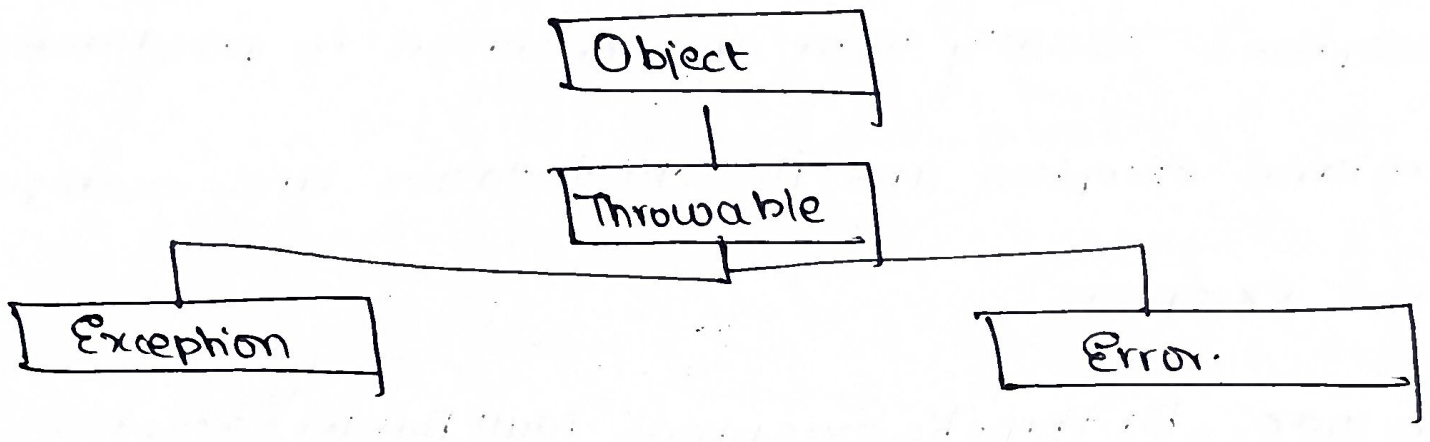
Unit 3 - Exception Handling

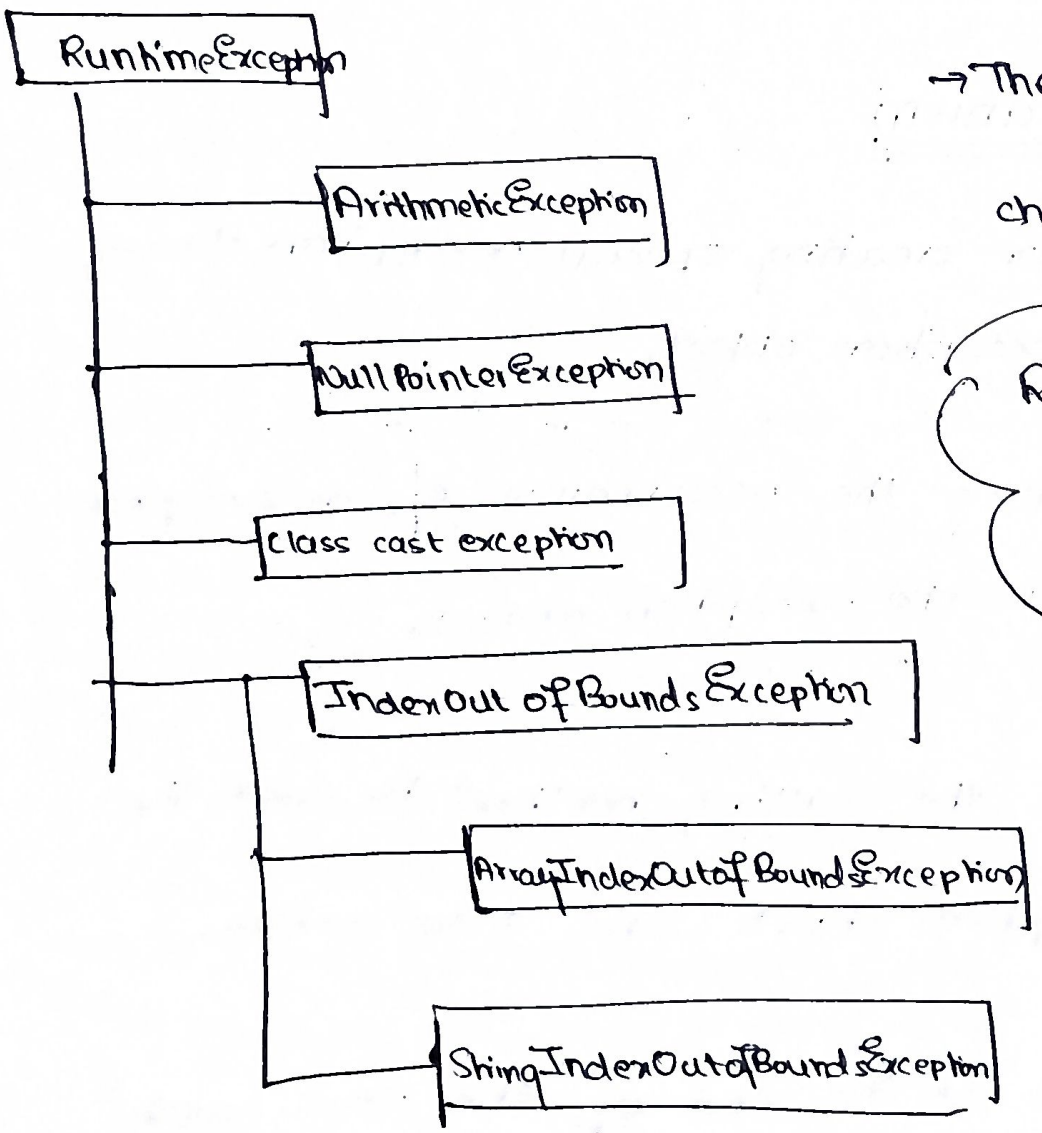
Exceptions: Exceptions are events that occur during the execution of programs that disrupt the normal flow of instructions.

They are runtime errors that are handled by exception handlers

Hierarchy:

Q subclasses extend Throwable : (i) Exception
(ii) error





→ There are 2 types of exceptions:
checked and unchecked

Runtime Exception and its sub-class are unchecked exceptions, all others are checked exceptions.

* Checked Exception

→ Exceptions checked at compile-time during compilation

For eg. If a file to be read does not exist, it could throw a FileNotFoundException.

* Unchecked Exception

→ Exceptions that are not checked at compile-time

→ These exceptions usually occur due to errors in programming.

For eg. Runtime exception and its child classes are examples of unchecked exceptions.

→ These include ArithmeticException, NullPointerException

* Exception Handling Mechanism

→ There is a mechanism for creating special exception classes, whose instances are called exception objects

(i) throw e - signals the occurrence of an exception
- e. is the exception objects

(ii) try, catch - allows the calling method to catch the thrown exception object and take action.

(iii) finally - executes at the end of the try-catch block irrespective of whether an exception was thrown or not

EXAMPLE 1

```
public class program1 {
    public static void main(String args[]) {
        try {
            int a = 5;
            int b = 0;
            System.out.println(a/b);
        }
        catch (ArithmeticException e) {
            System.out.println("can't divide by 0");
        }
        finally {
            System.out.println("All done!");
        }
```

OUTPUT

```
Can't divide by 0
All done!
```

Note: Even if the exception didn't occur, "All done!" would be printed.

EXAMPLE2 : When an exception occurs, but is not handled.

```
try {
    int a = 5;
    int b = 0;
} System.out.println(a/b);
catch (NullPointerException) {
    System.out.println("Hi");
}
Finally {
    System.out.println("All done!");
}
```

Output: Here the catch block cannot handle the exception

All done!

Exception in thread main

java.lang.ArithmeticException: / by zero

line __ : program2.java

finally, then in built error message

* Displaying the description of an Exception

→ The exception object within the catch block can be printed

for eg. for the division by zero catch block

```
catch (ArithmeticException e) {
        System.out.println(e);
}
```

OUTPUT : java.lang.ArithmeticException: / by zero

* Multiple catch clauses

→ If a program is susceptible to throw more than one kind of error, multiple catch blocks can be written.

→ When an exception is thrown, each of the catch blocks is inspected, the first one that matches is executed.

* Usage of Exception subclasses before superclasses

→ When using multiple catch statements, exception subclasses must come before any of the superclasses.

→ This is because a catch statement that uses a superclass would

~~never be reached~~ would catch exceptions of that and any of its subclasses.

→ The subclass exceptions would be unaccessible and would lead to an error.

EXAMPLE 3:

```
try {
    int a = 0;
    int b = 45 / a;
    System.out.println(b);
}
catch (Exception e) {
    System.out.println("Generic exception handler");
}
catch (ArithmeticException e) {
    S.O.P ("Div by 0");
}
```

OUTPUT

Error : The arithmetic exception class is unreachable, since the Exception class would handle all cases.

FIX : Reverse the catch statements.

Put specific catch statements in the beginning, generic Exception handler at the end.

* Examples of Exceptions and Errors

Runtime Exceptions

Arithmetic Exception

Array Index Out Of Bounds Exception

String Index Out Of Bounds Exception

Null Pointer Exception

Errors

Stack Overflow Error

Out of Memory Error

Illegal Access Error

* Possible Exception Handling Blocks

→ try, catch

→ try catch, catch

→ try, catch, finally

→ try, finally ⟶ If there's an erron order of o/p would be finally, then built in Exception.

Invalid try alone

* Try-catch blocks inside and outside functions

EXAMPLE 4:

```
public class HelloWorld {

    static void f() {
        int a = 1; b = 0;
        int c = a/b;
    }

    public static void main (String [] args) {
            s.o.p. ("Hello world");
            f();
    }
}
```

OUTPUT:

HelloWorld

<default error handler: > Exception in thread "main"
java.lang.ArithmeticException: / by zero

EXAMPLE 5

```
public class HelloWorld {
    static void f() {
        int a=1, b=0;
        try {
            int c=a/b;
        }
        catch (ArithmeticException e) {
                S.O.P("Exception in fn");
        }
    }
    public static void main(String[] args) {
        S.O.P("Hello");
        f();
    }
}
```

OUTPUT : Hello
Exception in fn

EXAMPLE 6

```
public class Hello {
    static void f() {
        int a=1, b=0;
        in c =a/b;
    }
    public static void main (String[] args) {
        S.O.P("Hello");
        try {
            f();
        }
        catch (ArithmeticException e) {
            S.O.P("Exception caught in main");
        }
    }
```

OUTPUT

Hello
Exception caught in main

EXAMPLE 7:

```
public class Hello {
    static void f() {
        int a=1, b=0;
        try {
            int c = a / b;
        }
        catch( ArithmeticException e) {
                s.o.p (" Exception caught in fn ");
        }
}
public static void main( String [ ] args ) {
        s.o.p(" Hello ") ;
        try {
            f();
        }
        catch (ArithmeticException e) {
                so.p (" Exception caught in main ");
        }
    }
}
```

OUTPUT

```
Hello
Exception caught in fn
```

EXAMPLE 8.

```
public class HelloWorld {
    static void f() {
        int a=1, b=0;
        try {
            int c = a / b;
        }
        catch (ArrayIndexOutOfBoundsException e ) {
            S.O.P (" Exception caught in fn ");
        }
}
```

```
    public static void main (String [] args) {
            S.O.P ("Hello world");
            try {
                f();
            }
            catch (ArithmeticException e) {
                S.O.P("Exception caught in main");
            }
    }
}
```

OUTPUT

Hello World
Exception caught in main

EXAMPLE 9.

```
public class Hello {
    static void f() {
        int a=1, b=0;
        try {
            int c = a/b;
        }
        catch (ArrayIndexOutOfBoundsException e) {
            S.O.P (" Exception caught in try1 ");
        }
        try {
            int s=8;
        }
        catch (ArithmeticExpression e) {
            S.O.P ("Exception caught in try2");
        }
    }
    public static void main (String [] args) {
        S.O.P("Hello World");
        f();
    }
}
```

OUTPUT : Hello World
default handler, Exception in thread "main"

* Creating User Defined Exceptions

METHOD 1

→ Create a new class that extends Exception

→ within the main body, within the try block :

```
throw new <customexceptionname> ("<msg>");
```

→ in exception, create a constructor to assign the message.

→ within the catch block, write

```
catch (<customexceptionname> e)
{
    s.o.p (e.message);
}
```

customexception
└→ defined in the class

EXAMPLE 10 : To check if a person is underage or not using a custom exception

```
class MinorException extends Exception {
    String message;
    public MinorException (String message) {
        this.message = message;
    }
}

class program2 {
    public static void main (String [] args) {
        int age = 16;
        try { if (age < 18) {
                throw new MinaException ("Under 18 years");
            }
        }
```

```
        catch (Minor Exception e) }
                S.O.P ("Error:  " + e.message);
        }
}
```

OUTPUT

Error: Under 18 years

METHOD2 : To print additional messages, along with the Exception object, override Throwable's to String() method.

EXAMPLE 11

```
class MinorException extends Exception {
    String message;
    public MinorException (String message) {
        this.message = message;
    }
    public String toString() {
        return "Default stuff:  " + message;
    }
}

class program2 {
    public static void main(String [] args) {
        int age = 13;
        try {
            if (age<18) {
                throw new MinorException("Under 18");
            }
        }
        catch (MinorException e) {
            S.O.P ("Error:  " +e);
        }
}
```

OUTPUT : Error : Default stuff : Under 18

EXAMPLE 12:

Throwing and Rethrowing Exceptions

→ Exceptions can be explicitly thrown with the throw keyword. It can also be caught multiple times.

```
class program 3 {
    static void fn() {
        try {
            throw new ArithmeticException ("throw inside fn");
        }
        catch (ArithmeticExpression e) {
            System.O.P.("Inside function: " + e);
            throwe;
    }   }

public static void main (String[] args) {
        try {
            fn();
        }
        catch(ArithmeticException e){
            s.o.p ("Main body catch: " + e);
        }
}}
```

OUTPUT

Inside function : java.lang. ArithmeticExpression : throw insidefn

Mainbody catch: java.lang. ArithmeticExpression: throw insidefn

Throws

→ If a method is capable of throwing an exception it does not handle, it must specify this behaviour so that the callers can guard themselves against the exception

Case1: When a function may cause a pre-defined exception

EXAMPLE 13

```
class program 4 {
    static void divideNums(int a, int b) throws ArithmeticException{
        if (b==0) {
            throw new ArithmeticException("Error in fn");
        }
}
    public static void main (String[] args) {
        try {
            divideNums(6,0);
        }
        catch (ArithmeticException e) {
            System.out.println("Exception caught: " + e);
        }
}}
```

OUTPUT:

~~Error in fn: java.lan~~

Exception caught : java.lang.ArithmeticException : Errron in fn

Case2: When the function may cause a user defined exception

EXAMPLE13:

```
class TooShortStringException extends Exception {
    String message;
    TooShortStringException (String message) {
        this.message = message;
    }
    public String toString(){
        return message + " is too short";
    }
}

class program5 {
    static void lengthCheck (String str) throws TooShortStringException{
        if (str.length()<5 ){
            throw new TooShortStringException (str);
        }
    }
    public static void main (String [] args) {
        String sh = "cat";
        try {
            lengthCheck(str);
        }
        catch (TooShortStringException e) {
            S.O.P ("Message: " +e);
        }
}
```

OUTPUT

Message: cat is too short

* Throwing Exceptions between Fn. calls

EXAMPLE 14 :

```
class z {
    void m() {
        try {
            n();
        }
        catch (myex iie) {
            SOP ("Exception caught" + iie);
        }
    void n() throws myex {
        p();
    }
    void p() throws myex {
        throw new myex ("Mine");
    }
    public static void main (String [] args) {
        z h = new z();
        h.m();
    }
}
```

OUTPUT

Exception caught : mine

EXPLANATION

m() calls n()

n() calls p()

p() goes back to n()

n() goes back to m()

EXAMPLE 15: When there is no handler

```
class z {
    void m() throws myex {
        n();
    }
    void n() throws myex {
        p();
    }
    void p() throws myex {
        throw new myex ("Mine");
    }
Public static void main ( String [] args ) throws myex {
        zh = new z();
        h.m();
    }
}
```

OUTPUT Exception in thread "main" Mine
at z.p (z.java <line no>)
at z.n (z.java <line nos>)
at z.m (z.java < line nos)
at z.main (z.java <line nos>)

Finally : a block of code that is executed after a try/catch block.

The finally block executes whether or not there is an exception.

NOTE : Any time a method is about to return to the calla from inside a try catch block, via an uncaught or explicit return the finally block runs before the method returns.

EXAMPLE 16:

```
class z {
    Void m() throws Exception {
        try { s.o.p ("m");
        }
        finally { s.o.p ("m finally");
        }
    public static void main (String [] args) {
        z z1 = new z();
        try {
            z1.m();
        }
        catch (Exception e) {
        }     s.o.p ("Exception caught");
    }
}
```

OUTPUT : m
m finally

EXAMPLE17:

```
class z {
    void m() throws Exception {
        try { return;
        }
        catch (Exception e) { }
        finally { s.o.p ("m finally"); }
    public static void main (String [] args) {
        z z1 = new z();
        try { z1.m(); }
        catch ( Exception e) {
            s.o.p ("Exception caught");
        }
    }
}
```

OUTPUT : m finally

EXAMPLE 18:

```
class z {
    void m() throws Exception {
        try { throw new Exception (); }
        finally { s.o.p ("m finally"); }
    public static void main ( String [ ] args ) throws Exception {
        z z1 = new z();
        try { z1. m(); }
        catch ( Exception e) {
            s.o.p ( " Exception caught " );
    }   }
}
```

OUTPUT
m finally
Exception caught

Packages

→ Packages are a way of grouping a no. of related class and / or interfaces together into a single unit. Packages act as containers for classes.

Benefits of packages

→ classes contained within packages can be reused.

→ 2 classes in 2 diff. packages can have the same name.

→ Classes in packages can be hidden, if one doesn't want other packages to access them.

6 Foundation Packages in Java

→ java.lang

→ java.util

→ java.io

→ java.awt

→ java.net

→ java.applet

Accessing classes inside packages

→ using the fully qualified class name

for eg. java.~~lang~~ lang.Math.sqrt(x)

→ importing selected classes

import package.class

→ importing all the classes inside a package.

import package.*

Making Packages

To create a ~~package~~ package myPack, include the following statement at the beginning

```
package myPack;
.
.
.
include classes.
```

The name of the java file is the classname.java.

EXAMPLE

To create a package called Shapes with 2 classes, Circle & Square in it.

Folder name : Shapes

Circle.java

```
package shapes
public class Circle {
    const()
    fn()
    fn()
}
```

Square.java

```
package shapes
public class Square {
    const()
    fn()
    fn()
}
```

To use the package Shapes in a program, use

import Shapes.*;

* Creating Subpackages

To define package : myPackage.insideOnePack.insideTwoPack

To access : import myPackage.insideOnePack.insideTwoPack;

* Visibility of Packages

Accessible To	public	protected	package	private
same class	Yes	Yes	Yes	Yes
class in subpackage	Yes	Yes	Yes	No
subclass in diff package	Yes	Yes	No	No
non-subclass diff. package	Yes	No	No	No