

MODULE 3

CHAPTER 1 PACKAGES INTERFACES & EXCEPTION HANDLING

Prepared By Mr.EBIN PM, AP, IESCE

1

PACKAGES

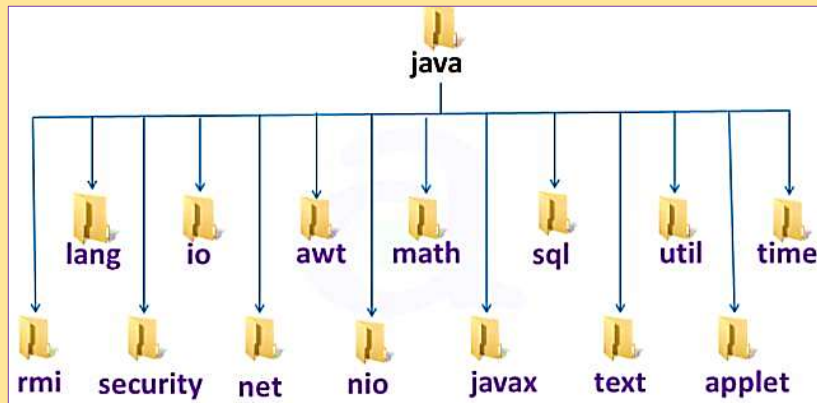
- A package in Java is used to **group related classes and interfaces**
- Think of it as a folder in a file directory.
- We use packages to **avoid name conflicts**, and to write a better maintainable code
- Packages in Java is a mechanism to **encapsulate** a group of classes, interfaces and sub packages which is used to **providing access protection**
- Package in Java can be categorized in two form,
 - built-in package**
 - user-defined package**

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE

2

- **Built-in Package:-** Existing Java package. for example, java.io.*, java.lang , java.util etc.
- **User-defined-package:-** Java package created by user to categorized classes and interface



Prepared By Mr.EBIN PM, AP, IESCE

EDULINE

3

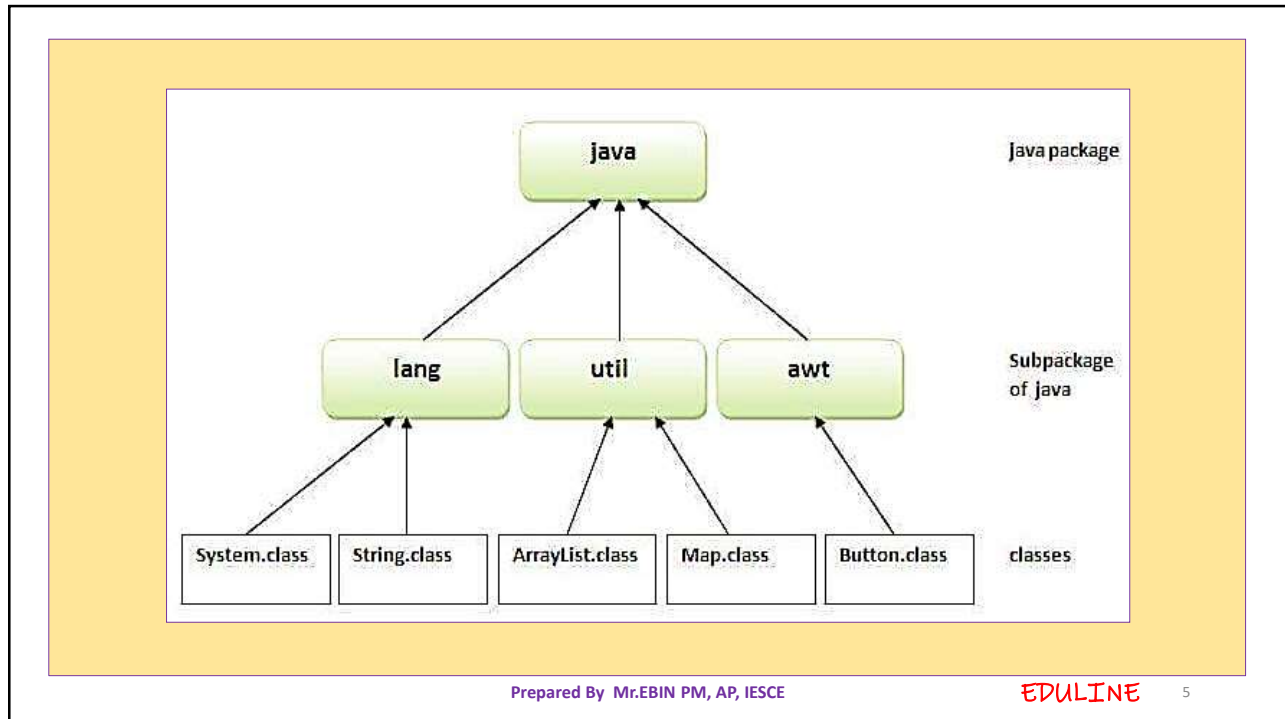
❖ Advantage of Java Package

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) In real life situation there may arise scenarios where we need to define files of the same name. This may lead to name-space collisions. Java package removes naming collision.
- 4) **Reusability:** Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency. A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.
- 5) Easy to locate the files.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE

4



➤ To use a class or a package from the library, we need to use the **import** keyword:

Syntax:

```
import package.name.Class; // Import a single class
import package.name.*; // Import the whole package
```

➤ The **package** keyword is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
public static void main(String args[]){
System.out.println("Welcome to package");
}
}
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 6

Access Packages from another package

There are three ways to access the package from outside the package.

```
import package.*;  
import package.classname;  
fully qualified name
```

1. Using packagename.*

- If we use **packagename.*** then all the classes and interfaces of this package will be accessible but **not subpackages**.
- The **“import”** keyword is used to make the classes and interface of another package accessible to the current package.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 7

Example of package that import the packagename.*

```
//save by A.java  
package pack;  
public class A{  
    public void msg(){System.out.println("Hello");}  
}
```

```
//save by B.java  
package mypack;  
import pack.*;  
  
class B{  
    public static void main(String args[]){  
        A obj = new A();  
        obj.msg();  
    }  
}
```

Output:Hello

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 8

2. Using packagename.classname

- If you import **package.classname** then only declared class of this package will be accessible.

- Example



Output: Hello

```
//save by A.java
package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.A;

class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE

9

3. Using fully qualified name

- If we use fully qualified name then only declared class of this package will be accessible.
- Now there is **no need to import**. But you need to use fully qualified name every time when you are accessing the class or interface.
- It is generally used when **two packages have same class name** e.g. **java.util** and **java.sql** packages contain **Date** class.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE

10

- Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

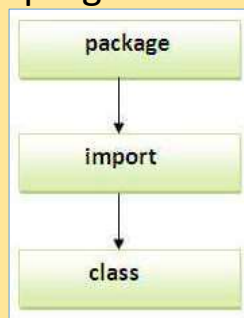
//save by B.java
package mypack;
class B{
    public static void main(String args[]){
        pack.A obj = new pack.A();//using fully qualified name
        obj.msg();
    }
}
```

Output:Hello

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 11

- If we import a package, subpackages will not be imported.
 - If we import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages.
 - Hence, you need to import the subpackage as well
- Note:** Sequence of the program must be **package** then **import** then **class**.



Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 12

INTERFACE

- An interface in Java is a **blueprint of a class**. It has static constants and abstract methods.
- The interface in Java is a mechanism to **achieve abstraction**. There can be only abstract methods in the Java interface, not method body. It is used to **achieve abstraction and multiple inheritance** in Java.
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 13

- Like abstract classes, interfaces cannot be used to create objects
- Interface methods do not have a body - the body is provided by the "**implement**" class
- On implementation of an interface, you must override all of its methods
- **Interface methods** are by default **abstract and public**
- **Interface attributes** are by default **public, static and final**
- An interface cannot contain a constructor (as it cannot be used to create objects)

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 14

❖ Declare an interface

- An interface is declared by using the **interface** keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

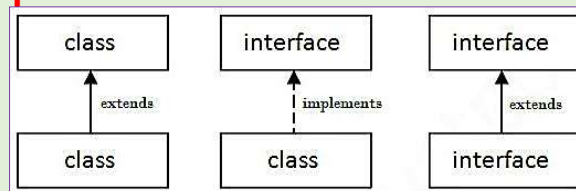
```
// interface
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void run(); // interface method (does not have a body)
}
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 15

- To access the interface methods, the interface must be "implemented" by another class with the **implements** keyword (instead of extends).
- The body of the interface method is provided by the "implement" class

❖ The relationship between classes and interfaces



- As shown in the figure given above, a class extends another class, an interface extends another interface, but a class implements an interface.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 16


```
// Interface
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
    public void sleep() {
        // The body of sleep() is provided here
        System.out.println("Zzz");
    }
}

class MyMainClass {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

Output

```
The pig says: wee wee
Zzz
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 17

❖ Why And When To Use Interfaces

1) **To achieve security** - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance". However, it can be achieved with interfaces, because the class can implement multiple interfaces.

➤ **Note:** To implement multiple interfaces, separate them with a comma (see example below).

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 18

```
interface FirstInterface {  
    public void myMethod(); // interface method  
}  
  
interface SecondInterface {  
    public void myOtherMethod(); // interface method  
}  
  
class DemoClass implements FirstInterface, SecondInterface {  
    public void myMethod() {  
        System.out.println("Some text..");  
    }  
    public void myOtherMethod() {  
        System.out.println("Some other text...");  
    }  
}  
  
class MyMainClass {  
    public static void main(String[] args) {  
        DemoClass myObj = new DemoClass();  
        myObj.myMethod();  
        myObj.myOtherMethod();  
    }  
}
```

Output

```
Some text...  
Some other text...
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE

19

EXCEPTION HANDLING

- Exception is an **abnormal condition**.
- In Java, an exception is an event that **disrupts the normal flow** of the program. **It is an object** which is thrown at runtime.
- Exception Handling is a mechanism to **handle runtime errors** such as **ClassNotFoundException**, **IOException**, **SQLException**, **RemoteException**, etc.
- The core advantage of exception handling is to maintain the normal flow of the application.
- An exception normally disrupts the normal flow of the application that is why we use exception handling.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE

20

Let's take a scenario:

- Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed.
- If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

```
statement 1;  
statement 2;  
statement 3;  
statement 4;  
statement 5;//exception occurs  
statement 6;  
statement 7;  
statement 8;  
statement 9;  
statement 10;
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 21

❖ Types of Java Exceptions

- There are mainly two types of exceptions: checked and unchecked.
- Here, an **error** is considered as the **unchecked exception**.
- According to Oracle, there are three types of exceptions:

Checked Exception
Unchecked Exception
Error



Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 22

➤ Checked Exception

- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions
- e.g. IOException, SQLException etc.
- Checked exceptions are **checked at compile-time**.

➤ Unchecked Exception

- The classes which inherit RuntimeException are known as unchecked exceptions
- e.g. ArithmeticException, NullPointerException,
- Unchecked exceptions are not checked at compile-time, but they are **checked at runtime**

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 23

➤ Error

- Error is **irrecoverable**
- e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

❖ Java Exception Keywords

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 24

❖ Common Scenarios of Java Exceptions

➤ A scenario where ArithmeticException occurs

- If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0; //ArithmeticException
```

➤ A scenario where NullPointerException occurs

- If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s=null;
```

```
System.out.println(s.length()); //NullPointerException
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 25

➤ A scenario where NumberFormatException occurs

- The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
```

```
int i=Integer.parseInt(s);//NumberFormatException
```

➤ A scenario where ArrayIndexOutOfBoundsException occurs

- If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
```

```
a[10]=50; //ArrayIndexOutOfBoundsException
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 26

TRY & CATCH

- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The try and catch keywords come in pairs

Syntax

```
try {  
    // Block of code to try  
}  
catch(Exception e) {  
    // Block of code to handle errors  
}
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 27

Consider the following example

```
public class MyClass {  
    public static void main(String[ ] args) {  
        int[] myNumbers = {1, 2, 3};  
        System.out.println(myNumbers[10]); // error!  
    }  
}
```

This will generate an error, because myNumbers[10] does not exist.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10  
    at MyClass.main(MyClass.java:4)
```

- If an error occurs, we can use **try...catch** to catch the error and execute some code to handle it

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 28

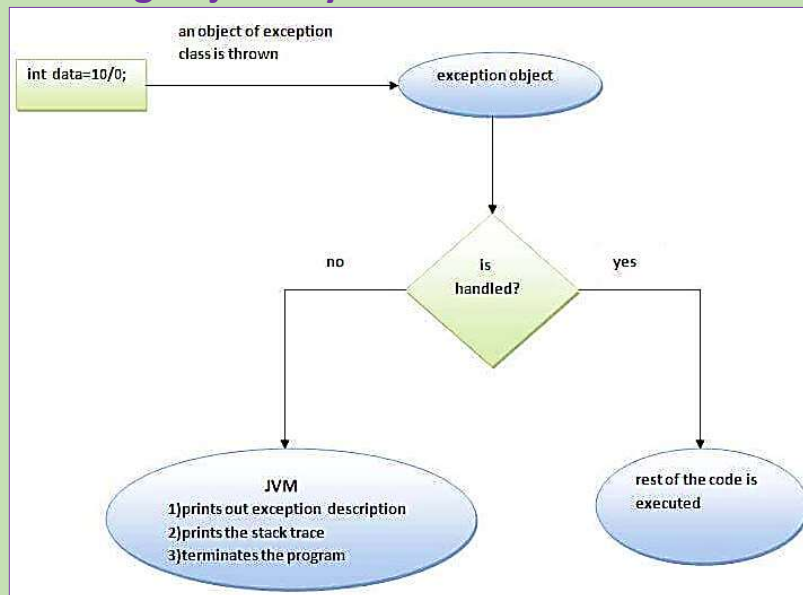
```
public class MyClass {  
    public static void main(String[] args) {  
        try {  
            int[] myNumbers = {1, 2, 3};  
            System.out.println(myNumbers[10]);  
        } catch (Exception e) {  
            System.out.println("Something went wrong.");  
        }  
    }  
}
```

Output**Something went wrong.**

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 29

❖ Internal working of java try-catch block



Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 30

- The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:
 - Prints out exception description.
 - Prints the stack trace (Hierarchy of methods where the exception occurred).
 - Causes the program to terminate.
- But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 31

Multi-catch block

- A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 32


```

public class MultipleCatchBlock1 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
        }
        catch(ArithmeticException e)
        {
            System.out.println("Arithmetic Exception occurs");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("ArrayIndexOutOfBoundsException occurs");
        }
        catch(Exception e)
        {
            System.out.println("Parent Exception occurs");
        }
        System.out.println("rest of the code");
    }
}

```

Output

Arithmetic Exception occurs
rest of the code

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 33

Nested try block

- The try block within a try block is known as nested try block in java.
- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

```

....
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
....

```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 34

```
class Excep6{
public static void main(String args[]){
try{
try{
System.out.println("going to divide");
int b =39/0;
}catch(ArithmeticException e){System.out.println(e);}

try{
int a[]=new int[5];
a[5]=4;
}catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

System.out.println("other statement");
}catch(Exception e){System.out.println("handed");}

System.out.println("normal flow..");
}
}
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 35

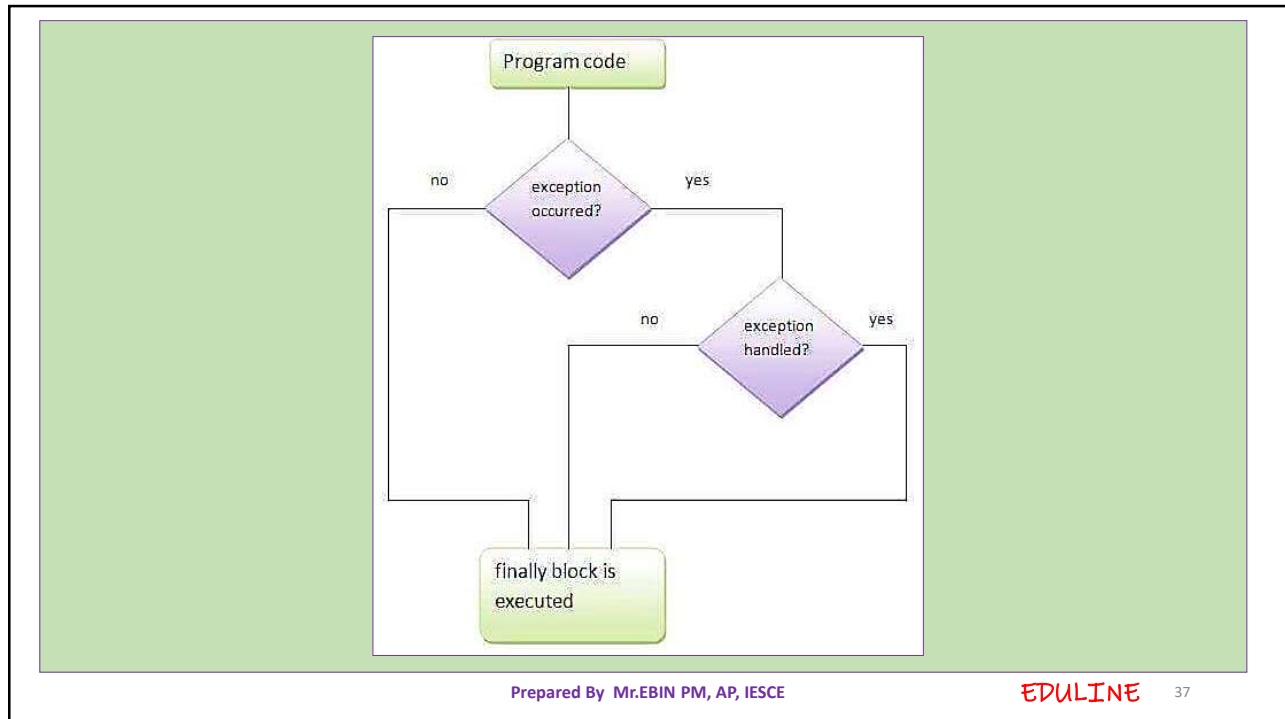
finally block

- Java finally block is a block that is used to execute important code such as **closing connection**, **stream** etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.

Note: If you don't handle exception, before terminating the program, JVM executes finally block(if any).

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 36



❖ Usage of Java finally

- **Case 1** - Let's see the java finally example where **exception doesn't occur**.

```

class TestFinallyBlock{
    public static void main(String args[]){
        try{
            int data=25/5;
            System.out.println(data);
        }
        catch(NullPointerException e){System.out.println(e);}
        finally{System.out.println("finally block is always executed");}
        System.out.println("rest of the code...");
    }
}
  
```

```

Output:5
        finally block is always executed
        rest of the code...
  
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 38

Case 2 - Let's see the java finally example where **exception occurs and not handled.**

```
class TestFinallyBlock1{
    public static void main(String args[]){
        try{
            int data=25/0;
            System.out.println(data);
        }
        catch(NullPointerException e){System.out.println(e);}
        finally{System.out.println("finally block is always executed");}
        System.out.println("rest of the code...");
    }
}
```

```
Output:finally block is always executed
        Exception in thread main java.lang.ArithmeticException:/ by zero
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 39

Case 3 - Let's see the java finally example where **exception occurs and handled**

```
public class TestFinallyBlock2{
    public static void main(String args[]){
        try{
            int data=25/0;
            System.out.println(data);
        }
        catch(ArithmeticException e){System.out.println(e);}
        finally{System.out.println("finally block is always executed");}
        System.out.println("rest of the code...");
    }
}
```

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
        finally block is always executed
        rest of the code...
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 40

throw keyword

- The Java throw keyword is used to **explicitly throw an exception**.
- We can throw either checked or unchecked exception in java by throw keyword

Output

```
Exception in thread main java.lang.ArithmeticException: not valid
```

```
public class TestThrow1{
    static void validate(int age){
        if(age<18)
            throw new ArithmeticException("not valid");
        else
            System.out.println("welcome to vote");
    }
    public static void main(String args[]){
        validate(13);
        System.out.println("rest of the code...");
    }
}
```

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 41

throws keyword

- The Java throws keyword is used to **declare an exception**.
- It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained
- Exception Handling is mainly used to handle the checked exceptions.
- If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 42

Syntax of java throws

```
return_type method_name() throws exception_class_name{  
    //method code  
}
```

Which exception should be declared

- checked exception only, because:
- unchecked Exception: under your control so correct your code.
- error: beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 43

Eg:

```
import java.io.*;  
class ThrowExample {  
    void myMethod(int num)throws IOException, ClassNotFoundException{  
        if(num==1)  
            throw new IOException("IOException Occurred");  
        else  
            throw new ClassNotFoundException("ClassNotFoundException");  
    }  
}  
  
public class Example1{  
    public static void main(String args[]){  
        try{  
            ThrowExample obj=new ThrowExample();  
            obj.myMethod(1);  
        }catch(Exception ex){  
            System.out.println(ex);  
        }  
    }  
}
```

Output

Output:

java.io.IOException: IOException Occurred

Prepared By Mr.EBIN PM, AP, IESCE

EDULINE 44