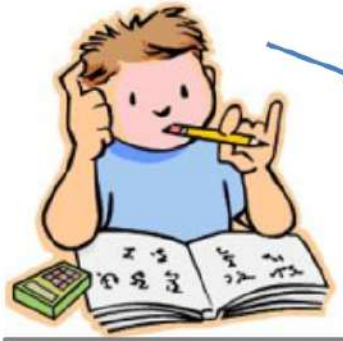
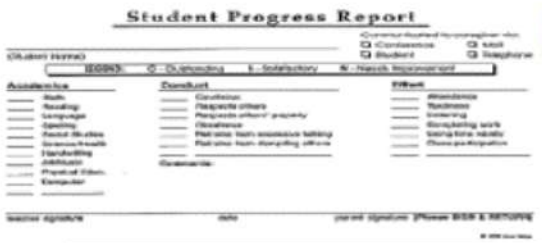

TEKNOTURF
Exception Handling
Part 1

Error Handling

TEKNOTURF

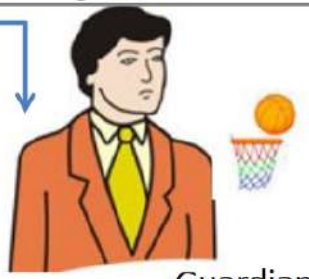


Exception occurs during execution



Exception object containing information about it is created and thrown by the run time system

Runtime checks if principal can handle the exception. If not, the exception is thrown to the guardian.



Guardian

Runtime checks if guardian can handle the exception. If not, the exception is thrown to the parent

Parents

Exception is thrown to the next method in the invocation chain until it is handled

In Java, Handlers handle the exception

In this module you will learn

TEKNOTURF

- Exception
- Exception Types
- Exception Hierarchy
- Try-catch-finally
- Try with Multi catch

Error Handling

TEKNOTURF

Applications will encounter errors while executing

Reliable applications should handle errors as gracefully as possible

Errors

- Should be the “exception” and not the expected behavior
- Must be handled to create reliable applications
- Can occur as the result of application bugs
- Can occur because of factors beyond the control of the application
  - Databases become unreachable
  - Hard drive fails

Exceptions

TEKNOTURF

What is an Exception ?

- Exceptional event - typically an abnormality or error that occurs during runtime
- Cause the normal flow of a program to be disrupted

Examples

- Divide by zero errors
- Accessing the elements of an array beyond its range
- Invalid input
- Opening a non-existent file

Advantages of Exceptions

TEKNOTURF

Separating Error-Handling Code from "Regular" Code

Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program

Propagating Errors Up the Call Stack

If the readFile method is the fourth method in a series of nested method calls made by the main program: method1 calls method2, which calls method3, which finally calls readFile

Grouping and Differentiating Error Types

An example of a group of related exception classes in the Java platform are those defined in java.io — IOException and its descendants

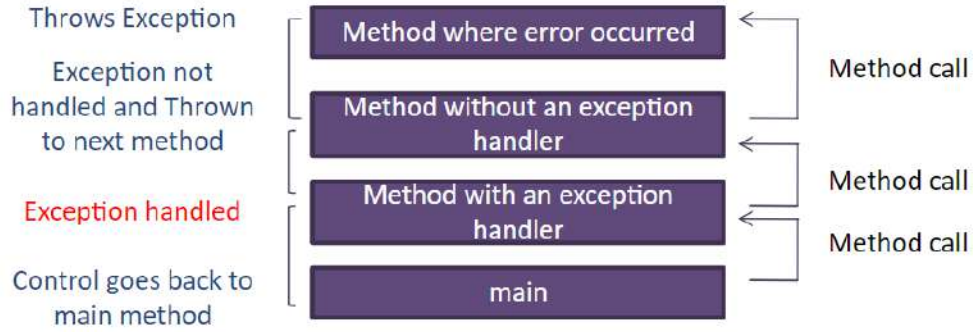
Exception

TEKNOTURF

When an exception occurs within a method, the method typically creates an exception object and hands it off to the runtime system

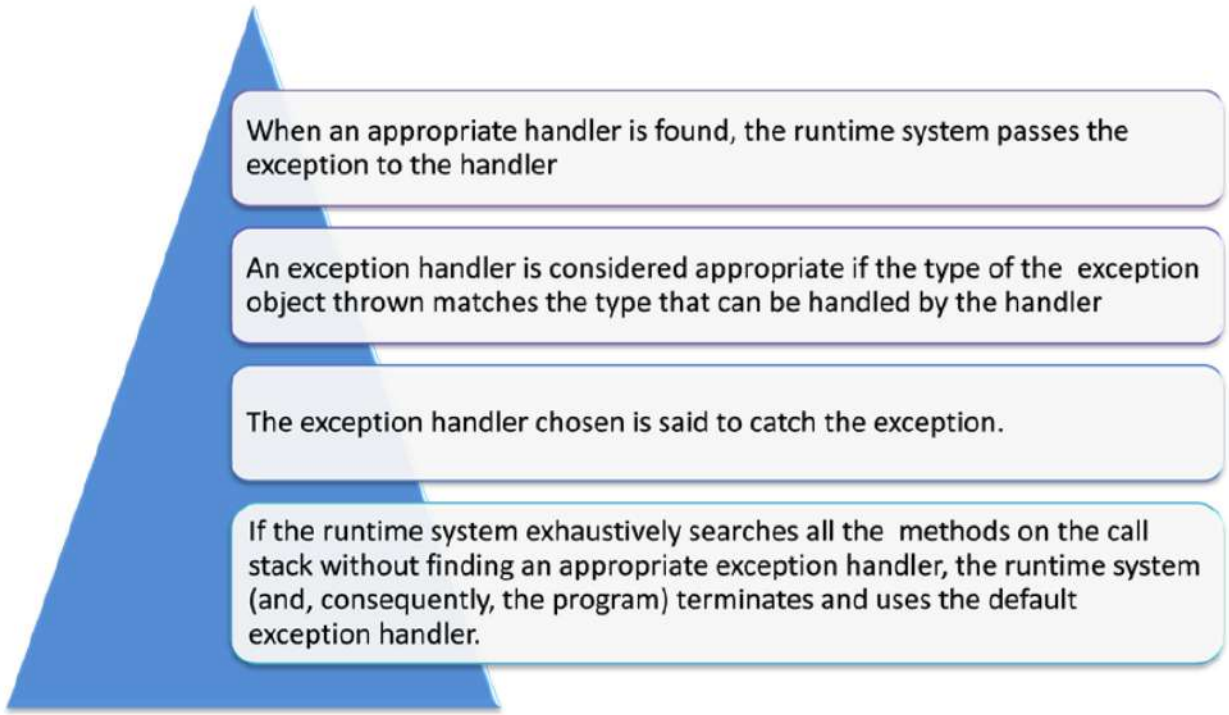
- Creating an exception object and handing it to the runtime system is called "throwing an exception"
- Exception object has information about the error, its type and state of the program when the error occurred

The runtime system searches the call stack for the method that has the block of code that handles the exception



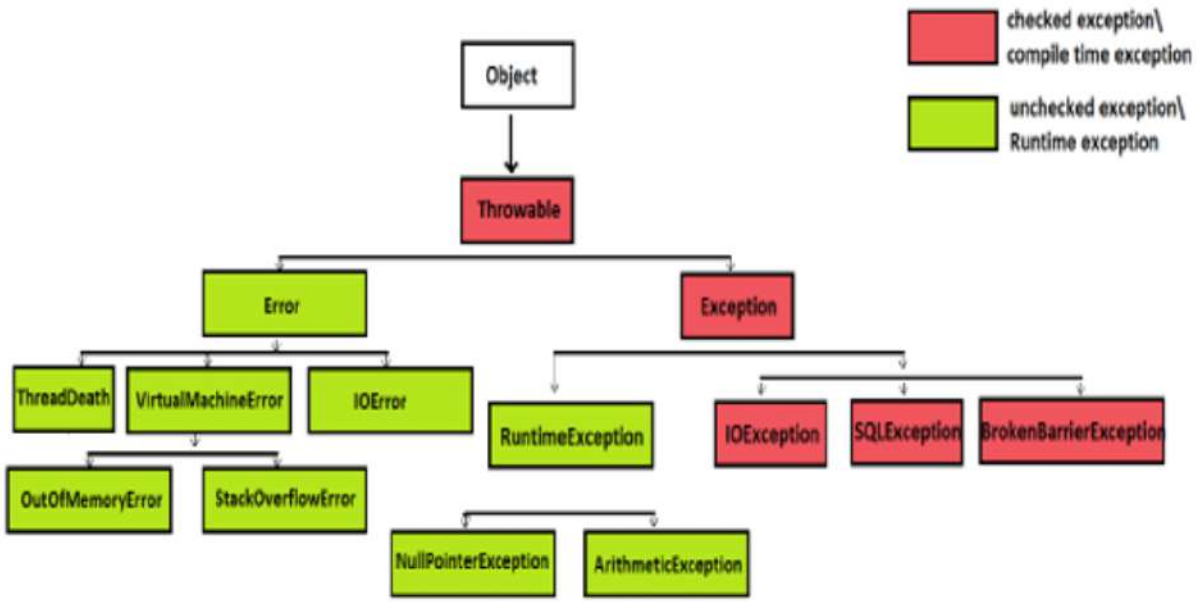
Exception

TEKNOTURF



Exception Hierarchy

TEKNOTURF



The Error class

TEKNOTURF

Errors

- Used by the Java run-time system to handle errors occurring in the run-time environment
- Generally beyond the control of user programs
- Examples
  - Out of memory errors
  - Hard disk crash

Checked and Unchecked Exception

TEKNOTURF

Checked exception

- Java compiler checks if the program either catches or lists the occurring checked exception
- Checked exceptions should be explicitly handled or properly propagated
- If not, compiler error will occur

Unchecked exceptions

- Not subject to compile-time checking for exception handling
- Built-in unchecked exception classes
  - Error
  - RuntimeException and their subclasses
- Handling all these exceptions may make the program cluttered and may become difficult to manage

Exception Example

```
public class NumberFormatExceptionDemo{
   public static void main(String[] args) {
      int sum=0;
      for(String a : args) {
           sum +=Integer.parseInt(a);    }
      System.out.println("Sum is  "+ sum);
   }
}
```

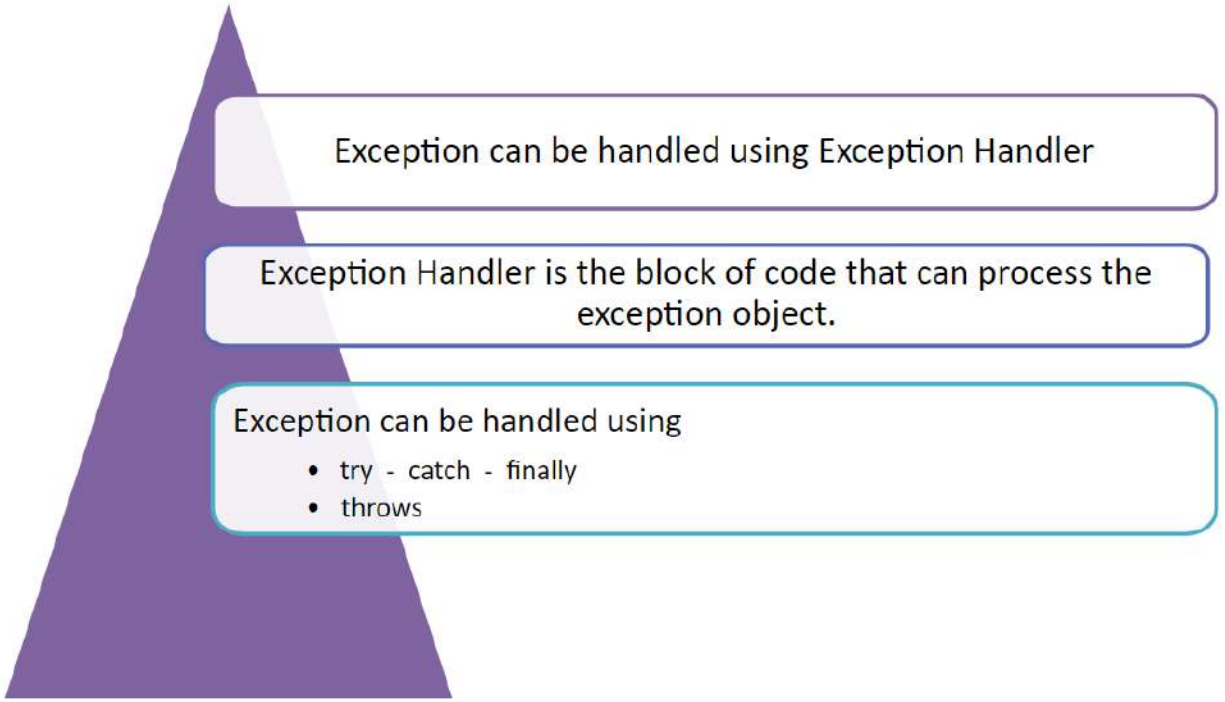
Output :

```
java  NumberFormatExceptionDemo  8  12  1  4  2
Sum is 27

java  NumberFormatExceptionDemo  8  12  four 3.0  2
      Exception in thread "main" java.lang.NumberFormatException: For input string: "four"
      at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
      at java.lang.Integer.parseInt(Integer.java:580)
      at java.lang.Integer.parseInt(Integer.java:615)
      at model.NumberFormatExceptionDemo.main(NumberFormatExceptionDemo.java:7)
```

Catching and Handling Exceptions

TEKNOTURF



Exception can be handled using Exception Handler

Exception Handler is the block of code that can process the exception object.

Exception can be handled using

- try - catch - finally
- throws

try Block

TEKNOTURF

try Block

The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block

```
try {
    code
}
catch and finally blocks . . .
```

catch Block

TEKNOTURF

Associate exception handlers with a try block by providing one or more catch blocks directly after the try block

Each catch block is an exception handler and handles the type of exception indicated by its argument

The argument type, declares the type of exception that the handler can handle and must be a class that inherits from the Throwable class

This catch block

- can contain the code to print the exception and also code to recover from the exception
- gets executed only if an exception object is thrown from its corresponding try block

No code can be between the try and the catch blocks

```
try {
} catch (ExceptionType name) {
} catch (ExceptionType name) {
}
```

The try-catch Example

TEKNOTURF

```
public class NumberFormatExceptionDemo{
   public static void main(String[] args) {
      int sum=0;
      for(String a : args) {
            try{
                sum +=Integer.parseInt(a);
            }
            catch(NumberFormatException   e) {
                //a not an int
                System.err.println(a+" is not an integer ");
            }
      }
       System.out.println("Sum is  "+ sum);
   }
}
```

```
java  NumberFormatExceptionDemo  8  12  four 3.0  2

Output :
four is not an integer
3.0 is not an integer
Sum is  22
```

finally Block

TEKNOTURF

The finally block always executes whether or not an exception occurs

It allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break.

Putting cleanup code in finally block is always a good practice

This block is optional

A try block can have multiple catch blocks , but only one finally block

```
finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

finally Block

```
public class DivideByZeroException {

   public static void main(String[] args) {
      int result  = divide(100,0);       // Line 2
      System.out.println("result : "+result);
   }

   public static int divide(int totalSum, int totalNumber)
   {
      int quotient = -1;
      System.out.println("Computing Division.");
      try{
                quotient  = totalSum/totalNumber;
      }
      catch(ArithmeticException e)  {
          System.out.println("Exception : "+
                                          e.getMessage());
      }
```

```
finally  {
     if(quotient != -1)  {
          System.out.println("Finally Block Executes");
          System.out.println("Result : "+ quotient);
     }
     else {
          System.out.println("Finally Block Executes.
                                    Exception Occurred");
     }
 }
     return quotient;
 }
}
```

Output :

Computing Division.
Exception : / by zero
Finally Block Executes. Exception Occurred
result : -1

try with Multiple catch block

TEKNOTURF

If a try block can throw multiple exceptions it can be handled using multiple catch blocks

If a try has multiple catch blocks, it should be ordered from subclass to super class

```
try {
  // code that might throw one or more exceptions

} catch (MyException e1) {
  // code to execute if a MyException exception is thrown

} catch (MyOtherException e2) {
  // code to execute if a MyOtherException exception is thrown

} catch (Exception e3) {
  // code to execute if any other exception is thrown
}
```

try with multiple catch Example

TEKNOTURF

```
public class MultipleCatchExample  {
   public static void main (String args[]) {
      int array[]={20,10,30};
      int num1=15,num2=0;
      int result=0;
      try  {
         result = num1/num2;
         System.out.println("The result is" +result);
         for(int index=0;index<3;index++)  {
              System.out.println("The value of array are" +array[index]);
         }
      }
      catch (ArrayIndexOutOfBoundsException e)  {
          System.out.println("Error. Array is out of Bounds");
       }
       catch (ArithmeticException e)  {
          System.out.println ("Can't be divided by Zero");
       }
    }
}
```

Output :
Can't be divided by Zero

In the example, the first catch block is skipped and the second catch block handles the error