Chapter – 7

- **Packages**
- **Wrapper Classes**
- **Exception Handling**

Packages in Java

-> In small projects, all the java files have unique names. So, it is not difficult to put them in a single folder.

-> But, in the case of huge projects where the number of java files is large, it is very difficult to put files in a single folder because the manner of storing files would be disorganized.

-> Moreover, if different java files in various modules of the project have the same name, it is not possible to store two java files with the same name in the same folder because it may occur naming conflict.

This problem of naming conflict can be overcome by using the concept of packages.

A package is nothing but group of related classes, interfaces, and sub-packages according to their functionality.

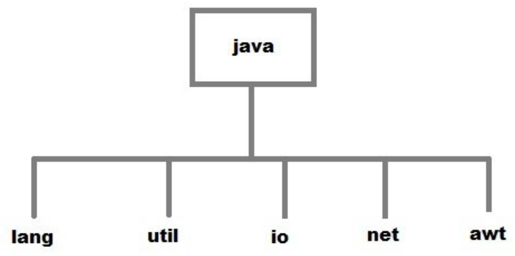
Types of Packages in Java

There are two different types of packages in Java. They are:

1. Predefined Packages in Java (Built-in Packages)

2. User-defined Package

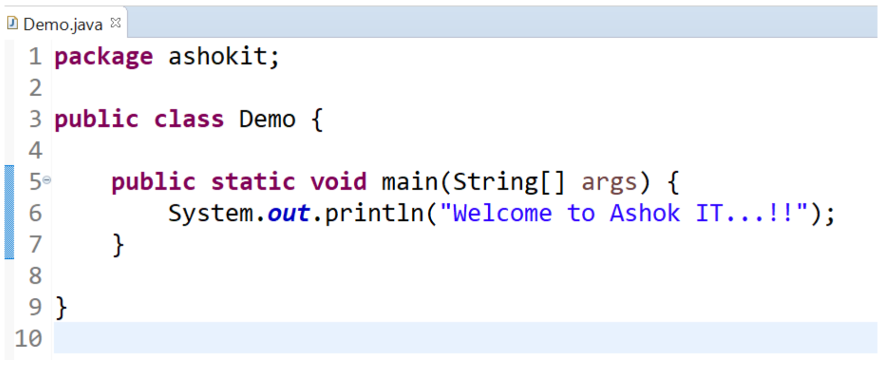
-> Built-in packages are existing java packages that come along with the JDK. For example, java.lang, java.util, java.io, etc.



The package which is defined by the user is called a User-defined package. It is used to create user-defined classes and interfaces.

Creating package in Java

Java supports a keyword called “package” which is used to create user-defined packages in java programming.



Demo.java

```java
package ashokit;

public class Demo {

    public static void main(String[] args) {
        System.out.println("Welcome to Ashok IT...!!");
    }

}
```

How to compile Java programs inside packages?

This is just like compiling a normal java program. If you are not using any IDE, you need to follow the steps given below to successfully compile your packages:

- javac -d . Demo.java

How to run Java package program?

To run the compiled class that we compiled using above command, we need to specify package name too. Use the below command to run the class file.

- java ashokit.FirstProgram

How to import Java Package

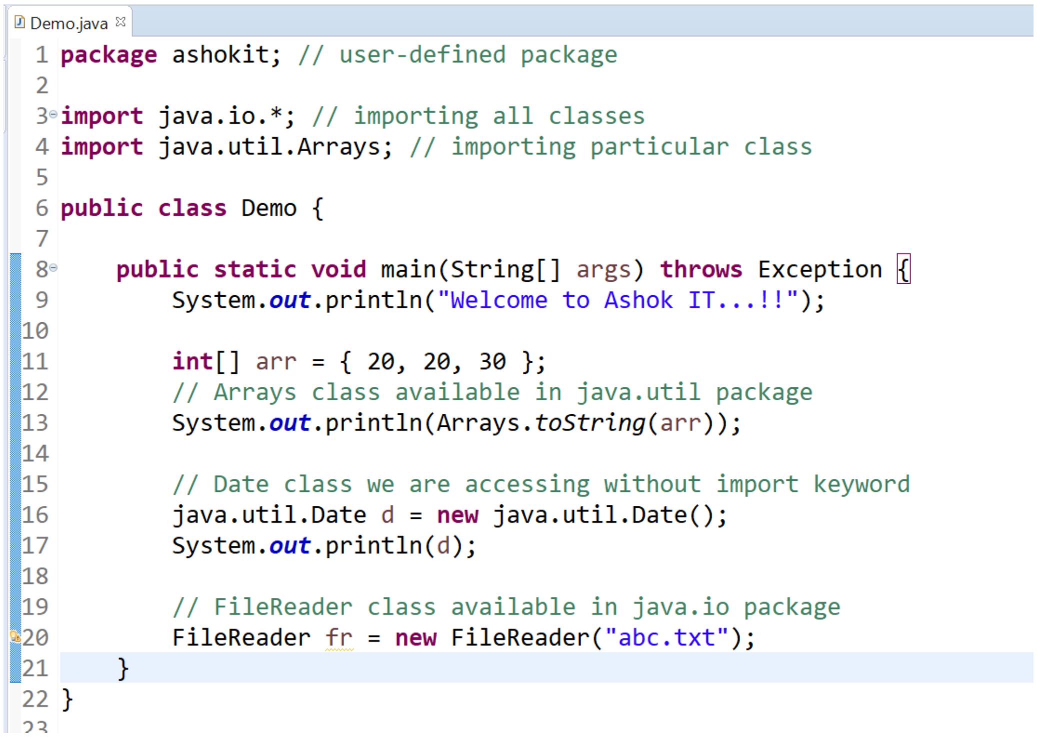
To import java package into a class, we need to use java import keyword which is used to access package and its classes into the java program.

Use import to access built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name.

There are 3 different ways to refer to any class that is present in a different package:

- without import the package
- import package with specified class
- import package with all classes

// Example on Packages importing

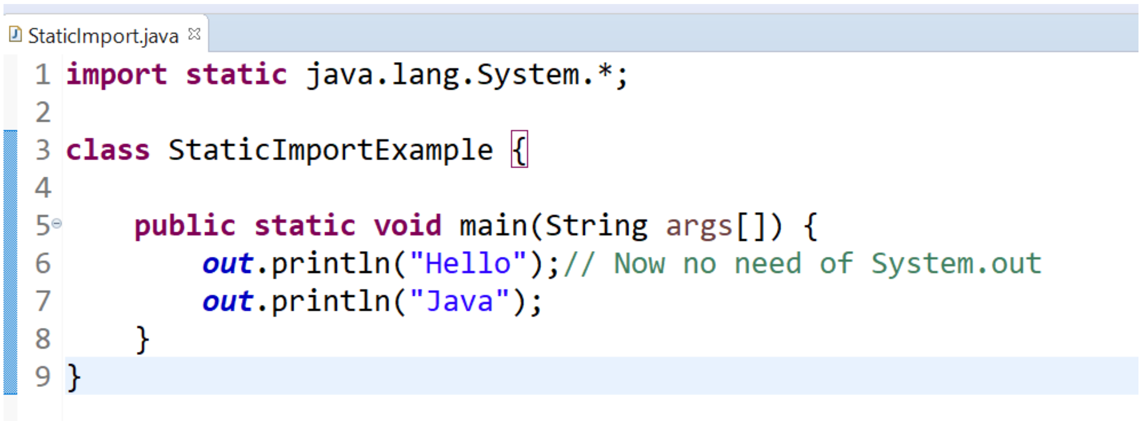


Demo.java

```java
package ashokit; // user-defined package

import java.io.*; // importing all classes
import java.util.Arrays; // importing particular class

public class Demo {

    public static void main(String[] args) throws Exception {
        System.out.println("Welcome to Ashok IT...!!");

        int[] arr = { 20, 20, 30 };
        // Arrays class available in java.util package
        System.out.println(Arrays.toString(arr));

        // Date class we are accessing without import keyword
        java.util.Date d = new java.util.Date();
        System.out.println(d);

        // FileReader class available in java.io package
        FileReader fr = new FileReader("abc.txt");
    }
}
```

Static imports

static imports are special kind of import statements given in java 1.5 version which are used to import the static members of any class into the program so that we can access those static members directly without any class name or object.



StaticImport.java

```java
import static java.lang.System.*;

class StaticImportExample {

    public static void main(String args[]) {
        out.println("Hello");// Now no need of System.out
        out.println("Java");
    }
}
```

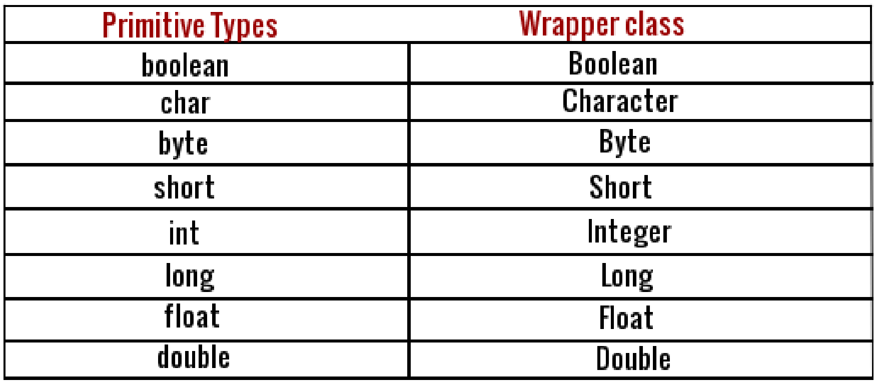
Java Wrapper Classes

-> In Java, Wrapper Class is used for converting primitive data type into object and object into a primitive data type.

-> For each primitive data type, a pre-defined class is present which is known as Wrapper class.

-> From J2SE 5.0 version the feature of autoboxing and unboxing is used for converting primitive data type into object and object into a primitive data type automatically.

-> All wrapper classes available in java.lang package



Primitive Types	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

8 Wrapper Classes in Java

Constructors of Wrapper classes

-> to create an object for any Wrapper class we have to use constructors of Wrapper classes.

-> In all most all wrapper classes we have following 2 constructors

-> first one will take primitive value directly

-> and second one will take primitive value in the form of String

Byte b = new Byte(byte);

Byte b = new Byte(String);

Short s = new Short(short);

Short s = new Short(String);

Integer i = new Integer(int);

Integer i = new Integer(String);

Long l = new Long(long);

Long l = new Long(String);

Float f = new Float(float);

Float f = new Float(double);

Float f = new Float(String);

Double d = new Double(double);

Double d = new Double(String);

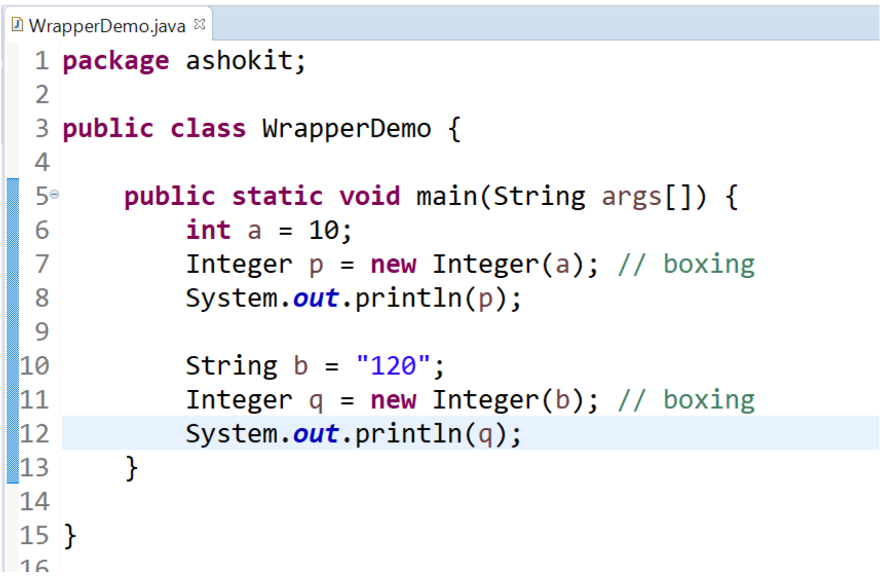
Character c = new Character(char);

Character c = new Character(String); // Not available

Boolean b = new Boolean(boolean);

Boolean b = new Boolean(String);

// example on Wrapper classes



```java
package ashokit;

public class WrapperDemo {

    public static void main(String args[]) {
        int a = 10;
        Integer p = new Integer(a); // boxing
        System.out.println(p);

        String b = "120";
        Integer q = new Integer(b); // boxing
        System.out.println(q);
    }

}
```

-> While creating an object for Boolean Wrapper class if constructor takes primitive boolean value as true it stores true otherwise if it takes primitive boolean value as false it stores false.

-> While creating an object for Boolean Wrapper class if constructor takes primitive boolean value in the form of string then if string value is true in any case (true, True,TRUE,...) then it stores true otherwise if string value is other than true it stores false.

Eg:

```
Boolean b1 = new Boolean(); //CTE

Boolean b2 = new Boolean(true); //true

Boolean b3 = new Boolean(false); //false

Boolean b4 = new Boolean(True); //CTE

Boolean b5 = new Boolean("TRUE"); //true

Boolean b6 = new Boolean("false"); //false

Boolean b7 = new Boolean("suresh"); //false

Boolean b8 = new Boolean(null); //false

Boolean b9 = new Boolean(1); //CTE
```

Methods of Wrapper classes

1. valueOf() : : This methd is also used to convert the primitive value into object(boxing)

syntax1: public static WrapperClass valueOf(primitive)

-> This method is available in all the 8 wrapper classes.

syntax2: public static WrapperClass valueOf(String)

-> This method is available in all the wrapper classes except Character class.

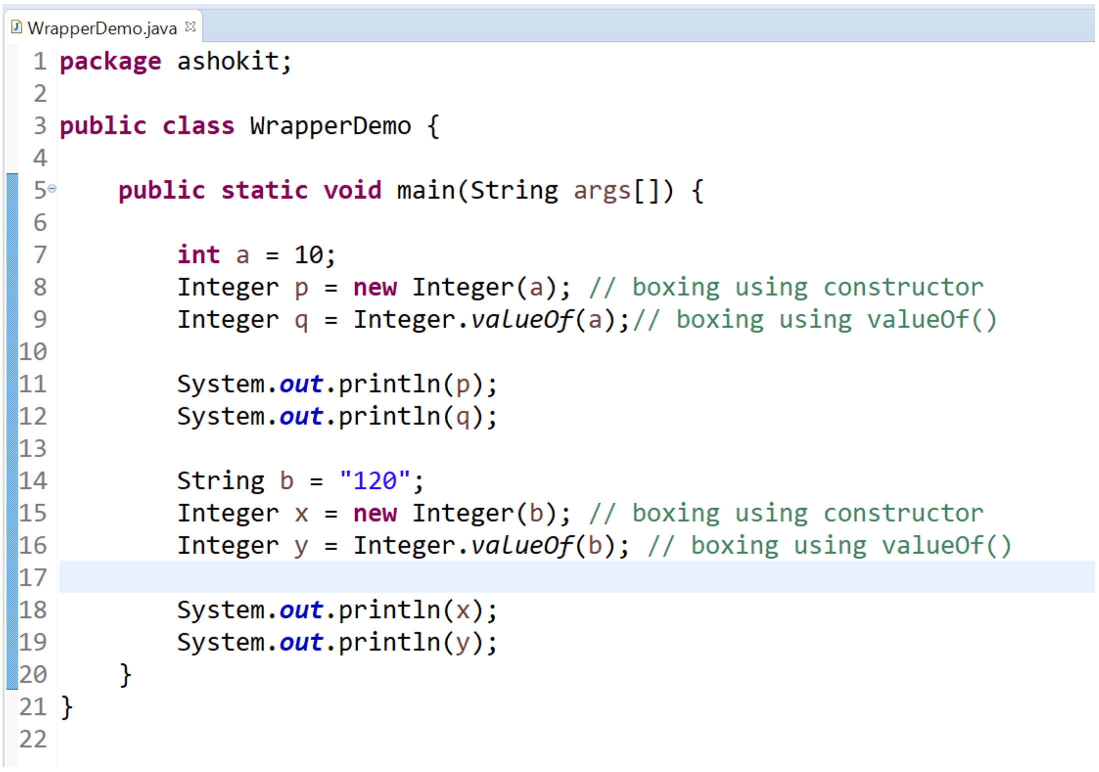
2. primitivetype xxxValue() :: This method return the primitive value available in the respective object, it means we are converting object into primitive value this procedure is called as unboxing.

3. static primitiveType parseXXX(String) : This method converts the primitive value available in the form of String into required primitive type.

-> This procedure is called as parsing.(kind of unboxing)

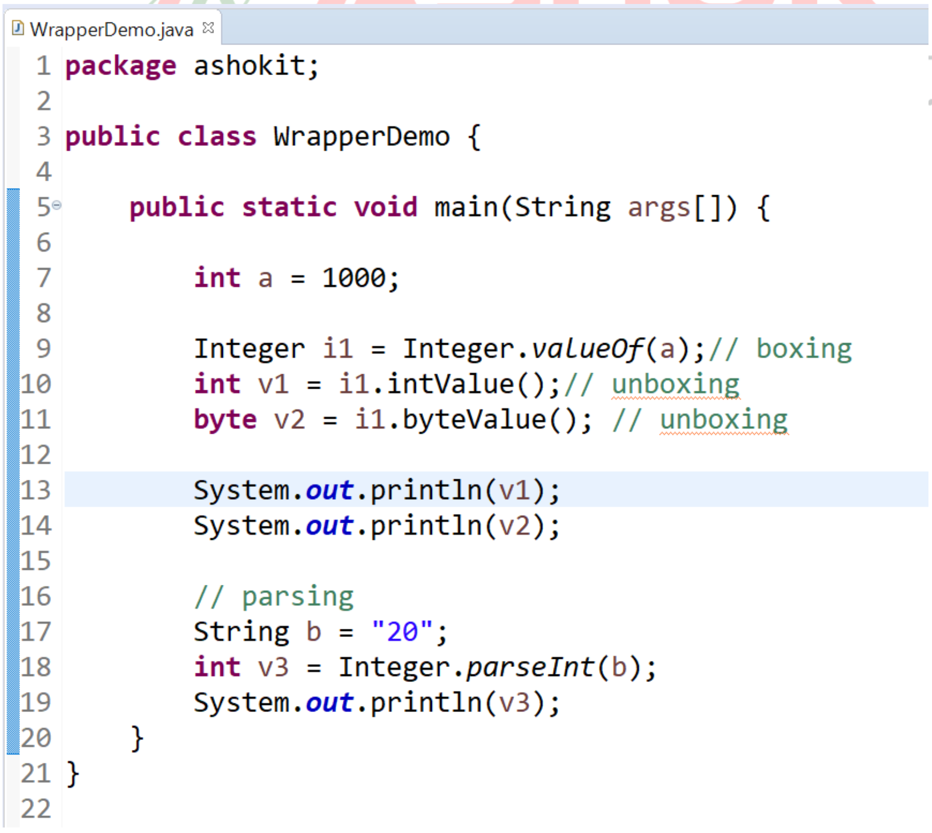
-> This method is available in all wrapper classes except Character class.

// Example on valueOf ( ) method

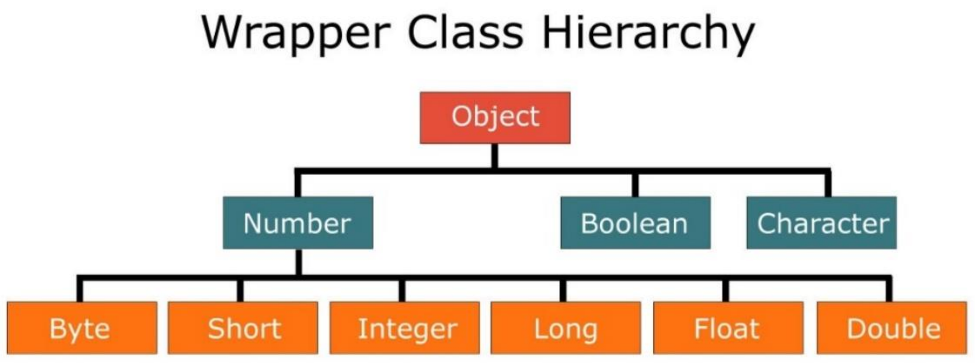

WrapperDemo.java

```
package ashokit;

public class WrapperDemo {

    public static void main(String args[]) {

        int a = 10;
        Integer p = new Integer(a); // boxing using constructor
        Integer q = Integer.valueOf(a);// boxing using valueOf()

        System.out.println(p);
        System.out.println(q);

        String b = "120";
        Integer x = new Integer(b); // boxing using constructor
        Integer y = Integer.valueOf(b); // boxing using valueOf()

        System.out.println(x);
        System.out.println(y);
    }
}
```

// example on static valueOf ( ) method


WrapperDemo.java

```
package ashokit;

public class WrapperDemo {

    public static void main(String args[]) {

        int a = 1000;

        Integer i1 = Integer.valueOf(a);// boxing
        int v1 = i1.intValue();// unboxing
        byte v2 = i1.byteValue(); // unboxing

        System.out.println(v1);
        System.out.println(v2);

        // parsing
        String b = "20";
        int v3 = Integer.parseInt(b);
        System.out.println(v3);
    }
}
```

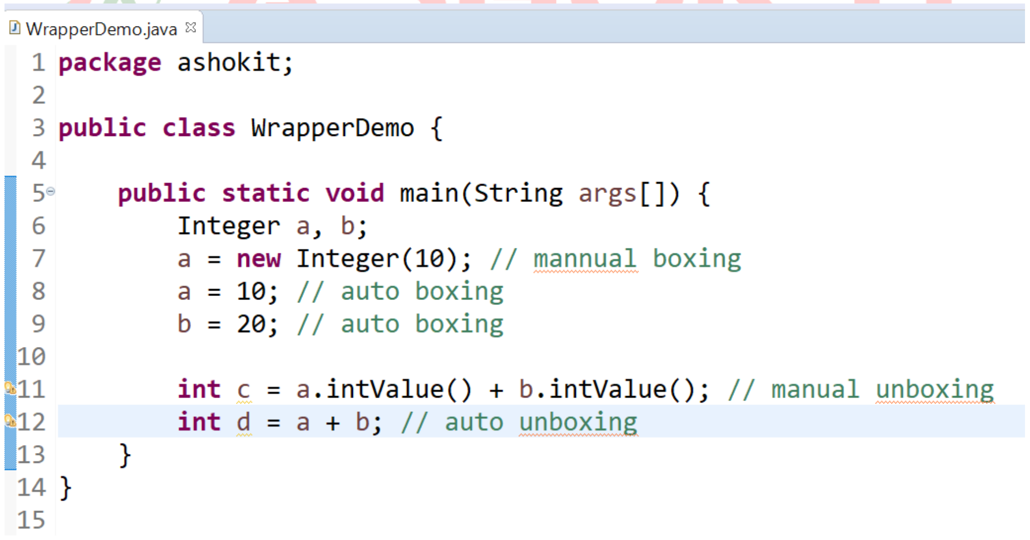


Typecasting: typecasting is a process of converting one primitive type to another primitive type or one reference to another reference type

Parsing: parsing is a process of converting String into corresponding primitive type

Boxing: converting a primitive value into object is called boxing. from java 1.5 onwards this procedure is done automatically by compiler hence it is called auto boxing.

Unboxing: converting an object value into primitive type is called un boxing. from java 1.5 onwards this procedure is automatically done by compiler hence it is called auto un boxing



```
package ashokit;

public class WrapperDemo {

    public static void main(String args[]) {
        Integer a, b;
        a = new Integer(10); // mannual boxing
        a = 10; // auto boxing
        b = 20; // auto boxing

        int c = a.intValue() + b.intValue(); // manual unboxing
        int d = a + b; // auto unboxing
    }
}
```

Exception Handling

Whenever we develop any application there is a chance of getting some errors in the application. When exception occurs in the program then our program will be terminated abnormally.

Exception handling is a mechanism to catch and throw Java exceptions so that the execution of the program will not get disturbed.

What is an Exception?

Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions.

Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

An exception can occur for many reasons. Some of them are:

- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file
- resource exhaustion

What is Error?

Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

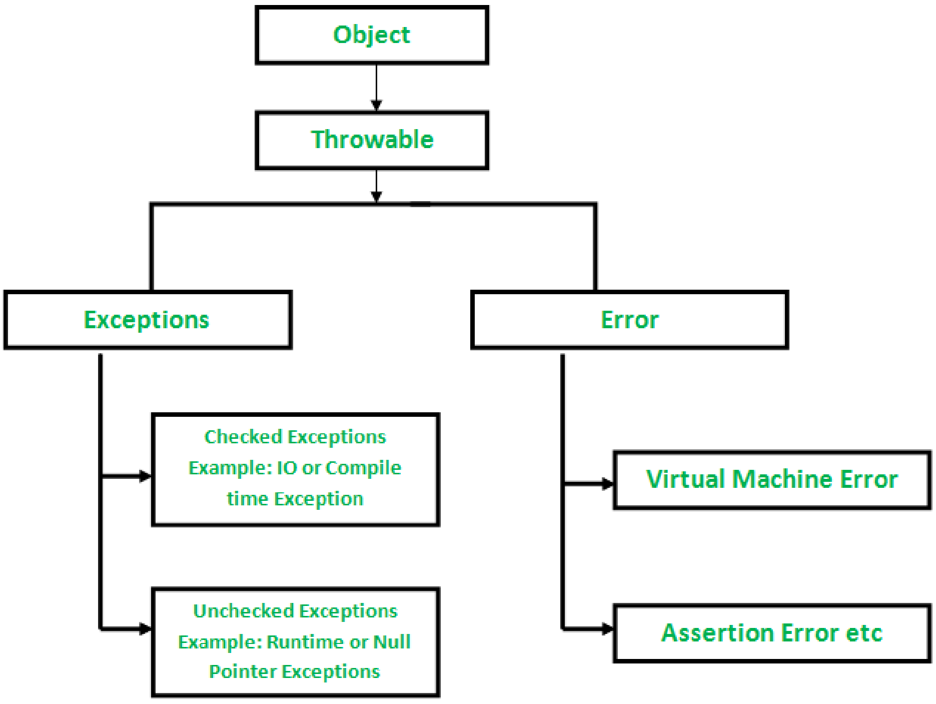
What is the difference between Exception and Error?

Error: An Error indicates a serious problem that a reasonable application should not try to catch.

Exception: Exception indicates conditions that a reasonable application might try to catch.

Exception Hierarchy

All exception types are subclasses of class Throwable, which is at the top of exception class hierarchy.


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Checked Exceptions

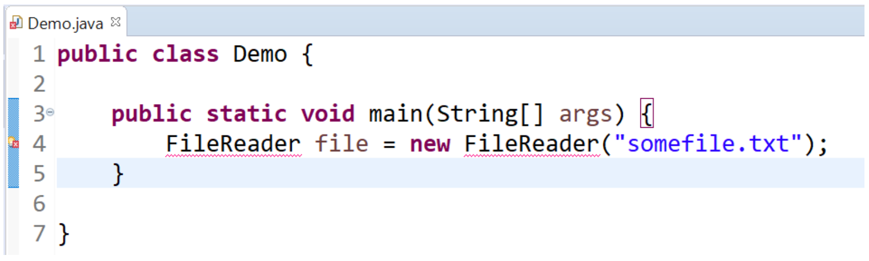
These are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the throws keyword.

Note: Checked exceptions are checked by the Java compiler so they are called compile time exceptions.

Note that all checked exceptions are subclasses of Exception class. For example,

- ClassNotFoundException
- IOException
- SQLException etc..

// Example on Checked Exception

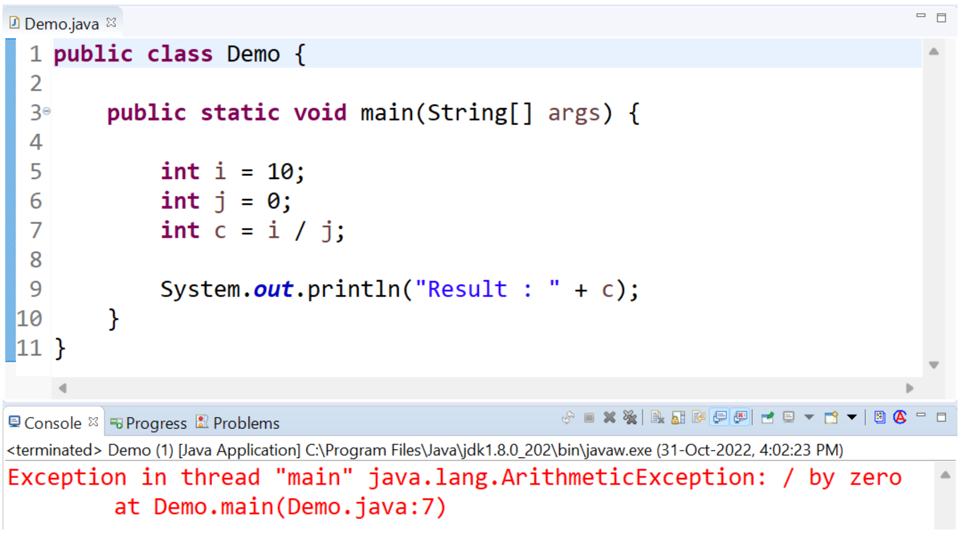


```
public class Demo {

    public static void main(String[] args) {
        FileReader file = new FileReader("somefile.txt");
    }

}
```

Un Checked Exceptions

Unchecked exceptions are not checked by the compiler. When the buggy code is executed then we will get exception at runtime, these are called runtime exceptions.

//example on Unchecked exception



Demo.java

```java
public class Demo {

    public static void main(String[] args) {

        int i = 10;
        int j = 0;
        int c = i / j;

        System.out.println("Result : " + c);
    }
}
```

Console

```
<terminated> Demo (1) [Java Application] C:\Program Files\Java\jdk1.8.0_202\bin\javaw.exe (31-Oct-2022, 4:02:23 PM)
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Demo.main(Demo.java:7)
```

Exception Handling Keywords

In java, we can handle exceptions using below 5 keywords

1. try
2. catch
3. finally
4. throw
5. throws

try: The try block is used to enclose the suspected code. Suspected code is a code that may raise an exception during program execution.

For example, if a code raise arithmetic exception due to divide by zero then we can wrap that code into the try block.

Syntax :

```
try {
  //group of statements which may generate the exception
}
```

Example:

```
try {
 int a = 10;
 int b = 0
 int c = a/b; // exception
}
```

catch : The catch block also known as handler is used to handle the exception. It handles the exception thrown by the code enclosed into the try block. Try block must provide a catch handler or a finally block.

The catch block must be used after the try block only. We can also use multiple catch block with a single try block.

Syntax :

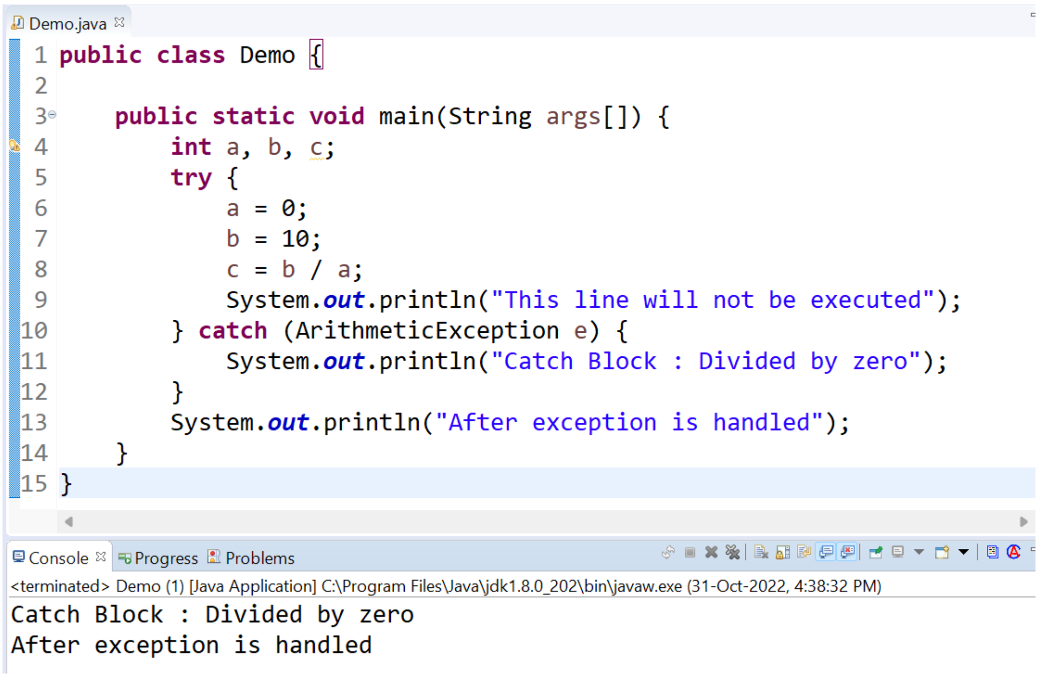
```
catch (AnyException ae) {
   // group of exception handling statements
}
```

Example:

```
try {
 int a = 10;
 int b = 0 ;
 int c = a / b;   // exception
} catch(ArithmeticException e){
 System.out.println(e);
}
```

Note: Exception handling is done by transferring the execution of a program to an appropriate exception handler (catch block) when exception occurs.

// write a java program on try-catch blocks



Demo.java

```java
public class Demo {

    public static void main(String args[]) {
        int a, b, c;
        try {
            a = 0;
            b = 10;
            c = b / a;
            System.out.println("This line will not be executed");
        } catch (ArithmeticException e) {
            System.out.println("Catch Block : Divided by zero");
        }
        System.out.println("After exception is handled");
    }
}
```

Console Progress Problems

```
<terminated> Demo (1) [Java Application] C:\Program Files\Java\jdk1.8.0_202\bin\javaw.exe (31-Oct-2022, 4:38:32 PM)
Catch Block : Divided by zero
After exception is handled
```

Exception handling is done by transferring the execution of a program to an appropriate exception handler (catch block) when exception occurs.

An exception will throw by this program as we are trying to divide a number by zero inside try block. The program control is transferred outside try block. Thus, the line "This line will not be executed" is never parsed by the compiler. The exception thrown is handled in catch block. Once the exception is handled, the program control is continued with the next line in the program i.e., after catch block. Thus, the line "After exception is handled" is printed.

Java try with Resource Statement

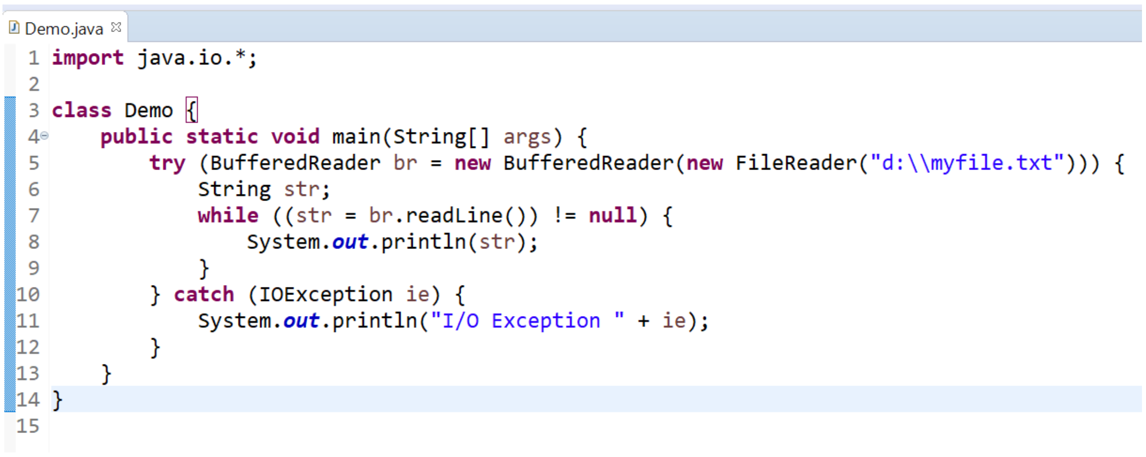
-> try with resource is a new feature of Java that was introduced in Java 7 and further improved in Java 9. This feature add another way to exception handling with resources management. It is also referred as automatic resource management. It close resources automatically by using AutoCloseable interface..

-> Resource can be any like: file, connection etc and we don't need to explicitly close these, JVM will do this automatically.

-> Suppose, we run a JDBC program to connect to the database then we have to create a connection and close it at the end of task as well. But in case of try-with-resource we don’t need to close the connection, JVM will do this automatically by using AutoCloseable interface.

```
try ( resource-specification )
{
   //use the resource
}
catch()
{
   // handler code
}
```

-> This try statement contains a parenthesis in which one or more resources is declared. Any object that implements java.lang.AutoCloseable or java.io.Closeable, can be passed as a parameter to try statement. A resource is an object that is used in program and must be closed after the program is finished. The try-with-resources statement ensures that each resource is closed at the end of the statement of the try block. We do not have to explicitly close the resources.



Demo.java

```java
import java.io.*;

class Demo {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("d:\\myfile.txt"))) {
            String str;
            while ((str = br.readLine()) != null) {
                System.out.println(str);
            }
        } catch (IOException ie) {
            System.out.println("I/O Exception " + ie);
        }
    }
}
```

<u>Points to Remember</u>

-> A resource is an object in a program that must be closed after the program has finished.

-> Any object that implements java.lang.AutoCloseable or java.io.Closeable can be passed as a parameter to try statement.

-> All the resources declared in the try-with-resources statement will be closed automatically when the try block exits. There is no need to close it explicitly.

-> We can write more than one resources in the try statement.

-> In a try-with-resources statement, any catch or finally block is run after the resources declared have been closed.

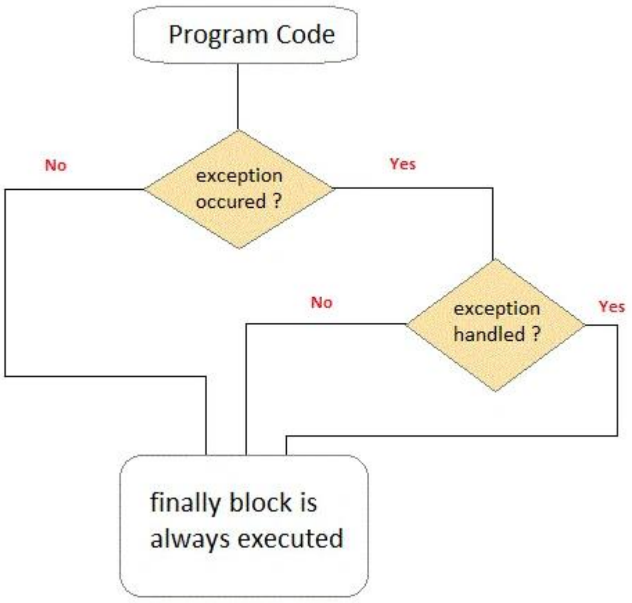
finally block :

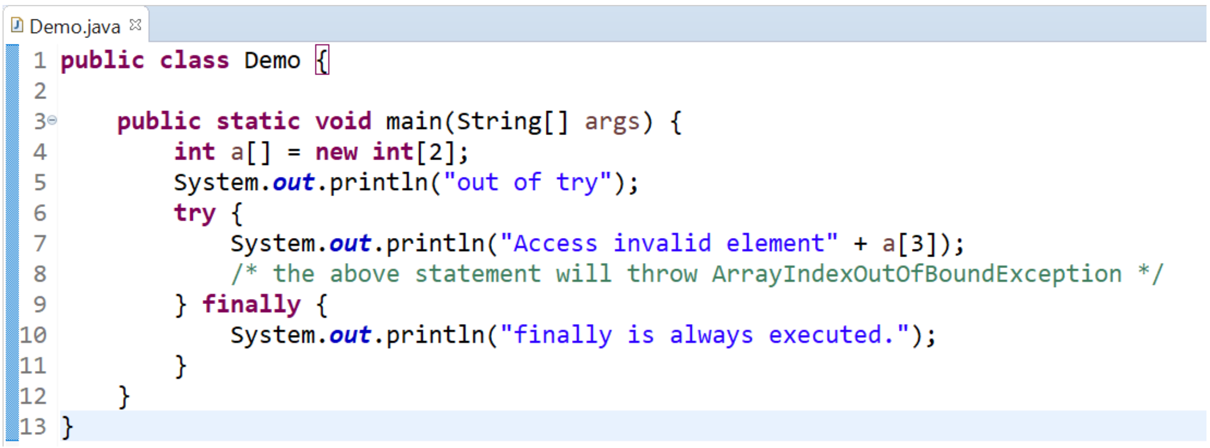
-> A finally keyword is used to create a block of code that follows a try block.

-> A finally block of code is always executed whether an exception has occurred or not.

-> Using a finally block, it lets you run any clean up type statements that you want to execute, no matter what happens in the protected code.

-> A finally block appears at the end of catch block.





```java
public class Demo {

    public static void main(String[] args) {
        int a[] = new int[2];
        System.out.println("out of try");
        try {
            System.out.println("Access invalid element" + a[3]);
            /* the above statement will throw ArrayIndexOutOfBoundException */
        } finally {
            System.out.println("finally is always executed.");
        }
    }
}
```

Rules for writing try-catch-finally

1.try must followed by either catch block or finally block or both

2.catch must be followed by either catch block or finally block

3.catch must be preceded by either catch block or try block

4.finally must preceded by either catch block or try block

5. we can write only one finally statement and one try block and many catch blocks in try-catch-finally chain.

6. nesting try-catch-finally is also possible

7. when we write multiple catch blocks if Exceptions are not having any Is-A relation then we can write catch block in any order otherwise we must write catch blocks in a order like first child class and followed by parent class.

8. we can not write 2 catch blocks which are going to catch same exceptions.

How JVM execute try-catch-finally

-> First it will execute the statements written before try-catch blocks next JVM Will enter into try block and if no exception occurred then it will execute all the statements of try block and it will skip all the catch block and control will be given after try-catch blocks

-> JVM Will enter into try block and if exception occurred then it will skip remaining statements of try block and control will be given any one of the matching catch blocks and executes the statements available in the catch block and next control will be given after try-catch blocks

Note:

1.when Exception occurred then control will be given any of the matching catch blocks and the remaining catch blocks will be skipped. and control will be given after try-catch blocks

2. If no catch block is matching then JVM will call or invoke " DefaultExceptionHandler" and which always cause for abnormal termination.

3. If Exception Occurred or not in both the cases finally block will be executed. `

Working with Multiple Catch Blocks

-> For single try block we can write multiple catch blocks

=> When we are writing multiple catch blocks, the hierarchy should be child to parent.

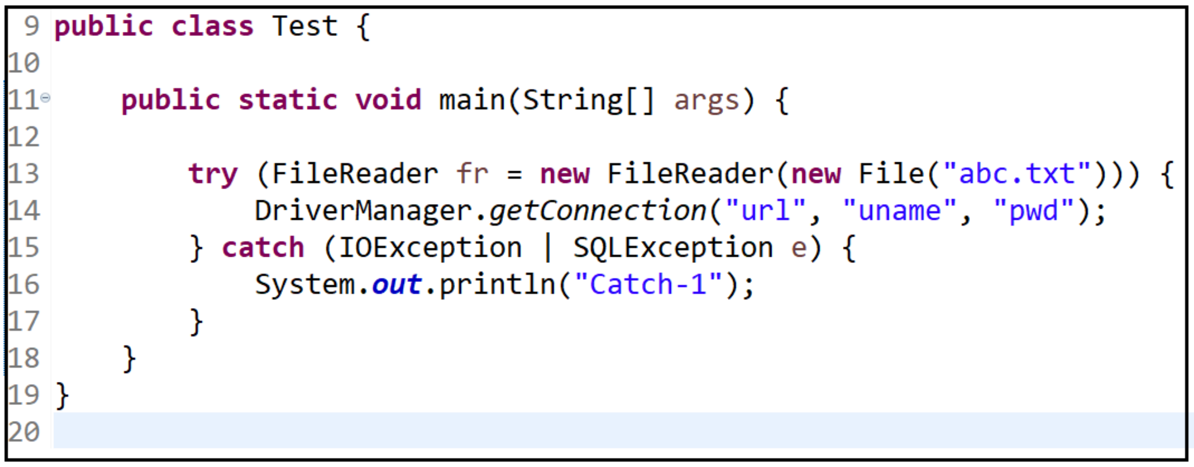
// invalid code because second catch block is un-reachable

```
try{
        int a = 10 / 0;
} cath (Exception e) {
        s.o.p("catch-1");
} catch (ArithematicException e) {
        s.o.p("catch-2");
}
```

// valid code

```
try{
        int a = 10 / 0;
} cath (ArithematicException e) {
        s.o.p("catch-1");
} catch (Exception e) {
        s.o.p("catch-2");
}
```

=> We can handle multiple Exceptions in Single Catch block also like below



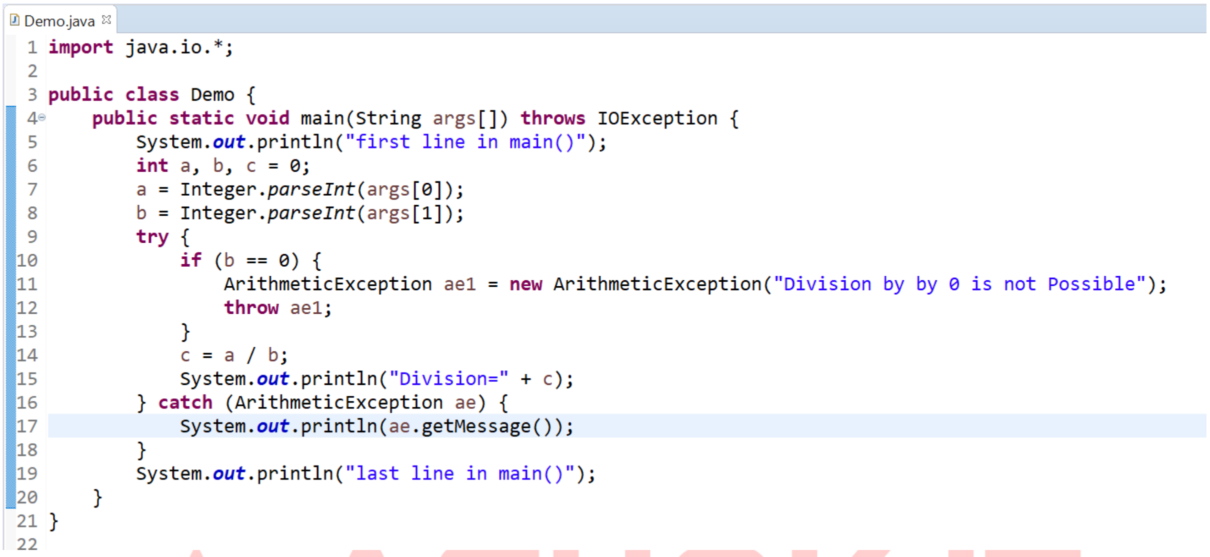
```
public class Test {

    public static void main(String[] args) {

        try (FileReader fr = new FileReader(new File("abc.txt"))) {
            DriverManager.getConnection("url", "uname", "pwd");
        } catch (IOException | SQLException e) {
            System.out.println("Catch-1");
        }
    }
}

```

Throw :

-> The throw keyword is used to throw an exception explicitly.

-> By default all predefined exceptions are created and thrown implicitly and identified by JVM

-> If we want to throw the exceptions explicitly then we have to use throw keyword.



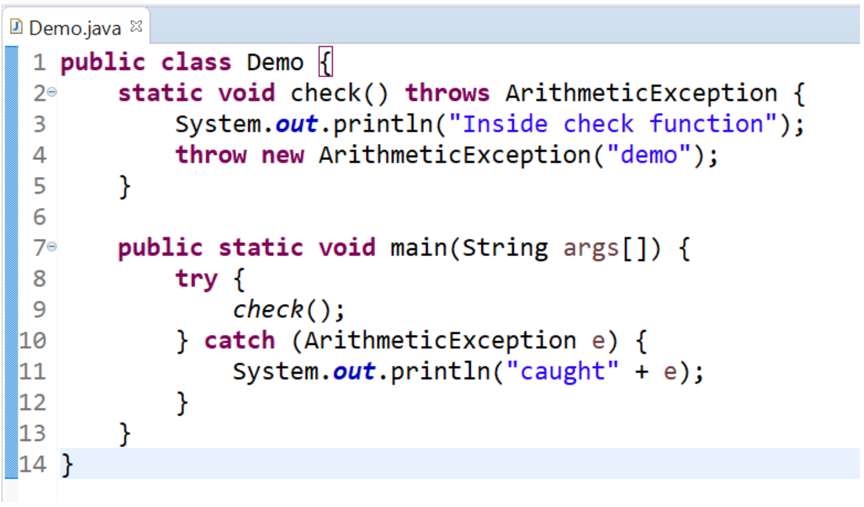
Demo.java

```java
import java.io.*;

public class Demo {
    public static void main(String args[]) throws IOException {
        System.out.println("first line in main()");
        int a, b, c = 0;
        a = Integer.parseInt(args[0]);
        b = Integer.parseInt(args[1]);
        try {
            if (b == 0) {
                ArithmeticException ae1 = new ArithmeticException("Division by by 0 is not Possible");
                throw ae1;
            }
            c = a / b;
            System.out.println("Division=" + c);
        } catch (ArithmeticException ae) {
            System.out.println(ae.getMessage());
        }
        System.out.println("last line in main()");
    }
}

```

Java throws Keyword:

The throws keyword is used to declare the list of exception that a method may throw during execution of program. Any method that is capable of causing exceptions must list all the exceptions possible during its execution, so that anyone calling that method gets a prior knowledge about which exceptions are to be handled. A method can do so by using the throws keyword.

```
type method_name(parameter_list) throws exception_list
{
 // definition of method
}
```


Demo.java

```java
public class Demo {
    static void check() throws ArithmeticException {
        System.out.println("Inside check function");
        throw new ArithmeticException("demo");
    }

    public static void main(String args[]) {
        try {
            check();
        } catch (ArithmeticException e) {
            System.out.println("caught" + e);
        }
    }
}
```

throw	throws
throw keyword is used to throw an exception explicitly.	throws keyword is used to declare an exception possible during its execution.
throw keyword is followed by an instance of Throwable class or one of its sub-classes.	throws keyword is followed by one or more Exception class names separated by commas.
throw keyword is declared inside a method body.	throws keyword is used with method signature (method declaration).
We cannot throw multiple exceptions using throw keyword.	We can declare multiple exceptions (separated by commas) using throws keyword.

<u>User defined Exceptions</u>

Java provides rich set of built-in exception classes like: ArithmeticException, IOException, NullPointerException etc. all are available in the java.lang package and used in exception handling. These exceptions are already set to trigger on pre-defined conditions such as when you divide a number by zero it triggers ArithmeticException.

Apart from these classes, Java allows us to create our own exception class to provide own exception implementation. These type of exceptions are called as user-defined exceptions or custom exceptions.

-> To create the user defined exceptions, we have to follow the following procedure

procedure:

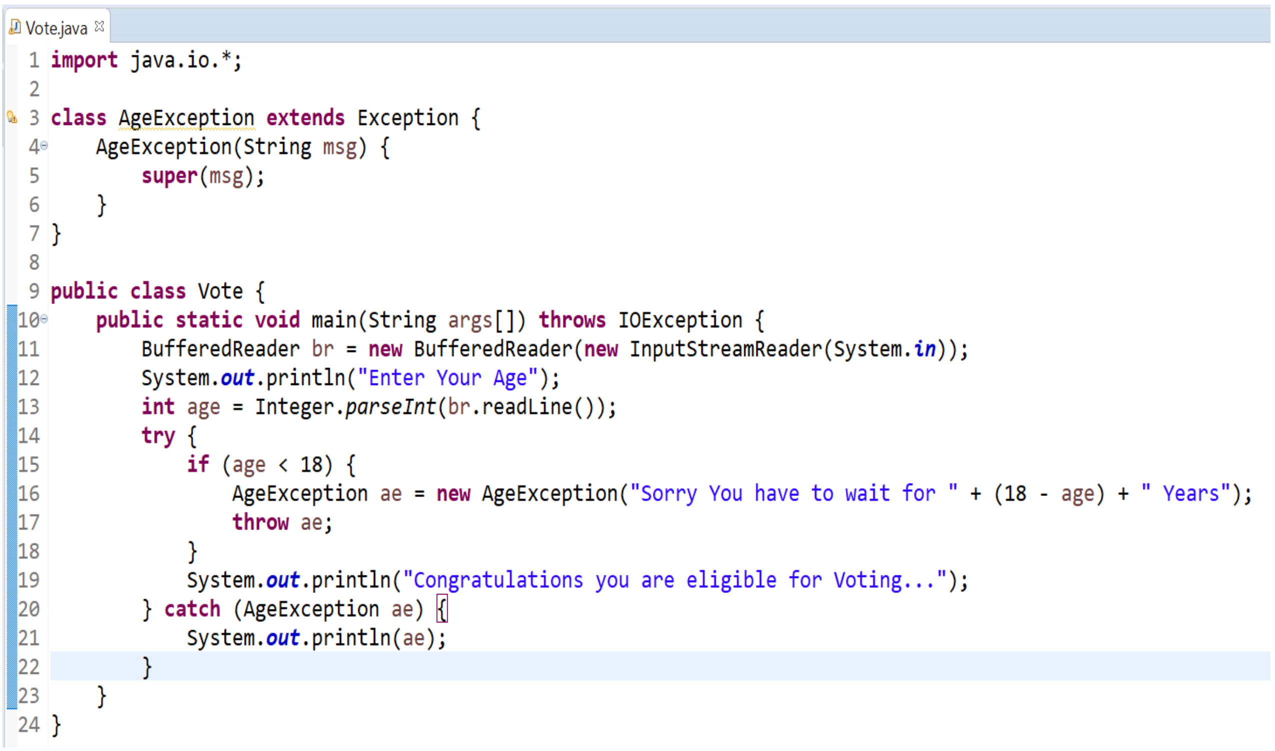
1. All exception classes are extending from Exception class. So that to create user defined exception we should create a class that extends any one of the following 2 classes

   a) Exception (Checked Exceptions)

   b) RuntimeException (Unchecked Exceptions)

2. Create a parameterized constructor which calls super class parameterized constructor to set the Exception Message.

3. JVM don't know when to generate the user defined exception and how to create and throw the object for User defined exception so that we should explicitly create the object for User defined exception and throw manually using throw keyword.



Vote.java

```java
import java.io.*;

class AgeException extends Exception {
    AgeException(String msg) {
        super(msg);
    }
}

public class Vote {
    public static void main(String args[]) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter Your Age");
        int age = Integer.parseInt(br.readLine());
        try {
            if (age < 18) {
                AgeException ae = new AgeException("Sorry You have to wait for " + (18 - age) + " Years");
                throw ae;
            }
            System.out.println("Congratulations you are eligible for Voting...");
        } catch (AgeException ae) {
            System.out.println(ae);
        }
    }
}
```

Knowledge – Check

1) What is package and why we need packages?
2) What is Import in java?
3) What is static import?
4) What is Wrapper class?
5) When we have primitive types why to go for Wrapper Classes?
6) What is Auto Boxing and Auto Unboxing?
7) What is Exception Hierarchy in Java?
8) Checked Exceptions Vs Un-Checked Exceptions?
9) How to handle Exceptions?
10) What is try-catch-finally?
11) What is the difference between throws and throw?
12) What is try with resources?
13) Why to write Multiple Catch blocks & do you any alternate?
14) How to create and throw User Defined Exception?
15) What is JVM's behaviour when Runtime Exception occurred?
16) What is Exception re-throw ?
17) Can you explain few checked exceptions?
18) Can you explain few Un-Checked exceptions?
19) Can you write a program which gives Stack Overflow Error?
20) What is the difference between final, finalize ( ) and finally block ?