



Java Programming

TEK RAJ CHHETRI

Course Objective

- Explain the Java programming environment
- Describe the concepts of programming elements using Java and object-oriented
- programming concepts
- Apply the exception handling and input/output in Java programming
- Apply the event handling, GUI programming using swing, and Java database connectivity

Unit 5: Exception Handling and Multithreading

- Dealing With Errors
- Catching Exceptions
- try, catch, throw, throws, and finally
- Introduction to Multithreading

Learning Outcome (Unit 5)

- Design and develop java error handling software.
- Understand multithreading and be able to apply it practically.



Dealing With Errors

Errors

- **Syntax errors**

- arise because the rules of the language have not been followed.
- detected by the compiler.

- **Logic errors**

- leads to wrong results and detected during testing.
- arise because the logic coded by the programmer was not correct.

- **Runtime errors**

- Occur when the program is running and the environment detects an operation that is impossible to carry out.

- **Code errors**

- Divide by zero
- Array out of bounds
- Integer overflow
- Accessing a null pointer (reference)

- **Exception**

- Is an abnormal condition that arises in a code sequence at run time.
- Also called runtime error.

- **Exception Handling**

- A mechanism to handle exception by detecting and responding to exception in systematic way in order to maintain the normal flow of application.
- Any exception not handled are caught by JRE (java runtime environment).

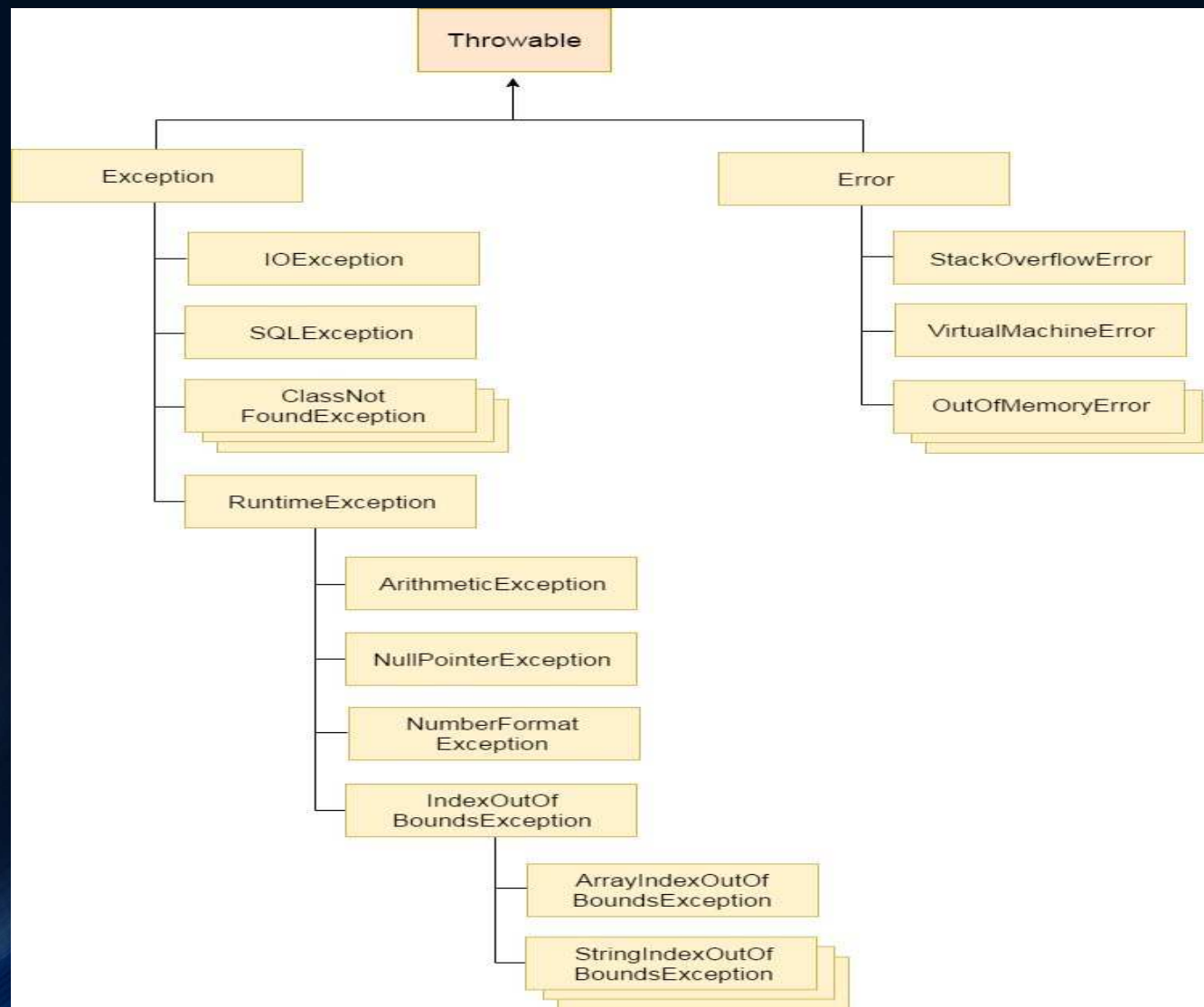


Fig: Exception Class Hierarchy (Image: javatpoint.com)

- **Types of Exception:**

- **Checked Exception**

- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.
 - . Checked exceptions are checked at compile-time.
 - E.g.
 - IOException, SQLException etc

- **Unchecked Exception**

- The classes that extend RuntimeException are known as unchecked exceptions
 - Unchecked exceptions are not checked at compile-time rather they are checked at runtime.
 - E.g.
 - ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

- **Error**

- Error is irrecoverable
 - E.g.
 - OutOfMemoryError, VirtualMachineError, AssertionError etc.

```
1. class Error{
2.     public static void main(String[] args) {
3.         int number = 5;
4.         float result = number / 0;
5.         System.out.println("Result = "+result);
6.     }
7. }
```

C:\WINDOWS\system32\cmd.exe

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>javac Error.java

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java Error
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Error.main(Error.java:4)

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>

```
1. class ArrayError{
2.     public static void main(String[] args) {
3.         int number[] = {1,2,3,4,5};
4.         int result = number[5];
5.         System.out.println("Result = "+result);
6.     }
7. }
```

C:\WINDOWS\system32\cmd.exe

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>javac ArrayError.java

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java ArrayError
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at ArrayError.main(ArrayError.java:4)

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>

Exception Handling

- Java provides 5 keywords that help handle exception.

try

catch

finally

throw

throws

- Handling exception

```
void compute(){  
    try{  
        getIO();  
    }catch(IOException e){  
    }  
}
```

```
void compute() throws IOException{  
    getIO();  
}
```

- Handling exception:

```
1. class ThrowsException{
2.     static void throwOne() throws IllegalAccessException{
3.         System.out.println("Inside throwOne");
4.         throw new IllegalAccessException("Sorry Illegal Access");
5.     }
6.     public static void main(String args[]){
7.         try{
8.             throwOne();
9.         }catch(IllegalAccessException ie){
10.            System.out.println("Caught:="+ie);
11.        }
12.    }
13. }
```

C:\WINDOWS\system32\cmd.exe

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>javac ThrowsException.java

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java ThrowsException

Inside throwOne

Caught:=java.lang.IllegalAccessException: Sorry Illegal Access

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>

- Catching Exceptions:

```
try {  
    //Statements that may throw exceptions  
}  
  
catch (Exception1 e1) {  
    //code to handle exceptions of type Exception1;  
}  
  
catch (Exception2 e2) {  
    // code to handle exceptions of type Exception2;  
}  
  
...  
catch (ExceptionN eN) {  
    // code to handle exceptions of type exceptionN;  
}  
  
// statement after try-catch block
```

- Catching Exceptions:

```
try {  
    int numbers[10] = 56 / userInput;  
}catch (ArithmeticException e1) {  
    System.out.println("ArithmeticException occur");  
}catch (ArrayIndexOutOfBoundsException e2) {  
    System.out.println("ArrayIndexOutOfBoundsException occur");  
}catch (ExceptionN e) {  
    System.out.println("Generic Exception occur");  
}  
|
```

- Getting Information from Exceptions:
- Use the instance of `java.lang.Throwable` class
 - `String toString()`
 - Returns a short description of the exception
 - `String getMessage()`
 - Returns the detail description of the exception
 - `void printStackTrace()`
 - Prints the stacktrace information on the console

- Getting Information from Exceptions:

```
1. class ErrorHandle{
2.     public static void main(String[] args) {
3.         try{
4.             int number = 5;
5.             float result = number / 0;
6.             System.out.println("Result = "+result);
7.         }catch(ArithmeticException ae){
8.             System.out.println(ae.toString());
9.             System.out.println(ae.getMessage());
10.            ae.printStackTrace();
11.        }
12.    }
13. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>javac ErrorHandler.java
C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java ErrorHandler
java.lang.ArithmeticException: / by zero
/ by zero
java.lang.ArithmeticException: / by zero
    at ErrorHandler.main(ErrorHandler.java:5)

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>
```

Issues

```
1. java.io.PrintWriter output = null;
2.     try {
3.         output = new java.io.PrintWriter("text.txt");
4.         output.println("Welcome to Java");
5.         output.close();
6.     }
7.     catch(java.io.IOException ex){
8.         ex.printStackTrace() ;
9.     }
```

- Output must be closed despite exception.
- Same kind of situation: eg database connection

Solution

- Use *finally* clause for code that must be executed "no matter what"

```
try{  
  
}catch(Exception1 e1){  
  
}catch(ExceptionN en){  
  
}  
finally{//optional  
  
}
```

```
1. java.io.PrintWriter output = null;
2.  try {
3.      output = new java.io. PrintWriter("text.txt");
4.      output.println("Welcome to Java");
5.  }
6.  catch(java.io.IOException ex){
7.      ex.printStackTrace() ;
8.  }
9.  finally {
10.      if (output != null) output.close();
11.  }
```


- **Finally Block:**
- Executed when try block is exited in any of three ways:
 - *After last statement of try block (success).*
 - *After last statement of catch clause, if this catch block caught an exception.*
 - *When an exception was thrown in try block and not caught*
- *Executed even if there is a return statement prior to reaching the finally block*

- Throwing Exceptions:
- When somebody writes a code that could encounter a **runtime error**,
 - it creates an object of appropriate Exception class and throws it
 - and must also declare it in case of checked exception

```
1 class ThrowExample {
2     static void demoproc(){
3         try{
4             throw new NullPointerException("Throw Demo");
5         } catch (NullPointerException npe){
6             System.out.println("Caught inside catch");
7             throw npe; //rethrowing exception
8         }
9     }
10    public static void main(String[] args) {
11        try{
12            demoproc();
13        } catch (NullPointerException ex){
14            System.out.println("Recought: "+ex);
15        }
16    }
17 }
```

C:\WINDOWS\system32\cmd.exe

```
C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java ThrowExample
Caught inside catch
Recought: java.lang.NullPointerException: Throw Demo
C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>
```

- Creating Custom Exceptions:

- Create custom exception only if predefined exceptions are not sufficient.
- Create a class that *extends Exception*
- A good practice is to:
 - Add one argument less constructor
 - And another one string type parameter

```
1. class BTypeError extends Exception{
2.     BTypeError(){
3.         //argument less constructor
4.     }
5.     BTypeError(String msg){
6.         //constructor with one string type parameter
7.     }
8. }
```

```

class CustomException extends Exception{
    private int detail;
    CustomException(int name){
        detail = name;
    }
    public String toString(){
        return "CustomException["+detail+"]";
    }
}

class CreateException{
    static void compute(int number) throws CustomException{
        System.out.println("Called compute (" + number + ")");
        if(number < 5){
            throw new CustomException(number);
        }
        System.out.println("Normal Exit");
    }
    public static void main(String args[]){
        try{
            compute(10);
            compute(20);
            compute(3);
        }catch(CustomException ce){
            System.out.println("Caught := "+ce);
        }
    }
}

```

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>javac CreateException.java

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java CreateException

Called compute (10)

Normal Exit

Called compute (20)

Normal Exit

Called compute (3)

Caught :=CustomException[3]

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>

- When to create Custom Exceptions?:
- Use the exception classes in the API whenever possible.
- You should write your own exception classes if you answer 'yes' to one of the following:
 - ✓ Do you need an exception type that isn't represented by those in the Java platform?
 - ✓ Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
 - ✓ Do you want to pass more than just a string to the exception handler?

- When to use Exceptions:
- Use it if the event is truly exceptional and is an error
- Do not use it to deal with simple, expected situations.

```
try {  
    System.out.println(message.toString());  
}  
catch (NullPointerException ex) {  
    System.out.println("message is null");  
}
```


Above code can be replaced by code below.

```
if(message != null){  
    System.out.println(message.toString());  
}else{  
    System.out.println("message is null");  
}
```




Introduction to Multithreading

mul·ti·thread·ing

/'mɒlti 'THredɪŋ, 'mɒlti-/ 

noun **COMPUTING**

noun: **multithreading**; noun: **multi-threading**

a technique by which a single set of code can be used by several processors at different stages of execution.

- Multithreading is achieved using thread, a light weight process.
- The can be easily spawned
- The Java Virtual Machine spawns a thread when your program is run called the Main Thread

- **Why threads?**

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU

- **Why threads?**

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU

- **Eg:**

- Web server with multithreaded environment can serve multiple request simultaneously increasing response time.

- **Creating Thread:**
- Two ways to create thread
 1. Implementing **Runnable** interface
 2. Extending **Thread** class

- **Creating Thread using Runnable Interface:**
- First you need to implement the runnable interface
 - `class RunnableThreadDemo implements Runnable{}`
- Then use the methods provided by runnable interface
 - `Thread(Runnable ThreadObj, String ThreadName)` = one of the Thread constructor, used to instantiate thread class.
 - `public void run()` = serves as a entry point and define code that constitutes new thread
 - `void start()` = starts the thread, thread doesn't start unless called this method

```

class RunnableThreadDemo implements Runnable{
    Thread t;
    RunnableThreadDemo(){
        //create new second thread
        t = new Thread(this,"Demo Thread");
        System.out.println("Child thread:= "+t);
        t.start();//start thread
    }
    //entry point for 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 RunnableThread{
    public static void main(String[] args) {
        new RunnableThreadDemo(); // creates 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("Exiting main thread");
    }
}

```

```

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java RunnableThread
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
Child Thread := 0
Main Thread := 2
Exiting child thread
Main Thread := 1
Main Thread := 0

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>

```


- **Creating Thread by extending Thread class:**
- First you need to extend the Thread class
 - `class RunnableThreadDemo extends Thread{}`
- Then override the **run()** method and also call **start()** to begin thread.

```

class ExtendingThreadDemo extends Thread{

    ExtendingThreadDemo(){
        super("Demo Thread");
        System.out.println("Child thread := "+ this);
        start();
    }
    //entry point for 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 ExtendingThread{
    public static void main(String[] args) {
        new ExtendingThreadDemo(); // creates 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("Exiting main thread");
    }
}

```

```

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>java ExtendingThread
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
Child Thread := 0
Main Thread := 2
Exiting child thread
Main Thread := 1
Main Thread := 0
Exiting main thread

C:\Users\USER\Desktop\lecture\java\Unit -V\Programs>

```

Suggested Readings

- The respective topics in The complete Reference Java 7 (or any higher edition) by Hebert Schildt
- <https://docs.oracle.com/javase/tutorial/essential/exceptions>
- <https://docs.oracle.com/javase/tutorial/essential/concurrency/>



References

- The complete Reference Java 7 by Hebert Schildt
- <https://www.javatpoint.com/> and [tutorialspoint.com](https://www.tutorialspoint.com)
- Java 8 in Action by Dreamtech press.
- Mit Opencourseware
- <http://ee402.eeng.dcu.ie/>
- <https://docs.oracle.com/javase/tutorial/essential/exceptions>
- <https://docs.oracle.com/javase/tutorial/essential/concurrency/>
- <https://www.geeksforgeeks.org>
- CIS3023: Programming Fundamentals for CIS Majors II Course Lecture Slide by Ganesh Viswanathan
- <https://images.google.com> for Images