Exception Handling

Tasnim Sharmin Alin
Lecturer
Department of Computer Science and Engineeing
Leading University,Sylhet

Exception Handling

- **Definition:** An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

- **An exception can occur for many different reasons, including the following:**

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

- **Exception Handling:**

  Exception handling enables programmers to create applications that can resolv s exceptions.

Form of Exception-handling block

```
try {
   // block of code to monitor for errors
}

catch(ExceptionType1 exOb){
    //exception handler for ExceptionType1
}

catch(ExceptionType2 exOb){
    //exception handler for ExceptionType2
}
……….
Finally{
   block of code to be executed after try block ends
}
```

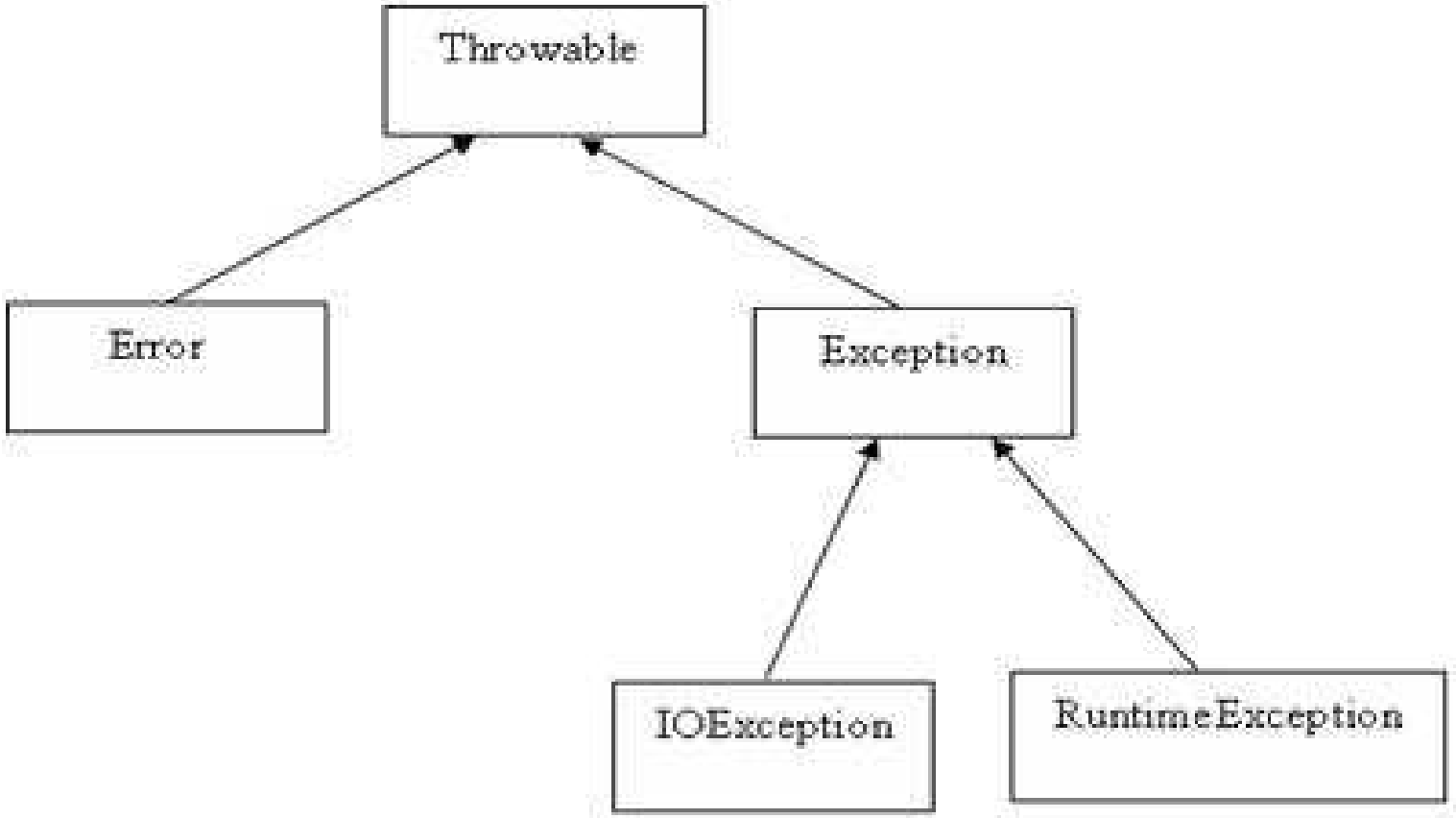
Uncaught Exception

```
public class Exc1 {
    static void subroutine(){
        int d=0;
        int a =10/d;
    }
    public static void main(String[] args) {
       Exc1.subroutine();
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by
    zero
    at exc1.Exc1.subroutine(Exc1.java:4)
    at exc1.Exc1.main(Exc1.java:7)
```

Exception Hierarchy



Types of Exception Handling

There are three types of Exception:

1. **Checked exceptions**
2. **Runtime exceptions**
3. **Errors**

Checked exceptions:

- A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer.
- For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.

Runtime exceptions

- A runtime exception occurs simply because the programmer has made a mistake.
- As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.

Types of Exception Handling

- You've written the code, it all looks good to the compiler and when you go to run the code it falls over because it tried to access an element of an array that does not exist or a logic error caused a method to be called with a null value.

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

- **Errors and Runtime Exceptions fall into the category of unchecked exceptions**.

When to use Exception Handling

- Exception handling is designed to process synchronous errors, which occurs when a statement executes.

- Examples of synchronous exception: out-of range array indices, arithmetic overflow, division by zero, invalid method parameter, thread interruption and unsuccessful memory allocation(due to lack of memory)

- Exception handling is not designed to process problems associated with asynchronous events such as disk I/O completions,network message arrivals,mouse clicks and keystrocks, which occur in parallel with and independent of the programs flow-of control.

Exception Handling Keywords

- Java provides five specific keywords for exception handling purposes:

1. try
2. catch
3. throw
4. throws
5. finally

Using ***try*** and ***catch***

- Try block contains the code which is under observation for exceptions.
- If an exception occurs within the **try** block, it is thrown.
- The catch block contains the remedy for the exception. If any exception occurs in the try block then the control jumps to catch block.
- If there are multiple matching catch blocks when an exception occurs, only the first is executed.

example

```
public class ExcTryCat {
    public static void main(String[] args) {
        int a ,d;
        try{
            d=0;
            a= 42/d;
            System.out.println("This will not be printed ");
        }
        catch(ArithmeticException ob){
            System.out.println("Division by zero  ");
        }
        System.out.println("After catch statement ");
    }
}
```

Output: Division by zero
After catch statement

example

```
import java.util.*;
public class ExcepTest {
   public static void main(String[] args) {
      try{
       int a[] = new int[2];
       System.out.println("Access element three :" + a[3]);
     }catch(ArrayIndexOutOfBoundsException e){
       System.out.println("Exception thrown  :" + e);
     }
     System.out.println("Out of the block");
   }
}
```

Output:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

example

```
public class TryCat {
   public static void main(String[] args) {
      try{
         for(int i=5;i>=0;i--){
            System.out.println(10/i);
         }
      } catch(Exception ex){
         System.out.println("Exception Message: "+ex.getMessage());
         ex.printStackTrace();
      }
      System.out.println("After for loop...");
   }
}
```

Output: 2

```
2
3
5
10
Exception Message: / by zero
After for loop...
java.lang.ArithmeticException: / by zero
      at trycat.TryCat.main(TryCat.java:5 )
```

Multiple catch block

```
public class ExceArr {
   public static void main(String[] args) {
      try{
         int arr[]={2,1};
         arr[3]=1/5;
      }
      catch(ArithmeticException e){
          System.out.println("Division by zero  ");
      }
      catch(ArrayIndexOutOfBoundsException ed){
         System.out.println("Array Index out of bound  ");
      }
   }
}
```

Output: Array Index out of bound

Multiple try block

```
public class MultTry {
    public static void main(String[] args) {
      try{
         int arr[] = {5,0,1,2};
         try{
            int x = arr[3]/arr[1];
            System.out.println("Result is: "+x);
         }     catch(ArithmeticException e){
           System.out.println("Division by zero  ");
       }
      }   catch(ArrayIndexOutOfBoundsException ed){
          System.out.println("Array Index out of bound  ");    }
      }
   }
```

Output: Division by zero

throw

throw

throw keyword is used to throw an exception explicitly. **throw** keyword is used to throw exception to the runtime to handle it.

Syntax: **throw** ***ThrawableInstance***

Here ***ThrawableInstance*** must be an object of type ***Throwable*** or a subclass ***Throwable* .**

There are two ways to obtain a Throwable object:

1. Using a parameter in a catch clause
2. Creating one with the new operator

new NullPointerException(“Test”);

This construct an instance of NullPointerException with the name Test

Throw example

```
public class ThrowDemo {
    static void demoproc(){
        try{
            throw new NullPointerException ("demo");
        }catch(NullPointerException ob){
            System.out.println("Caught inside demoporoc  ");
        }
    }
   public static void main(String[] args) {
        try{
            demoproc();
        } catch(NullPointerException ob){
            System.out.println("Caught inside demoporoc  ");
   }
  }
}
```

Output: Caught inside demoporoc

Throws keyword

- throws is a declaration to state that this method throw the specified exception(s) to the caller.

- Syntax:

```
type method_name(parameter-list) throws exception-list
    {
     // definiton of the method
    }
```

Throws example

```
import java.io.*;
class ThrowExample {
  void mymethod(int num)throws IOException, ClassNotFoundException{
     if(num==1)
       throw new IOException("Exception Message1");
     else
       throw new ClassNotFoundException("Exception Message2");   }
}
public class ThrowsDemo {
    public static void main(String[] args) {
     try{
     ThrowExample obj=new ThrowExample();
     obj.mymethod(2);
   }catch(Exception ex){
     System.out.println(ex);  }
    }
}           output: java.lang.ClassNotFoundException: Exception Message2
```

Difference between throw and throws keywords

1. **Throws clause** in used to declare an exception and **thow** keyword is used to throw an exception explicitly.

2. If we see syntax wise than **throw** is followed by an instance variable and **throws** is followed by exception class names.

3. The keyword **throw** is used inside method body to invoke an exception and **throws clause** is used in method declaration (signature).

Finally keyword

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

      int[] a={3,4,8,5};
      try{
          System.out.println("Invalid element: "+ a[9]);
      }catch(ArrayIndexOutOfBoundsException ed){
           System.out.println("out of range ");
      }
      finally{
          System.out.println("Finally is always executed ");
      }
   }
}
```

Output : out of range

Finally is always executed

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
UnsupportedOperationException	An unsupported operation was encountered.

Exceptions

Exception	Description
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.