**TASK 11**

1. **Access Modifiers :**

Access Modifiers are the keywords which defines the accessibility of the Class and its members.

Access modifiers are used in java to control the visibility of the program or block of code.

It helps to secure and provide authenticity to the code block.

**Real Time Example:**

**Facebook**

If you keep the status or profile picture in facebook, it provides three options for the post

|  |  |  |
| --- | --- | --- |
| Public | Public | Anyone can view the post or status |
| Protected | Friends | Your Friends on facebook/Friends of friends |
| Friends Except | Don’t show to some friends |
| Private | Only me | Only the user posting can see it |

**Four Access modifiers available in java**:

* Default Access Modifiers
* Private Access Modifiers
* Public Access Modifiers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accessibility | Default | Private | Protected | Public |
| Inside Same Class | Yes | Yes | Yes | Yes |
| Subclass of a class | Yes | No | Yes | Yes |
| Same Package Different class | Yes | No | Yes | Yes |
| Different Package Subclass | No | No | Yes | Yes |
| Different Package Non-Sub class | No | No | No | Yes |

**Examples**

**Default Access Modifier:**

* The access level is only within the package.
* If access modifier has no modifier specified, then its Default.
* It implies that the Classes, Data Member and methods are accessible within the same package.

**Example: Accessing the Default Class Within the Package**

**Package 1 Class 1:**

**package** task11;

//creating a default access modifier for class defaultDemo1

**public** **class** DefaultDemo {

//Creating a method disp

**public** **void** disp() {

System.***out***.println("This is default method");

}

}

**Package 1 Class 2:**

**package** task11;

**public** **class** DefaultDem {

//creating another class and calling the disp method

**public** **static** **void** main(String[] args) {

//creating object for class

DefaultDemo obj=**new** DefaultDemo();

//Calling the public method of defaultDemo1 class

obj.disp();

}

}

**Example: Accessing the Default Class Outside the Package**

**Package 1:**

**package** classtask;

**public** **class** defaultAMDemo {

**void** disp() {

System.***out***.println("Hi, I am Default!");

}

}

**Package 2:**

**package** classtaskDemo;

**import** classtask.defaultAMDemo;

**public** **class** Main {

**public** **static** **void** main(String args[]) {

defaultAMDemo obj=**new** defaultAMDemo();

obj.disp();

}

}

**OUTPUT:(Compile time Error)**

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method disp() from the type defaultAMDemo is not visible

at classtaskDemo.Main.main(Main.java:9)

* The scope of class A and its method disp() is default so it cannot be accessed from outside the package.

**Private Access Modifiers** :

* This modifier access is only within the Class.
* It cannot be used outside the class.

**Example:**

**package** classtask;

**class** privateAMDemo {

**private** **int** i=5; // Private variable declararion

**private** **int** disp() { //Private method

System.***out***.println(i);

**return** i;

}

**public** **class** Main{

**public** **static** **void** main(String[] args) {

privateAMDemo obj=**new** privateAMDemo(); // object creation for the class

**//Compile Time error**

**int** val=obj.i; //trying to access the private variable from another class

System.***out***.println(val);

**//Compile time error**

System.***out***.println("Hi, I am Abi " + obj.disp()); // Trying to access the private method from another class

}

}

}

**OUTPUT:**

Private variable cannot be accessed outside the defined block.

**Protected Access Modifiers:**

This accesses modifier access level is within the package and outside the package through the child class.

If you do not make the child class, it cannot be accessed form the package outside.

**Example:**

**Package 1:**

**package** classtask;

**public** **class** ProtectedAMDemo {

**protected** **void** msg(){System.***out***.println("Hello this is Protected");}

}

**Package 2:**

**package** classDemo;

**import** classtask.ProtectedAMDemo;

**public** **class** PrivateDemo1 **extends** ProtectedAMDemo {

**public** **static** **void** main(String[] args) {

PrivateDemo1 obj = **new** PrivateDemo1();

obj.msg();

}

}

**OUTPUT:**

Hello this is Protected

**Public Access Modifiers:**

* The public access modifier is accessible everywhere.
* It has the widest scope among all other modifiers.

**Example:**

**Package 1:**

**package** classtask;

**public** **class** ProtectedAMDemo {

**protected** **void** msg(){System.***out***.println("Hello this is Public");}

}

**Package 2:**

**package** classDemo;

**import** classtask.ProtectedAMDemo;

**public** **class** PrivateDemo1{

**public** **static** **void** main(String[] args) {

PrivateDemo1 obj = **new** PrivateDemo1();

obj.msg();

}

}

**OUTPUT:**

Hello this is Public

1. **Difference between Error and Exception**

Both exceptions and errors are the subclasses of a throwable class. The error implies a problem that mostly arises due to the shortage of system resources. On the other hand, the exceptions occur during runtime and compile time. Let’s find out some major differences between exceptions and errors.

**Errors:**

* The error signifies a situation that mostly happens due to the absence of system resources.
* The system crash and memory errors are an example of errors.
* Error is an unexpected event that cannot be handled at runtime.
* Errors can terminate your program.
* Most of the time, programs cannot recover from an error.
* Errors cannot be caught or handled.
* They are generally caused by the environment in which the code is running.
* e.g, The image below is seen by almost all the computer users.

There are 3 types of Errors:

* Syntax Error
* Runtime Error
* Logical Error

**Syntax Errors:**

* Syntax Errors are those errors detected during the compilation phase by the compiler when your code does not follow the syntactical rules of the programming language you are using.
* e.g, missing semi-colon(s), missing parenthesis, using else if() block directly without using if block first, returning nothing from the function when the return type is some data type, say, int, etc.

**Explanation:**

Since we have used else if and else condition blocks without providing an if condition block, which is a syntax error, so the compilation is terminated.

**Runtime Errors :**

* Runtime Errors occur during the execution of a program, due to lack of system resources, or due to irrelevant input by the user.
* The compiler has no idea whatsoever how to detect these kinds of errors.
* For example, dividing a number by 0, accessing an element from an array that is out of range, trying to convert an invalid string to an integer, out of memory error, etc.

**Example :**

In the code below, we are trying to find the count of the odd numbers present in an integer array.

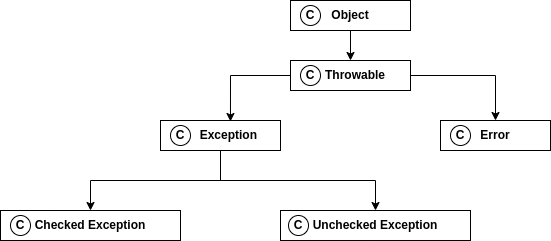
**Logical Errors :**

* Logical Errors are those errors where the program returned incorrect results when you were expecting the desired result.
* These occur due to some mistake in the code logic made by the programmer.
* The compiler cannot detect these errors.
* The user can just understand them after seeing the output.
* These are also known as Semantic Errors.

**Exceptions:**

* The exceptions are the issues that can appear at runtime and compile time.
* It majorly arises in the code or program authored by the developers.
* There are two types of exceptions:
  + Checked exceptions and Unchecked exceptions.

**Exception Hierarchy:**



**Examples:**

**ArithmeticException**:

It is thrown when an exceptional condition has occurred in an arithmetic operation.

// Java program to demonstrate ArithmeticException

**public** **class** ArithmeticException\_Demo {

**public** **static** **void** main(String[] args) {

**try** {

**int** a = 30, b = 0;

**int** c = a/b; // cannot divide by zero

System.***out***.println ("Result = " + c);

}

**catch**(ArithmeticException e) {

System.***out***.println ("Can't divide a number by 0");

}

}

}

**OUTPUT:**

Can't divide a number by 0

**ArrayIndexOutOfBoundsException:**

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

**package** classtask;

**public** **class** ArrayIndexOutOfBound\_Demo {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try**{

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

a[6] = 9; // accessing 7th element in an array of

// size 5

}

**catch**(ArrayIndexOutOfBoundsException e){

System.***out***.println ("Array Index is Out Of Bounds");

}

}

}

**OUTPUT:**

Array Index is Out Of Bounds

**ClassNotFoundException:**

This Exception is raised when we try to access a class whose definition is not found

**package** classtask;

**public** **class** ClassNotFoundException\_Demo {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try**{

Class.*forName*("Class1"); // Class1 is not defined

}

**catch**(ClassNotFoundException e){

System.***out***.println(e);

System.***out***.println("Class Not Found...");

}

}

}

**OUTPUT:**

java.lang.ClassNotFoundException: Class1

Class Not Found..

**FileNotFoundException:**

This Exception is raised when a file is not accessible or does not open.

**package** classtask;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**public** **class** File\_notFound\_Demo {

**public** **static** **void** main(String[] args) {

**try** {

// Following file does not exist

File file = **new** File("E://file.txt");

FileReader fr = **new** FileReader(file);

} **catch** (FileNotFoundException e) {

System.***out***.println("File does not exist");

}

}

}

**OUTPUT:**

File does not exist

**IOException:**

It is thrown when an input-output operation failed or interrupted

**package** classtask;

**import** java.util.Scanner;

**public** **class** IOException\_Demo {

**public** **static** **void** main(String[] args) {

// Create a new scanner with the specified String

// Object

Scanner scan = **new** Scanner("Hello Geek!");

// Print the line

System.***out***.println("" + scan.nextLine());

// Check if there is an IO exception

System.***out***.println("Exception Output: " + scan.ioException());

}

}

**OUTPUT:**

Hello Geek!

Exception Output: null

|  |  |  |
| --- | --- | --- |
| S.No | Error | Exception |
| 1 | The error indicates trouble that primarily occurs due to the scarcity of system resources. | The exceptions are the issues that can appear at runtime and compile time. |
| 2 | It is not possible to recover from an error. | It is possible to recover from an exception. |
| 3 | In java, all the errors are unchecked. | In java, the exceptions can be both checked and unchecked. |
| 4 | The system in which the program is running is responsible for errors. | The code of the program is accountable for exceptions. |
| 5 | They are described in the java.lang.Error package. | They are described in java.lang.Exception package |

1. **Difference between Checked and unchecked Exception**

In Java programming, an exception is an unplanned event that disrupts the normal flow of the program during execution.

These exceptions are generally of two types:

* Checked exceptions
* Unchecked exceptions

**Checked Exception:**

A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation; the programmer should take care of (handle) these exceptions. A checked exception is a type of exception that must be declared within the method where it is thrown.

* They occur at compile time.
* The compiler checks for a checked exception.
* These exceptions can be handled at the compilation time.
* It is a sub-class of the exception class.
* The JVM requires that the exception be caught and handled.
* Example of Checked exception- ‘File Not Found Exception’

**package** classtask;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**public** **class** File\_notFound\_Demo {

**public** **static** **void** main(String[] args) {

File file = **new** File("E://file.txt");

FileReader fr = **new** FileReader(file);

}

}

**OUTPUT:**

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Unhandled exception type FileNotFoundException

at classtask.File\_notFound\_Demo.main(File\_notFound\_Demo.java:12)

Eg:2

**package** classtask;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**public** **class** File\_notFound\_Demo {

**public** **static** **void** main(String[] args) {

**try** {

// Following file does not exist

// Reading file from path in local directory

File file = **new** File("E://file.txt");

// Creating object as one of ways of taking input

FileReader fr = **new** FileReader(file);

} **catch** (FileNotFoundException e) {

System.***out***.println("File does not exist");

}

}

}

OUTPUT:

File does not exist

**Unchecked Exception:**

An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation. An unchecked exception is an exception that occurs during the execution or runtime of a program.

* These exceptions occur at runtime.
* The compiler doesn’t check for these kinds of exceptions.
* These kinds of exceptions can’t be caught or handled during compilation time.
* This is because the exceptions are generated due to the mistakes in the program.
* These are not a part of the ‘Exception’ class since they are runtime exceptions.
* The JVM doesn’t require the exception to be caught and handled.
* Example of Unchecked Exceptions- ‘No Such Element Exception’

**package** classtask;

**public** **class** Unchecked\_Demo {

**public** **static** **void** main(String[] args) {

**int** num[] = {1, 2, 3, 4};

System.***out***.println(num[5]);

}

}

**OUTPUT:**

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 4

at classtask.Unchecked\_Demo.main(Unchecked\_Demo.java:7)

1. **write a program in java that takes two integers as input from the user and divides them. Handle the ArithmeticExpection in case of division by zero.**

**import** java.util.Scanner;

// Java program to demonstrate ArithmeticException

**public** **class** ArithmeticException\_Demo {

**public** **static** **void** main(String[] args) {

Scanner sc=**new** Scanner(System.***in***);

**int** a=sc.nextInt();

**int** b=sc.nextInt();

**try** {

// performing divison and storing th result

**int** c = a/b; // cannot divide by zero

System.***out***.println("Division process has been done successfully.");

System.***out***.println ("Result = " + c);

}

// handling the exception in the catch block

**catch**(ArithmeticException e) {

System.***out***.println ("Can't divide a number by 0");

}

}

}

**OUTPUT:**

5

0

Can't divide a number by 0

1. **ArrayOutofBoundException and StringOutofBoundException**

**ArrayOutofBoundException:**

**package** classtask;

**public** **class** ArrayOutofBoundDem {

**public** **static** **void** main(String[] args) {

// declaring and initializing an array of length 4

**int**[] x = { 1, 2, 3, 4 };

// accessing the element at 0 index

System.***out***.println(x[0]);

// accessing an index which is greater than the

// length of array

System.***out***.println(x[10]);

// accessing a negative index

System.***out***.println(x[-1]);

}

}

**OUTPUT:**

1

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 4

at classtask.ArrayOutofBoundDem.main(ArrayOutofBoundDem.java:14)

**StringOutofBoundException:**

**package** classtask;

**import** java.util.Scanner;

**public** **class** StringIndexOutofBoundDemo {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

String Line=sc.nextLine();

**try** {

**char** ch = Line.charAt(20); // Trying to access index out of bounds

System.***out***.println("Character at index 20: " + ch);

} **catch** (StringIndexOutOfBoundsException e) {

System.***out***.println("Caught StringIndexOutOfBoundsException: Index is out of bounds.");

}

}

}

OUTPUT:

Automation

Caught StringIndexOutOfBoundsException: Index is out of bounds.