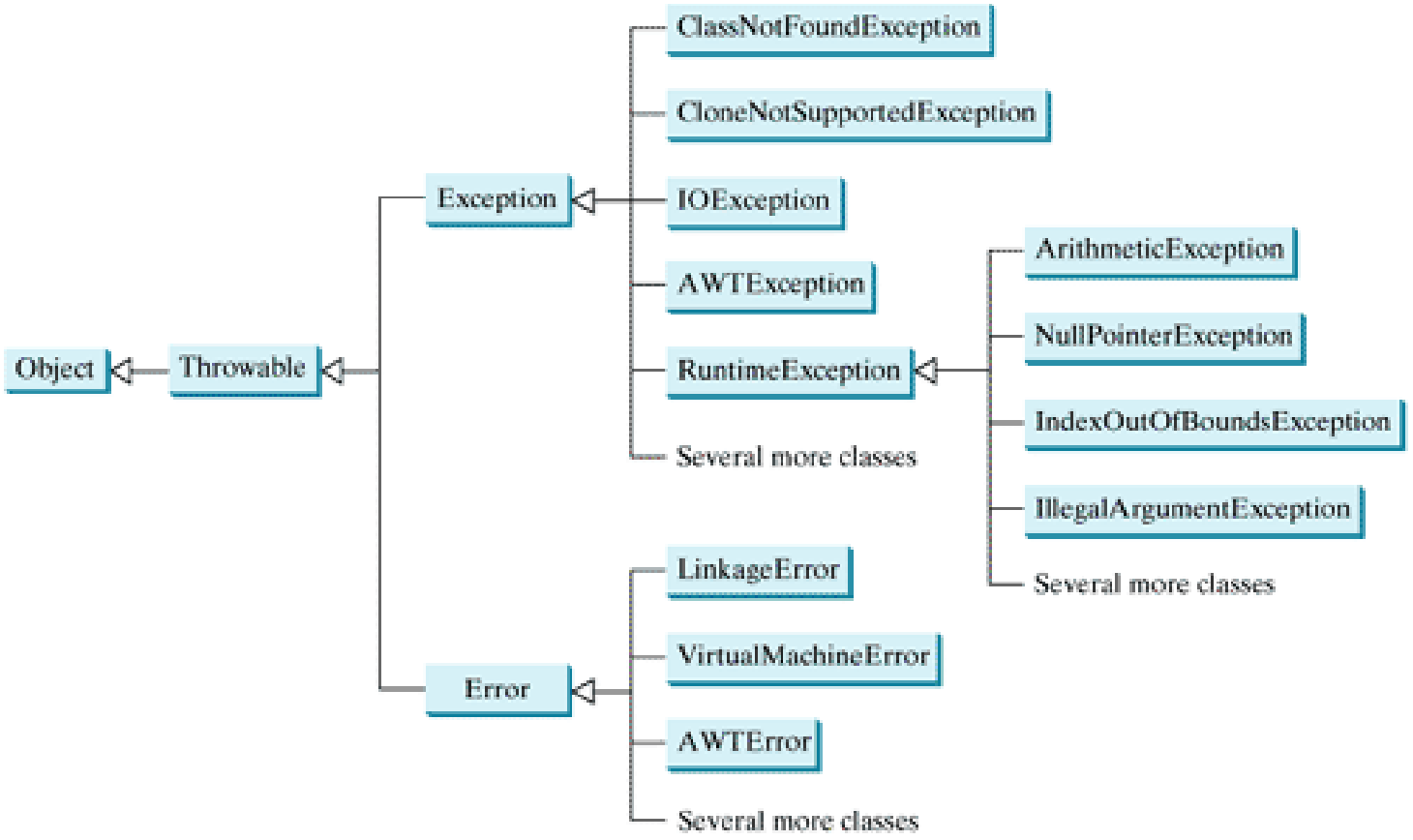
Figure 1: An exception occurs when you enter an invalid input

Exception Types

- **A Java exception is an instance of a class derived from Throwable.**
- **The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.**
- **Errors related to GUI components are included in the java.awt package;**
- **numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.**
- **You can create your own exception classes by extending Throwable or a subclass of Throwable.**

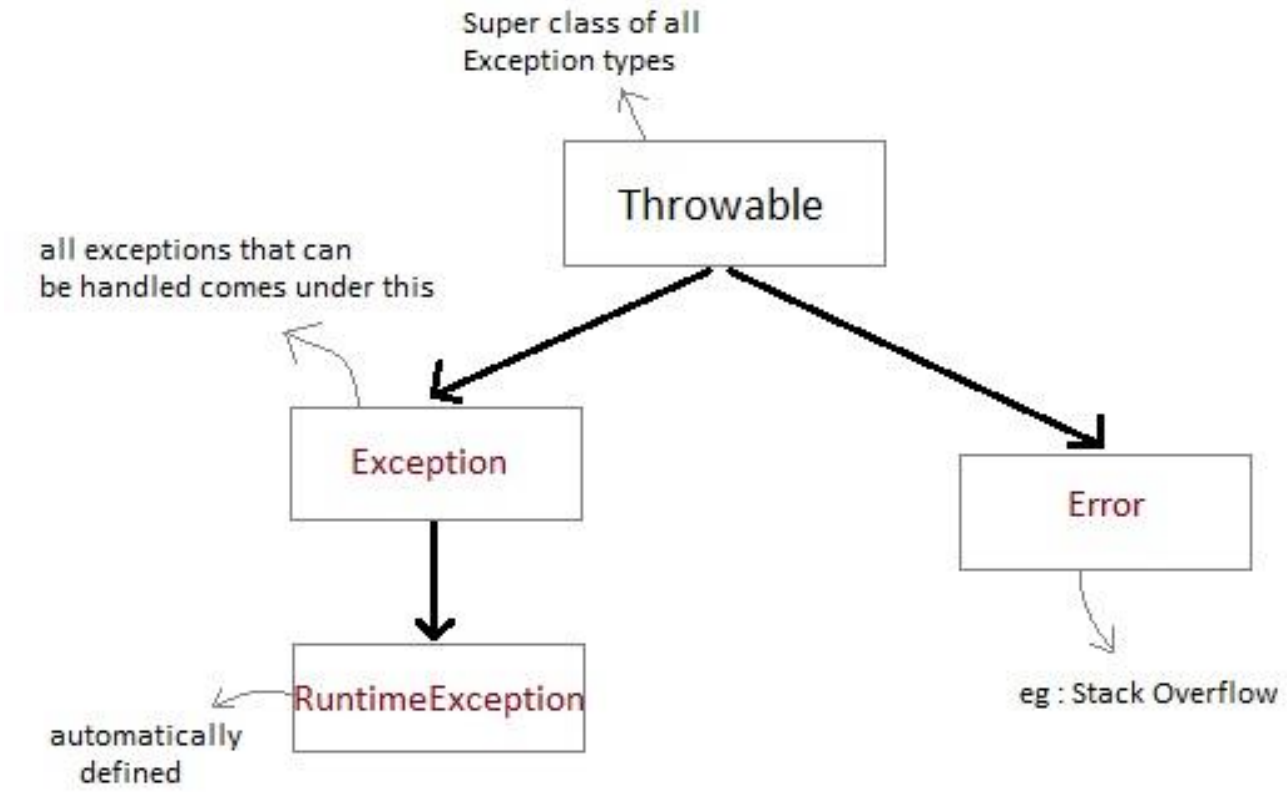
Cont'd ...

- Exceptions thrown are instances of the classes shown in this diagram, or of subclasses of one of these classes.



Cont'd ...

- All exception types are subclasses of class Throwable, which is at the top of exception class hierarchy.



System errors

- System errors are thrown by the JVM and represented in the Error class.
- The Error class describes internal system errors. Such errors rarely occur.
- If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.
- Examples: of subclasses of Error
  - LinkageError
  - VirtualMachineError
  - AWTError

Exceptions

- Exceptions are represented in the Exception class, which describes errors caused by your program and by external circumstances.
- These errors can be caught and handled by your program.
- Examples: of subclasses of Exception
  - ClassNotFoundException
  - IOException
  - CloneNotSupported Exception
  - AWTException

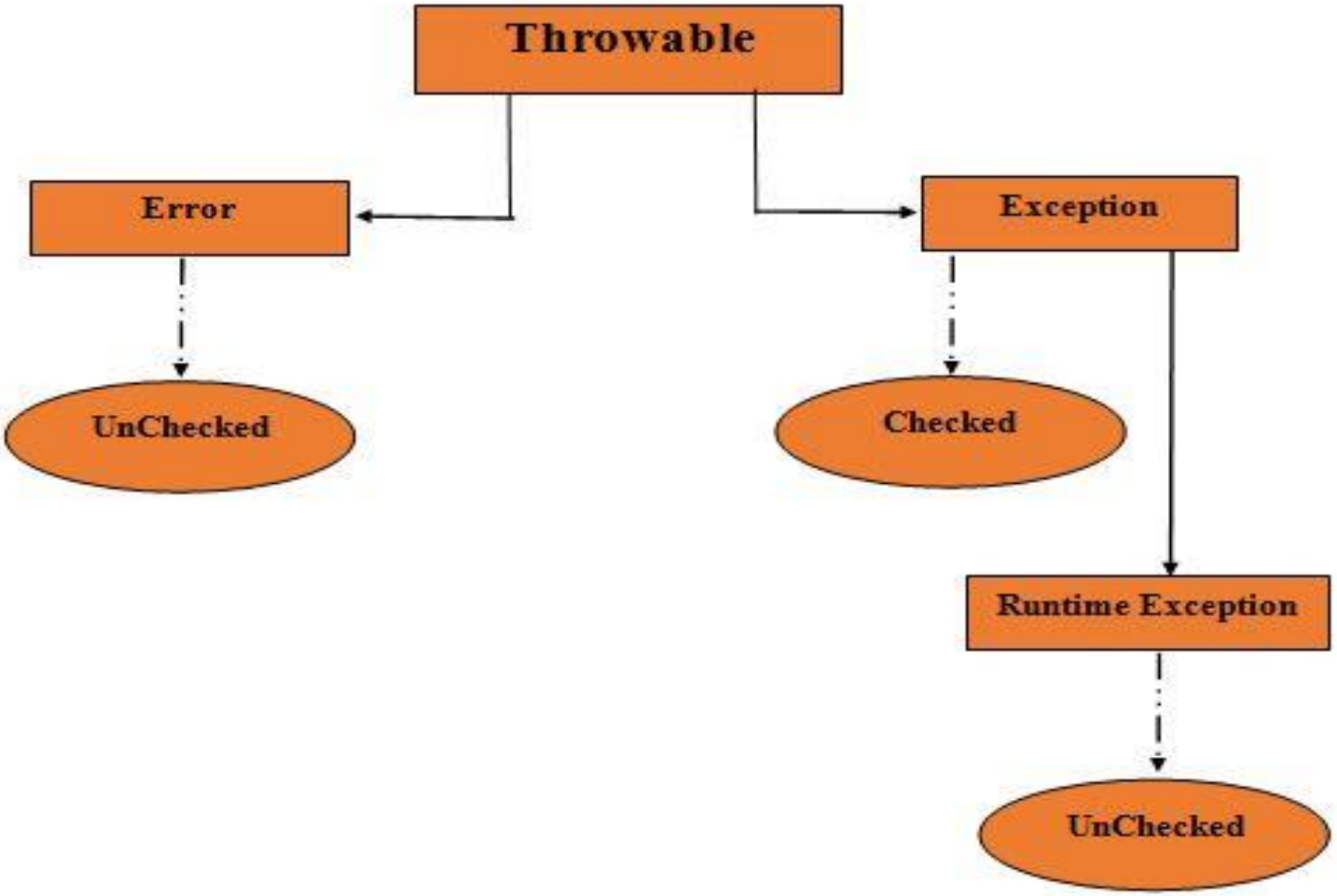
Runtime exceptions

- Runtime exceptions are represented in the Runtime Exception class, which describes programming errors.
  - such as bad casting, accessing an out-of-bounds array, and numeric errors.
- Runtime exceptions are generally thrown by the JVM. Examples of subclasses are:-
  - **ArithmeticException**
  - **NullPointerException**
  - **IndexOutOfBoundsException**
  - **IllegalArgumentException**

Cont'd ...

- Based on these, we have three categories of Exceptions:-
  - Checked exception
  - Unchceked Exception
  - Errors
- RuntimeException, Error, and their subclasses are known as unchecked exceptions.
- All other exceptions are known as checked exceptions:-
  - meaning that the compiler forces the programmer to check and deal with them.

Cont'd ...


Throwable
Error
Exception
UnChecked
Checked
Runtime Exception
UnChecked

Cont'd …

- Checked exceptions:
  - an exception that occurs at the compile time, these are also called as compile time exceptions.
  - Can't simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

Example:

```
package Exception;
import java.io.*;

public class Exception {
public static void main(String args[]){
File file = new File("E:\\file.txt");
FileReader fr=new FileReader(file);
   }
}
```

```
Exception in thread "main" java.io.FileNotFoundException: E:\file.txt
```

Cont'd ...

- Unchecked exceptions:-
  - An exception that occurs at the time of execution.
  - also called as Runtime Exceptions. These include programming bugs, such as:-
    - logic errors or improper use of an API.
  - Runtime exceptions are ignored at the time of compilation.

Example:

- if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then
  - an ArrayIndexOutOfBoundsExceptionexception occurs.

Cont'd ...

Example:-

```
public class UncheckedDemo {
   public static void main(String args[])
{
      int num[] = {1, 2, 3, 4};
      System.out.println(num[5]);
   }
}
```

Output:

- Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
- At Exceptions.

  Unchecked_Demo.main(Unchecked_Demo.java:8)

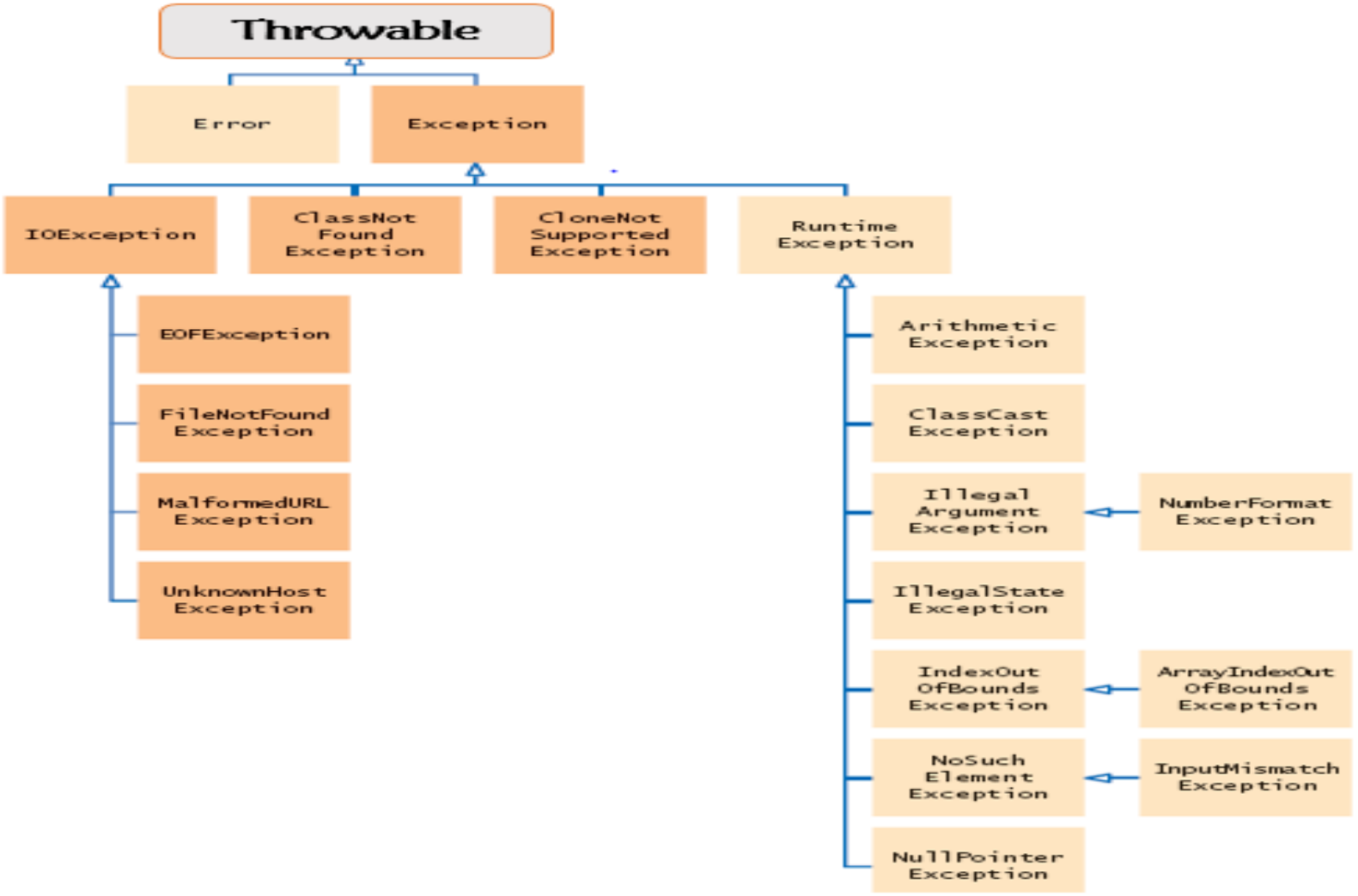
Cont'd …

- **Errors :-**
  - These are not exceptions at all, but problems that arise beyond the control of the user or the programmer.
  - Errors are typically ignored in your code because you can rarely do anything about an error.

  Example:-

  - if a stack overflow occurs, an error will arise.
  - They are also ignored at the time of compilation.

Exception Hierarchy



Cont'd …

Example:-

```
package Exception;

public class Jeganuna1 {
    public static void main (String k[ ])
    {
          int x=3,y=0;
           int a= x/y;
          //denominator become zero
          System.out.println("a  ="+a);
    }
}
```

Handout

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exception.Jeganuna1.main(Jeganuna1.java:8)
```

Exception Handling

- Java's exception-handling model is based on three operations:-
  - declaring an exception,
  - *throwing an exception*, and
  - catching an exception

Cont'd ...

- **In Java, exception handling is done using five keywords,**
  - **try**
  - **catch**
  - **throw**
  - **throws**
  - **finally**
- **Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs.**
-

Declaring Exceptions

- In java when every method must state the types of checked exceptions it might throw. This is known as declaring exceptions.
- System errors and runtime errors can happen to any code, Java does not require that you declare Error and RuntimeException (unchecked exceptions) explicitly in the method.
- However, all other exceptions thrown by the method must be explicitly declared in the method declaration so that the caller of the method is informed of the exception.

```
public class Program2 {
  public static void main(String args[]){
    int a,b,c;
    try{
      a = 0;
      b = 10;
      c = b/a;
      System.out.println("This line will not be executed");
    }
    catch(ArithmeticException e)
    {
      System.out.println("Divided by zero");
    }
  }
}
```

Cont'd …

- To declare an exception in a method, use the throws keyword in the method declaration:-
- Example:-

```
public void myMethod() throws IOException
```

- The throws keyword indicates that myMethod might throw an IOException.
- If the method might throw multiple exceptions, add a list of the exceptions, separated by commas, after throws:

Syntax:

```
public void myMethod()
throws Exception1, Exception2, …, ExceptionN
```

Throwing Exceptions

- A program that detects an error can create an instance of an appropriate exception type and throw it. This is known as throwing an exception.
- If a method is capable of causing an exception that it does not handle, it must specify this behavior so that caller of the method can guard themselves against that exception.
- A throws clause lists the types of exceptions that a method might throw except:-
  - Error or RuntimeExcception or any of their subclasses.
- All other exceptions that a method can throw must be declared in the throws clause. If they are not, a compile time error will result.

Cont'd ...

- We do this by including a throws clause in the method's declaration.
- The keyword to declare an exception is throws, and the keyword to throw an exception is throw.

Example:

- Suppose the program detected that an argument passed to the method violates the method contract
  - e.g:- the argument must be non-negative, but a negative argument is passed;
- The program can create an instance of IllegalArgumentException and throw it, as follows:

Cont'd …

Syntax:

- IllegalArgumentException ex = new IllegalArgumentException("Wrong Argument"); throw ex;

Or

- throw new IllegalArgumentException("Wrong Argument");

- In general, each exception class in the Java API has at least two constructors:-
- a no-arg constructor, and a constructor with a String argument that describes the exception.
- This argument is called the exception message, which can be obtained using getMessage().

Example

```
import java.io.*;
public class ArrayException1
{
public static void main(String args[])
{
    public void deposit(double amount)
    throws RemoteException {
        // Method implementation
    throw new RemoteException();
    }    // Remainder of class definition
}
```

Catching Exceptions

- to declare an exception and how to throw an exception.

- When an exception is thrown, it can be caught and handled in a try-catch block, as follows:

Syntax:

```
try {
    statements;
    throw exceptions
    }

  catch(Exception1 exVar1){
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}

...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

Example:

An array declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
public class ArrayException1 {
    public static void main(String args[]) {
        try{
            int a[] = new int[2];
            System.out.println("a[3]:= " + a[3]);
            }
        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("Error ... " + e);
            }
        System.out.println("Error Out of the block");
    }
}
```

Cont'd ...

- If no exceptions arise during the execution of the try block, the catch blocks are skipped.
- Various exception classes can be derived from a common superclass.
- If a catch block catches exception objects of a superclass, it can catch all the exception objects of the subclasses of that superclass.

```
try {
  ...
}
catch (Exception ex) {
  ...
}
catch (RuntimeException ex) {
  ...
}
```

(a) Wrong order

```
try {
  ...
}
catch (RuntimeException ex) {
  ...
}
catch (Exception ex) {
  ...
}
```

(b) Correct order

Multiple catch Clauses

- In some cases, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception.
- In case of multiple catch statements exception subclasses must come before any of their superclasses.
- When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed.

Cont'd …

Syntax:

```
try {
   …
}
catch (<exntype1> e1) {
…
}
catch(<exntype2>  e2) {
…
}
finally{
 // finally is optional
…
}
```

- Each try statement must be followed by at least one catch or finally block.

Example:

```
    public static void main(String args[]) {
        int a[] = new int[2];
            try {
                System.out.println("Access element three :" + a[3]);
                }
            catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("Exception thrown  :" + e);
               }
            finally {
                a[0] = 6;
                System.out.println("a[0]:= " + a[0]);
        System.out.println("The finally statement is executed");
        }
    }
}
```

```
run:
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
a[0]:= 6
The finally statement is executed
BUILD SUCCESSFUL (total time: 0 seconds)
```

Getting Information from Exceptions

- An exception object contains valuable information about the exception.
- Some methods to get information regarding the exception is:-
- printStackTrace()
  - method prints stack trace information on the console.
- getStackTrace()
  - method provides programmatic access to the stack trace information printed by printStackTrace().
- getMessage(), and toString() methods,

finally Clause

- Occasionally, you may want some code to be executed regardless of whether an exception occurs or is caught.
- Java has a finally clause that can be used to accomplish this objective.
- Syntax:

```
try {
  statements;
}
catch (TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}
```

- The catch block may be omitted when the finally clause is used.

Cont'd …

- **finally statement (block)**
  - The finally will execute whether or not an exception is thrown.
    - If an exception is thrown, the finally block will execute even if no catch statement matches the exception.
  - The finally block is used to execute the statements that must be executed in each and every condition like closing the opened files and freeing the resources.
  - It may be add immediately after the try block or after the last catch block

Example:

```
    try{
        int a = args.length;
        System.out.println("a:= " + a);
        int b = 42 / a;
        int c[] = { 1 };
        c[4] = 99;
        }
    catch(ArithmeticException e) {
        System.out.println( "Error Division by Zero ...>: "+e.getMessage());
        }
    catch(ArrayIndexOutOfBoundsException e) {
        System.out.println( e.getMessage());
        }
        finally {
    System.out.println("finally blocks.");
    }
}
```

```
run:
a:= 0
Error Division by Zero ...>: / by zero
finally blocks.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Rethrowing Exceptions

- allows an exception handler to rethrow the exception if the handler cannot process the exception or the handler simply wants to let its caller be notified of the exception.
  - Syntax:

```
try {
  statements;
}
catch (TheException ex) {
  perform operations before exits;
  throw ex;
}
```

- The statement throw ex rethrows the exception so that other handlers get a chance to process the exception ex.

Cont'd ...

Example: If exception object not handled properly by us, then the default handler handles it.

- The error handling code perform the following tasks.
  - Find the problem (*Hit* the exception).
  - Inform that an error has occurred (*Throw* the exception) .
  - Received the error information (*Catch* the exception).
  - Take corrective actions (*Handle* the exception).

Cont'd …

- exception mechanism is built around the throw-and-catch paradigm.
  - throw an exception is to signal that an unexpected error condition has occurred.
  - catch an exception is to take appropriate action to deal with the exception.
  - an exception is caught by an exception handler, and the exception need not be caught in the same context that it was thrown in.
  - the runtime behavior of the program determines which exceptions are thrown and how they are caught. The throw-and-catch principle is embedded in the try-catch-finally construct.

throw, throws and finally Keyword

throw keyword

- **used to throw an exception explicitly.**
- **Only object of Throwable class or its sub classes can be thrown.**
- **Program execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.**
- **Syntax :**

```
throw ThrowableInstance
```

- **Creating Instance of Throwable class**
- **There are two possible ways to create an instance of class Throwable,**

- **Using a parameter in catch block.**
- **Creating instance with new operator.**
- **new NullPointerException("test");**
- **This constructs an instance of NullPointerException with name test.**

```
class Program3
{
  static void avg()
  {
    try
    {
      throw new ArithmeticException("demo");
    }
    catch(ArithmeticException e)
    {
      System.out.println("Exception caught");
    }
  }

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

Cont'd …

throws Keyword

- **Any method that is capable of causing exceptions must list all the exceptions possible during its execution, so that anyone calling that method gets a prior knowledge about which exceptions are to be handled. A method can do so by using the throws keyword.**

- **Syntax:**

- **type method_name(parameter_list) throws exception_list**
- **{**
- **// definition of method**
- **}**

Cont'd ...

```
class Program4 {
  static void check() throws ArithmeticException
  {
    System.out.println("Inside check function");
    throw new ArithmeticException("demo");
  }

  public static void main(String args[])
  {
    try
    {
      check();
    }
    catch(ArithmeticException e)
    {
      System.out.println("caught" + e);
    }
  }
}
```

Difference between throw and throws

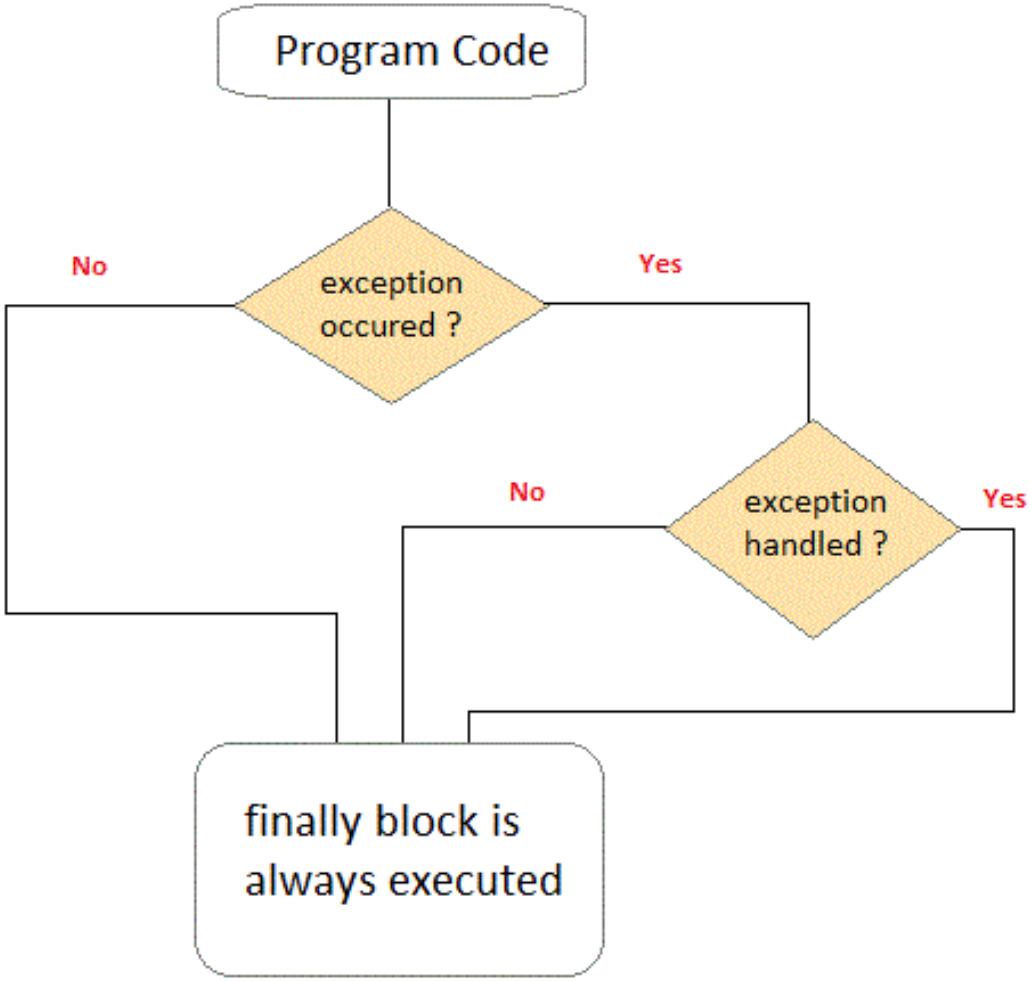
throw

- **throw keyword is used to throw an exception explicitly.**
- **throw keyword is followed by an instance of Throwable class or one of its sub-classes.**
- **throw keyword is declared inside a method body.**
- **We cannot throw multiple exceptions using throw keyword.**

throws

- **throws keyword is used to declare an exception possible during its execution.**
- **throws keyword is followed by one or more Exception class names separated by commas.**
- **throws keyword is used with method signature (method declaration).**
- **We can declare multiple exceptions (separated by commas) using throws keyword.**

- inally clause
- A finally keyword is used to create a block of code that follows a try block. A finally block of code is always executed whether an exception has occurred or not. Using a finally block, it lets you run any cleanup type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of catch block.


Program Code
exception occured ?
No
Yes
exception handled ?
No
Yes
finally block is always executed

```
public class FinalyKey {
  public static void main(String[] args)
  {
    int a[] = new int[2];
    System.out.println("out of try");
    try
    {
      System.out.println("Access invalid element"+ a[3]);
      /* the above statement will throw ArrayIndexOutOfBoundException */
    }
    finally
    {
      System.out.println("finally is always executed.");
    }
  }
}
```

Cont'd ...

- **User defined Exception**

- **You can also create your own exception sub class simply by extending java Exception class.**

- **You can define a constructor for your Exception sub class (not compulsory) and you can override the toString() function to display your customized message on catch.**

Cont'd …

```
public class udefinedEx extends Exception{
  private int ex;
  udefinedEx(int a){
    ex = a;
  }
  public String toString(){
    return "MyException[" + ex +"] is less than zero";
  }
}

class Test{
  static void sum(int a,int b) throws udefinedEx
  {
    if(a<0){
      throw new udefinedEx(a); //calling constructor of user-defined exception class
    }
    else
    {
      System.out.println(a+b);
    }
  }
```

Cont'd ...

```
public static void main(String[] args){
  try
  {
    sum(-10, 10);
  }
  catch(udefinedEx me){
    System.out.println(me); //it calls the toSt
  }
 }
}
```

Question