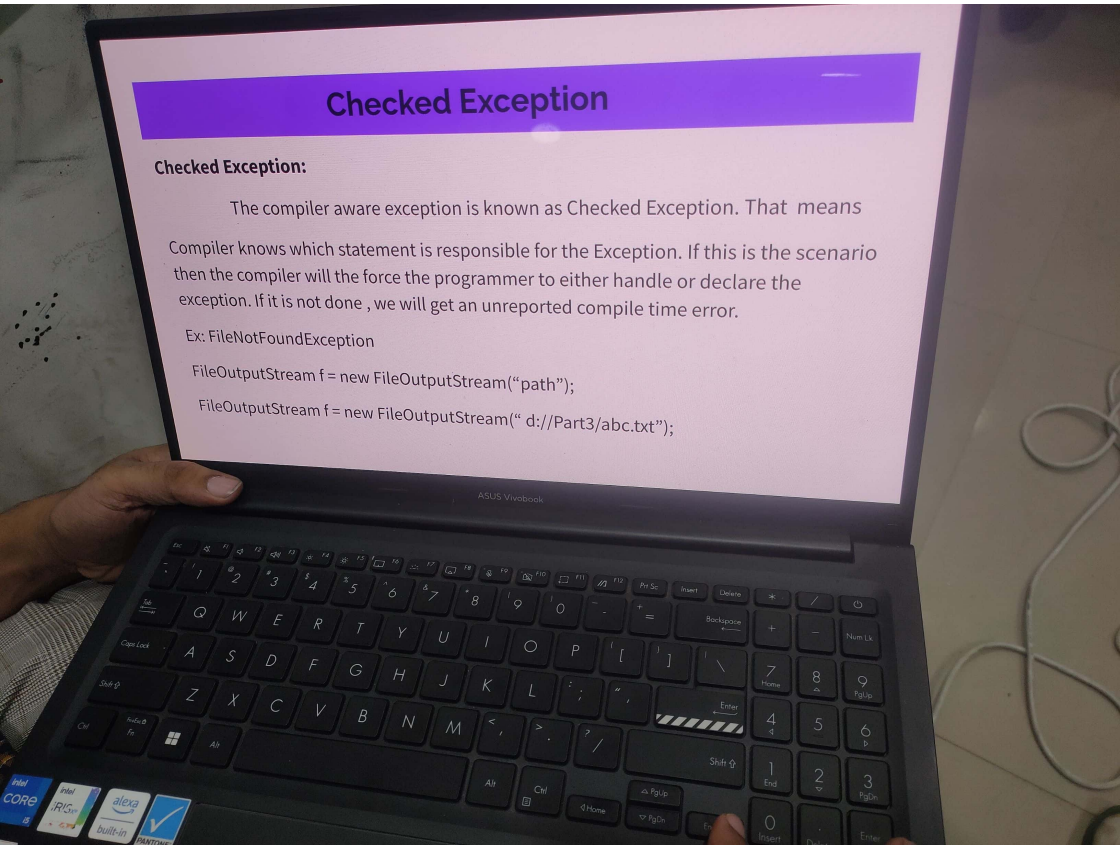
Types Of Exception

There are 2 types of Exceptions are there.

1. Checked Exception
2. Unchecked Exception



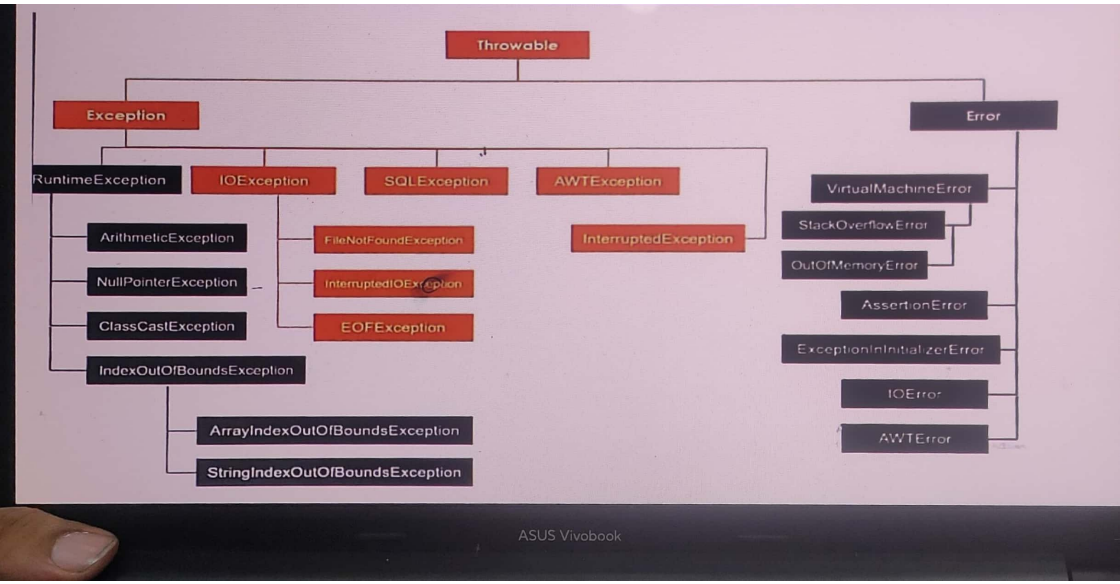
- FileNotFoundException is defined in the java.io package.
- FileNotFoundException is a checked Exception.
- We get FileNotFoundException when we try to create a file but the given path is wrong or no permission or hard disk has no sufficient memory to store the file.
- new FileOutPutStream("Path/name") -> this line is responsible for FileNotfoundException.

The compiler unaware exception is known as unchecked Exception. i .e , the compiler doesn't know the statements which are responsible for abnormal situations(Exception).

Hence , the compiler will not force the programmer either to handle or declare the exception.

Ex: ArithmeticException

C = a/b ; (if b = 0)



Note :

1. In Throwable hierarchy Error class and its subclasses and RuntimeException class and its subclasses are all known as Unchecked Exception.
2. All the subclasses of the Exception class except RuntimeException class , are considered as checked exception.
3. Throwable and Exception class are partially checked and partially unchecked classes.
4. RuntimeException and Error classes are Fully unchecked Exception classes.

- Throwable class is defined in the java.lang package.
- It is the supermost most parent class of the Exception Hierarchy (means parent of the all the Exception).
- As Object class is the parent of all the classes in java , so for the throwable class also object is parent class.
- In the throwable class toString() method of Object class is overridden , that why all the sub class of Throwable have the overridden toString() method. That's why when any of the exception occurs we are getting the fully qualified name of the exception.

  Ex : java.lang.ArithmeticException etc.

Important Methods of Throwable class

1. **String getMessage() : used to return the reason of the exception**
2. **void printStackTrace() : used to give Fully Qualified name of the exception and reason of the exception along with that flow of the exception.**

Both all the above methods are the non-static method.

Exception Handling

Exception Handling :

Exception handling is a mechanism used in java that is used to continue the normal flow of execution when the exception occured during runtime.

How to Handle the Exception

In java , we can handle exception by using the try {} and catch(){} block.

Syntax to use the try and catch block :

```
try
 {
//statements
}
catch(declare one variable of Throwable type)
{
        //statements
}
```

- **When an exception occurs :**
  1. Execution of the program will be stopped.
  2. A throwable type object will be created and reference of the created object will be thrown .

How to handle this one by using try and catch

try{ } block :

- Create try block and the statements which are responsible for exception should be written inside the try block.
- When an exception will be occurs

  1. Execution of try block is stopped.
  2. A throwable type object is created.
  3. The reference of throwable type object created is passed to the catch block.

```
try
  {
    stmt 1;
    stmt 2; // Exception occurs
Ox1
```

Throwable type object created

```
      stmt 3;
    stmt 4;
    //stmt 3 and stmt 4 will not execute

  }
  catch ( variable )
   {

    }
// Reference of the Throwable type object is
thrown to the catch block.
```

The catch block is used to catch the Throwable type reference thrown by the try block.

1. If it catches , we say the exception is handled . Statements inside the catch block are get executed and the normal flow of the program will continue.
2. If it doesn't catch , we can say the exception is not handled. Statements written inside the catch block are not executed and the program is terminated.

Q. When do we say exception is handled ?

We say exception is handled only if Exception is caught by catch block.

Exception occurs not caught

Case 1: Exception occurs not caught :

```
try
{
  sopln( 10/0);//ArithmeticException occurs
}

ArithmeticException@6788


catch ( NullPointerException e)
{
    // Not executed
}

//Not executed
```

Here the exception is not caught by the catch block . bocz it only catch the NullPointerException not ArithmeticException.

Exception occurs and caught :

Case 2: Exception occurs and caught :

```
 try {
        sopln( 10/0); // ArithmeticException occurs
         Stmt 1; //Not executed
}
catch ( ArithmeticException e )
{
     stmt 2 ; //Execute
}
Stmt 3; // Execute
```

Here it is caught by the catch block

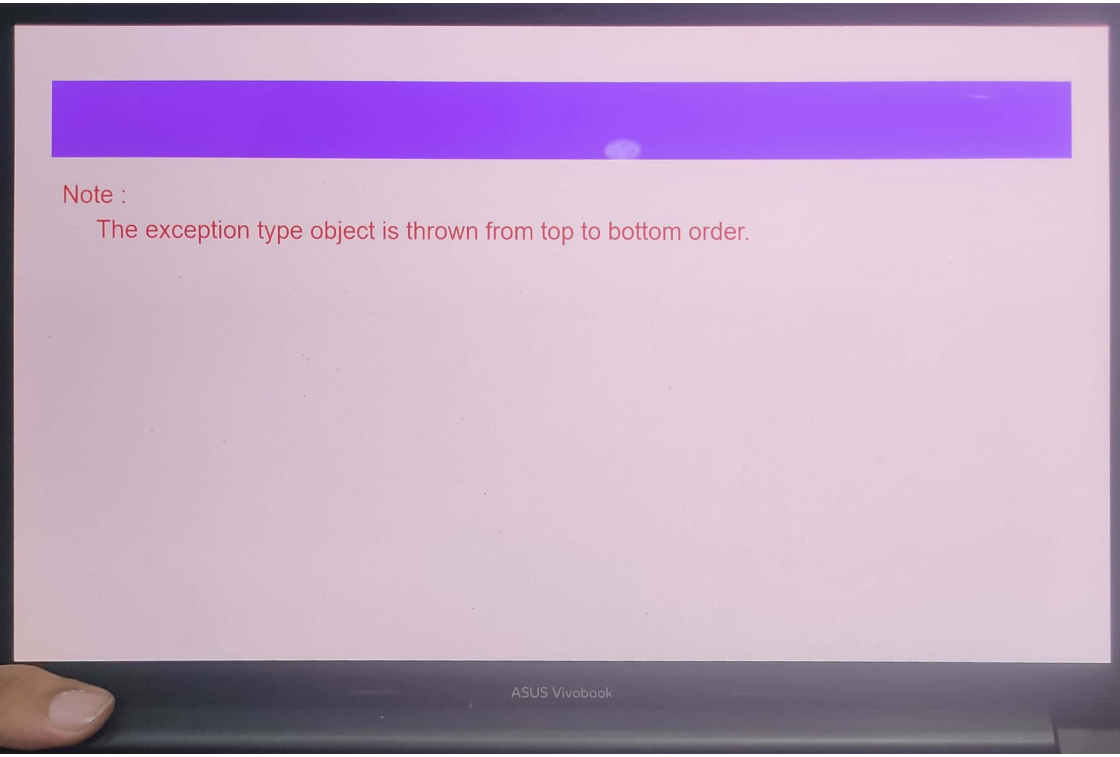
Try with multiple catch Block

Try with multiple catch block :

Try block can be associated with more than one catch blocks.

Syntax :

```
try {

}
catch (.......)
{

}
catch (......)
{

}
.
.
.
etc
```



Workflow of the try with multiple catch

Workflow :

```
try {
}
catch ()
{
}
catch ()
{
}
catch ()
{
}
.
.
.
.
```

If the exception is caught by the catch block then the try block does not throw the exception to the below catch blocks , then execution of the rest of the catch block will be skipped.

Rule :

The order of catch blocks should be maintained such that , child type should be on the top and parent type at the bottom.

Example

case 1 :

```
 try
{
   SopIn(10/0);
}
catch ( Exception e )
{

}

catch ( ArithmeticException e )
{

}
```

CTE : parent type is declared on the top and child type is on bottom.

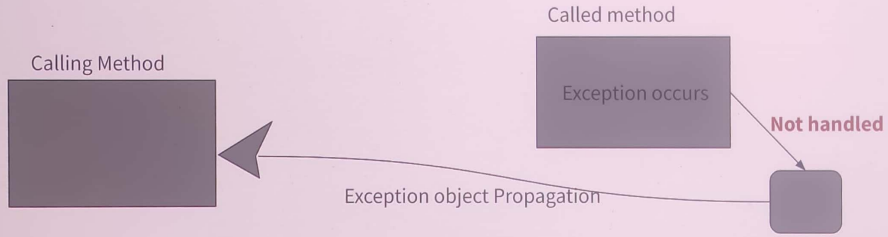
Case 2:

```
try {
 soopln(10/0);
}
catch ( ArithmeticException e )
{

}
catch ( Exception e )
{

}
```

CTS : parent type is declared on the bottom and child type is declared on the top.

Exception Object Propagation

The movement of exception from the called method to the calling method when it is not handled is known as Exception Object Program Propagation.



Note: For unchecked exception it is propagated automatically or implicitly but for checked exception it is not done implicitly , we have to do explicitly by using throws keyword.

ASUS Vivobook

Case 1: Exception occurred and handled by the calling method :

```
class Case1
{
   public static void main ( String args [] )
   {
      try
      {
         test ();
      }
      catch ( ArithmeticException e )
      {
        SopIn ( " Exception is handled by the calling method ") ;
      }
   }
static void test ()
   {
      int a = 10/0;
   }

}
```

Case 2: Exception occurred and not handled by the calling method :

```
class Case1
{
p s v m (.....)
{
  test () ;
}
static void test ()
{
int a = 10/0;
}
}
```

Exception in thread " main " java.lang.ArithmaticException : / by zero
at case 1 . test (Case1.java.8)

at case 1 .main (Case1.java:4)

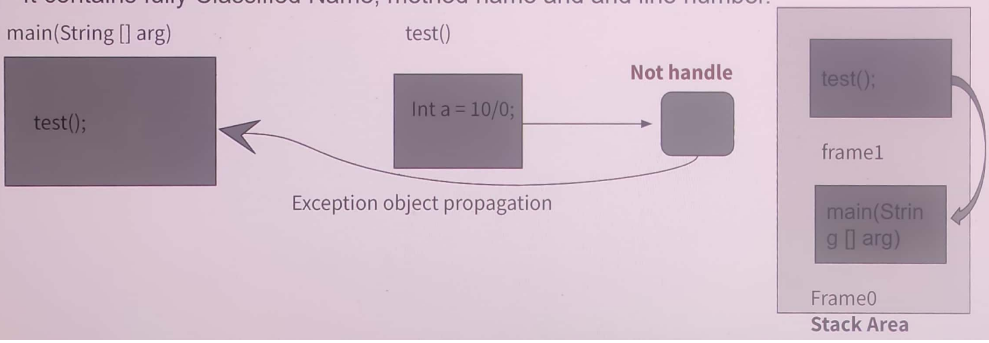
This is nothing but Exception Object propagation.

ASUS Vivobook

Stack Trace

Stack Trace:

- It provides the order in which the exception is occurred and flown from top to bottom of the stack. .
- It contains fully Classified Name, method name and and line number.



```
import java.io.FileOutputStream;

Class FileNotFoundDemo1

{  P s v m (){

// tocreate a file  demo.txt in e://file

FileOutputStream fout = new FileOutputStream( " e: //f1//demo.txt");

sopln("file created");

} }
```

We will get compile time error (because new FileOutputStream("e://f/demo.txt") is responsible for FileNotFoundException which is a checked exception and it is neither handle nor declared.

Handling checked exception

```
import java.io.FileOutputStream;
Class FileNotFoundDemo1
{  public static void main (String [] args){
// tocreate a file  demo.txt in e://file
try{
FileOutputStream fout = new FileOutputStream( " e: //f1//demo.txt");
sopln("file created");
}catch(FileNotFoundException e){
        System.out.println("Some problem is there with file , check is ");
}  } }
```

throws keyword

throws :

- It is a keyword.It is used to declare an exception which is to be thrown to the caller.
- It does not thrown an exception. It specifies that there may occurs an exception in the method.
- throws keyword should be used in the method declaration statement.
- It is used to propagate the checked exception object bcoz check exception not propagated automatically.
- We can declare more than one exceptions by providing comma.

Syntax :

```
[AM] [M] return_type methodName( [FA]) throws exception1, exception2, exception3….

{

    //stmt

}
```

Declaring Checked exception

```
import java.io.FileOutputStream;
class FileNotFoundDemo1
{
   public static void main (String [] args) throws FileNotFoundException
    {
                // tocreate a file  demo.txt in e://file
        FileOutputStream fout = new FileOutputStream( " e: //f1/demo.txt");
        sopln("file created");
   }
}
```

finally{ } :

- It is a block which is used in exception handling.
- finally block always gets executed even the exception occurs or not and handled or not.
- it is used along with try-catch block or only with try block.
- A single try block can have maximum of one finally block .
- Usually costly resources are closed in finally block (like DB connection , IO Stream).
- We can't use finally block alone .

Syntax:

```
try{

}catch(.....){

}finally{    // stmt wants to be execute if still exception is not handle

}
```

```
class Case1
{
   public static void main ( String args [] )
   {
      try
      {
         test ();
      }
      catch ( ArithmeticException e )
      {
         Sopln ( " Exception is handled by the calling method ") ;
      }
      finally
      {
           sopln("thank you");
      }
   }
static void test ()
   {
      int a = 10/0;
   }

} o/p : Exception is handled by the calling method
thank you
```

```
class Case1
{
p s v m (.....)
{
  test () ;
}
static void test ()
{
        try{
                int a = 10/0;
           }
        catch(NullPointerException e )
        {
                sopln("problem is there check");
        }
        finally
        {
                sopln("thank you");
        }
}
}
thank you
Exception in thread  " main " java.lang.ArithmaticException : / by zero
at case 1 . test ( Case1.java.8)
at case 1 .main (Case1.java:4 )
```

Note :

We can also use the finally block with try block alone.

Ex: try{

```
        // stmts
    }
    finally{
        //stmts
    }
```

Final vs finally{} vs finalize()

final : 1. final is a keyword.

2. It is applicable to class, variable and method. Final class can't be inherited, final variable can't be re-initialized and final method can't be overridden.

finally { } : 1. finally { } is a block.

2. Instruction written inside finally block will be executed even if the execution isn't handled.

finalize( ): 1. finalize( ) is a method.

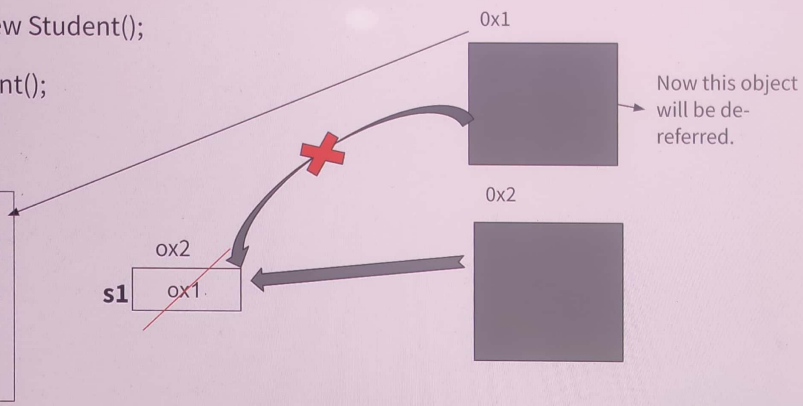
2. finalize () method is used to perform clean up processing just before the object garbage collected.

Diagram

Student s1 = new Student();

s1 = new Student();

Or s1 = null;



ASUS Vivobook

throw keyword

throw:

- It is keyword.
- It is used to throw exception manually.
- By using the throw keyword we can throw checked and unchecked exception. But it is mainly used to throw custom exception.

Syntax:

throw exception_object;

Example : throw new CustomeException("String");

Throwing the unchecked exception by throw keyword without the handling it

```
class ThrowDemo {
        public static void main (String [ ] args){
        int a = 15, b = 10;
        if(a>b)
                throw new ArithmeticException("manually thrown");
        else
                System.out.println("no exception");
System.out.println("next stmt won't be  executed");

} } // OUTPUT : Exception in thread main java.lang.ArithmeticException : manually thrown
```

Throwing the unchecked exception by throw keyword with the handling it

```
class ThrowDemo {
        public  static void main (String [ ] args){
        int a = 15, b = 10;
        if(a>b) {        try {
                throw new ArithmeticException("manually thrown");
                } catch (ArithmeticException ae){
                System.out.print("Handled"); }
        else
                System.out.println("no exception");
System.out.println("next stmt will also executed");
} } // OUTPUT :  Handlednext stmt will also executed
```

Note : if we don't handle the manually thrown exception we will get unreported exception.

Custom Exception

The user defined exception is known as the Custom exception.

Q. What is the requirements of the Custom Exception ?

Ans: When the inbuilt exceptions are not enough for the software development , we can write or define our own exceptions.(Ex : invalidPassword , InsufficientBalance etc)

How to create the Custom exception ?

1. Create a class with Exception_name which one you wants to create
2. Make the exception as child of the Any of the Exception (either checked or unchecked for example Throwable or Exception or RuntimeException etc.)
3. Override the getMessage() method (inside this which message u want to print mention there) inside the exception you created.

Types of Custom Exception

It can be

1. Custom Checked Exception.
2. Custom Unchecked Exception

Custom Checked Exception

If User defined exception's parent is partially checked or fully checked exception

Such as Exception , FileNoteFoundException,Then it will be considered as Custom Checked Exception.

Custom Unchecked Exception

If User defined exception's parent is fully unchecked exception such as RuntimeException,Then it will be considered as Custom UnChecked Exception.

Types of Custom Exception

It can be

1. Custom Checked Exception.
2. Custom Unchecked Exception

Custom Checked Exception

If User defined exception's parent is partially checked or fully checked exception

Such as Exception , FileNoteFoundException,Then it will be considered as Custom Checked Exception.

Custom Unchecked Exception

If User defined exception's parent is fully unchecked exception such as RuntimeException,Then it will be considered as Custom UnChecked Exception.

Note : if the super class method does not declare an exception,

Subclass overridden method should not declare checked exception but it can declare an unchecked exception.

If the super class method declares an exception ,

Subclass overridden method can declare the same exception or no exception or sub class exception but it should not declare a parent exception.

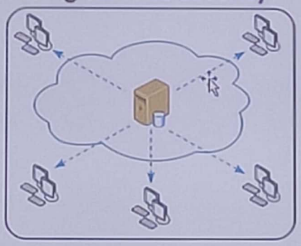
Connected to Qsp1 Disconnect

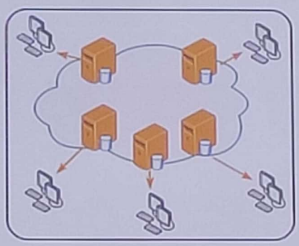
Insert Design Layout References Mailings Review View Tell me what you want to do...

Calibri (Body) 15

Font Paragraph Styles

- Improving websites load time
- Reducing bandwidth cost
- Increasing content availability and redundancy
- Improving website security





Installation: -

- Go to https://legacy.reactjs.org/