AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER 6
EXCEPTION HANDLING,I/O,ABSTRACT CLASSES AND INTERFACES

Problem Occurs
Create Exception
Throw Exception
Handle Exception

SUBJECT:OOP-I CODE:3140705	PREPARED BY: ASST.PROF.NENSI KANSAGARA (CSE DEPARTMENT,ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

EXCEPTION HANDLING

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CONCEPT OF EXCEPTION

WHAT IS AN EXCEPTION?

An 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.

WHY AN EXCEPTION OCCURS?

There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

CONT..

EXCEPTION HANDLED BY 5 KEYWORDS:

1. TRY
2. CATCH
3. THROW
4. THROWS
5. FINALLY

EXAMPLE OF EXCEPTION:

```
class Main {

  public static void main(String[] args) {

    try {

      int divideByZero = 5 / 0;

      System.out.println("Rest of code in try block");

    } catch (ArithmeticException e) {

      System.out.println("ArithmeticException => " + e.getMessage());

    }  }

  }
```

EXCEPTION AND ERROR

Exceptions are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions

Errors indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.

There are two types of errors:

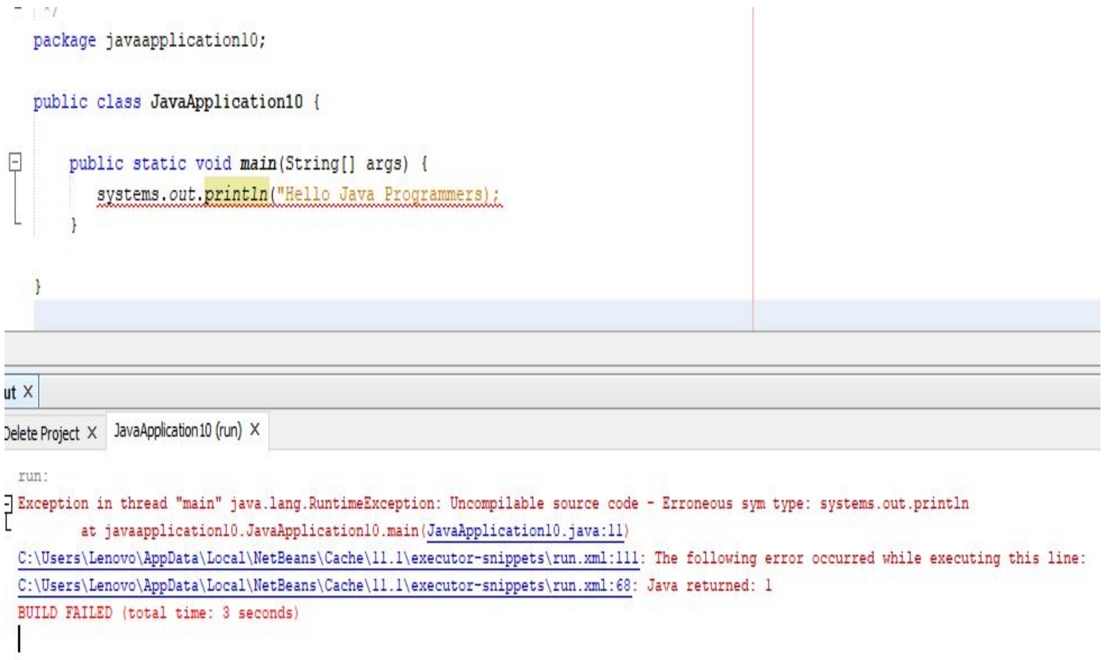
1.Compile time error

2.Run time error.

COMPILE TIME ERROR

The errors that are detected by the java compiler during the compile time are called the compile time errors.When compile time errors occurred it will not generate .class file.

1. Missing Semicolon
2. Wrong Spelling of keywords and identifiers
3. Missing brackets of class and methods
4. Missing double quotes in the string
5. Use of undeclared Variables.
6. Use of = instead of ==
7. Incompatible types in assignment statements
8. Illegal references to the object



```java
package javaapplication10;

public class JavaApplication10 {

    public static void main(String[] args) {
        systems.out.println("Hello Java Programmers);
    }

}
```

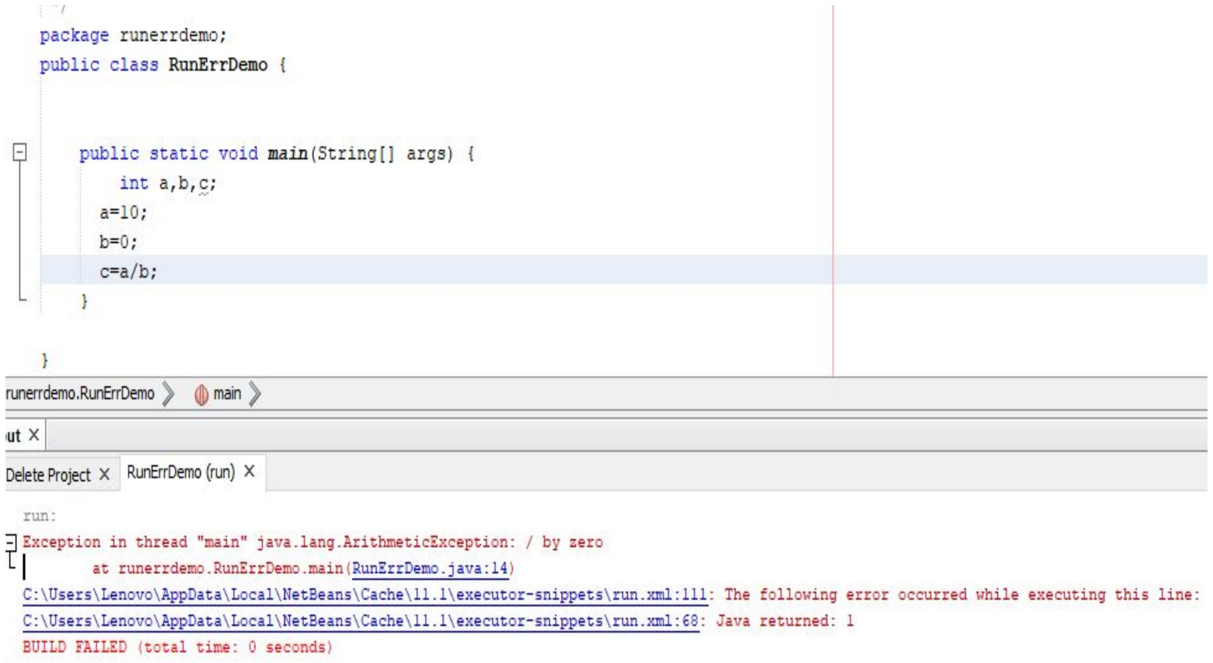
```
run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - Erroneous sym type: systems.out.println
	at javaapplication10.JavaApplication10.main(JavaApplication10.java:11)
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 3 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

RUN-TIME ERROR

- ➔ Sometimes the program is free from the compile time errors and creates a .class file
- ➔ It is basically the logic errors,that get caused due to the wrong logic.

1. Accessing the array element which is out of the index.
2. In an expression,divided by zero.
3. Trying to store a value into an array of incompatible type.
4. Passing the parameters that is not in valid in range
5. Converting invalid string to numbers
6. Trying to illegally change the state of thread.



```
package runerrdemo;
public class RunErrDemo {


    public static void main(String[] args) {
        int a,b,c;
      a=10;
      b=0;
      c=a/b;
    }


}
```

runerrdemo.RunErrDemo 〉 main 〉

ut ✕

Delete Project ✕ RunErrDemo (run) ✕

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at runerrdemo.RunErrDemo.main(RunErrDemo.java:14)
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

TRY-CATCH BLOCK:

Try block

Exception causing statement

Exception objects gets created over here

Catch block

Exception handling statement

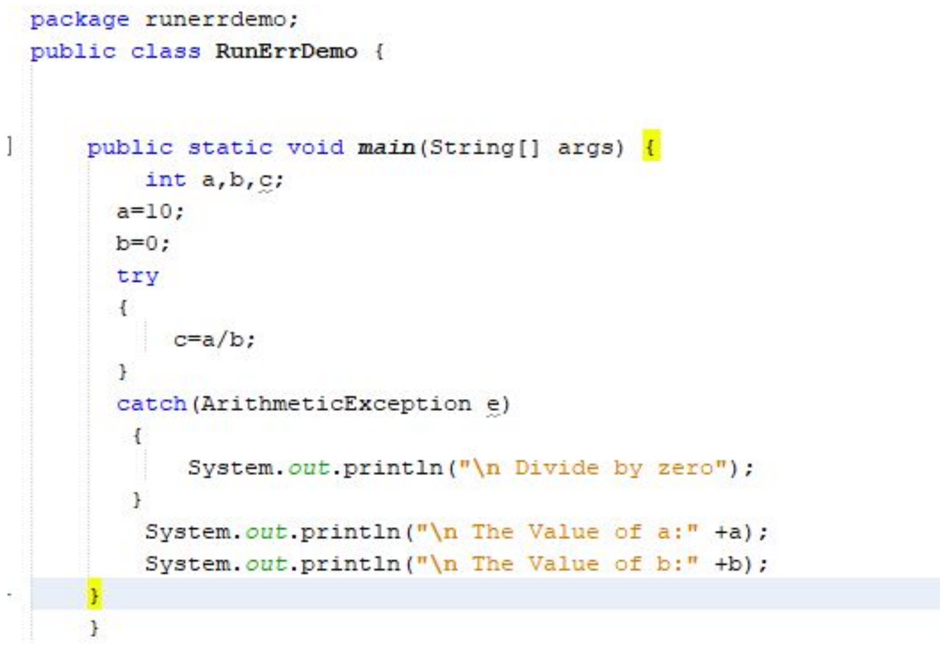
Exception Handler

SYNTAX:

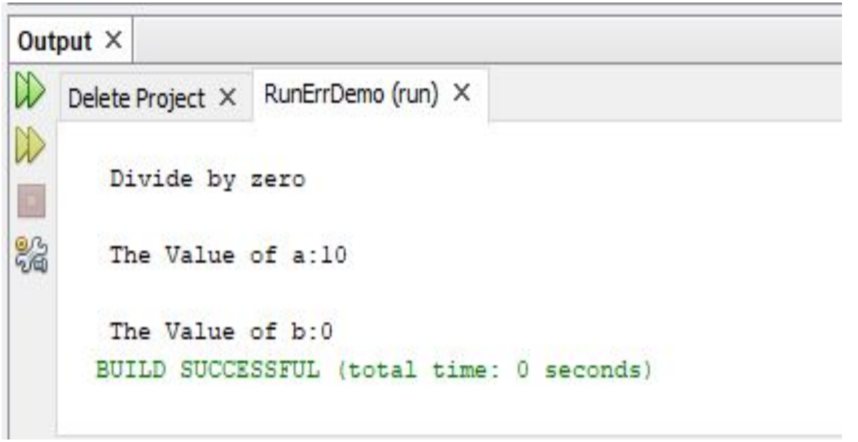
```
try

{

//Exception gets generated here

}

catch (ExceptionType name)

 {

//Exception is handled here

 }
```

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Example of Try-Catch Block:



```
package runerrdemo;
public class RunErrDemo {


    public static void main(String[] args) {
        int a,b,c;
      a=10;
      b=0;
      try
      {
          c=a/b;
      }
      catch(ArithmeticException e)
       {
           System.out.println("\n Divide by zero");
       }
        System.out.println("\n The Value of a:" +a);
        System.out.println("\n The Value of b:" +b);
    }
    }
```



```
 Divide by zero

 The Value of a:10

 The Value of b:0
BUILD SUCCESSFUL (total time: 0 seconds)
```

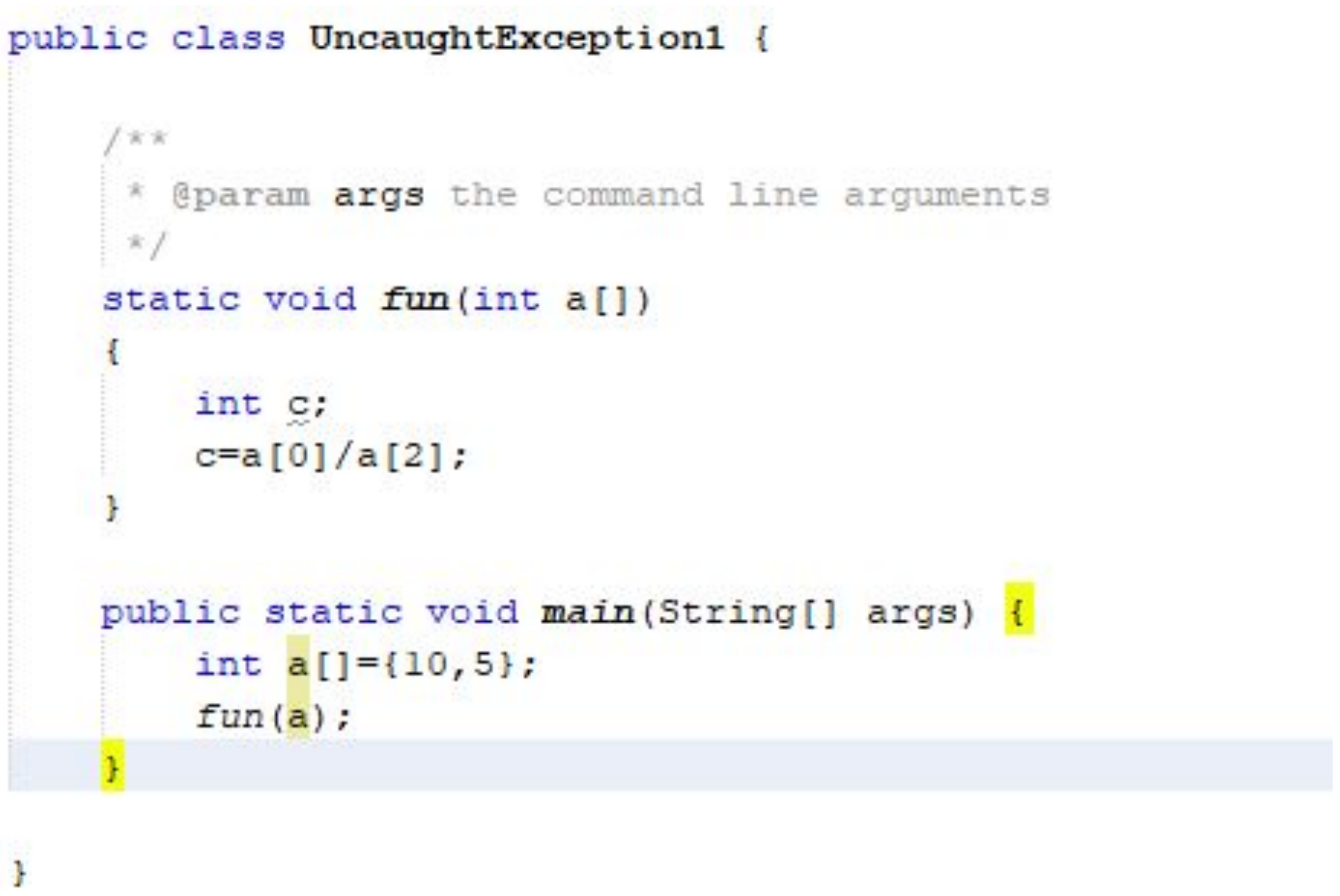
https://www.youtube.com/watch?list=PLHuIqV4DwPicArdMjI2YI5Hu5c2TnMgvV&time_continue=110&v=VIbZRUjh2d0&feature=emb_logo

Uncaught Exception:

If there exists some code in the source program which may cause an exception and if the programmer does not handle this exception than java run time system raised the exception.

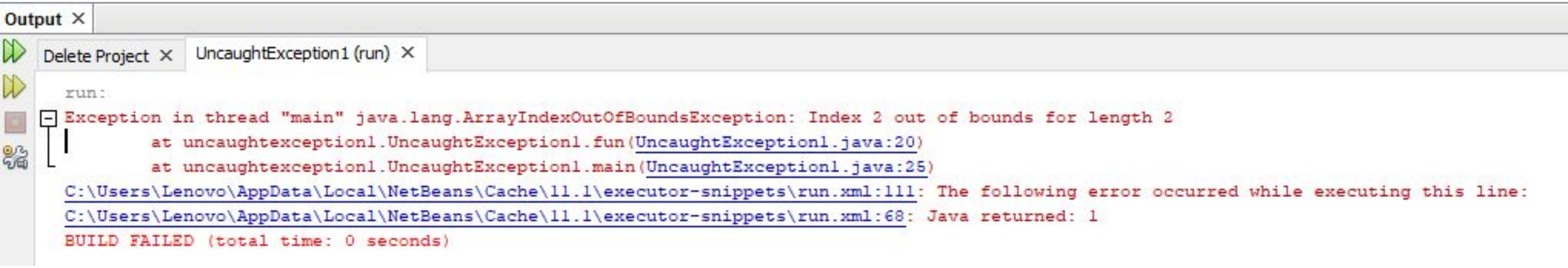
Following example shows uncaught exception

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```
public class UncaughtException1 {

    /**
     * @param args the command line arguments
     */
    static void fun(int a[])
    {
        int c;
        c=a[0]/a[2];
    }

    public static void main(String[] args) {
        int a[]={10,5};
        fun(a);
    }

}
```

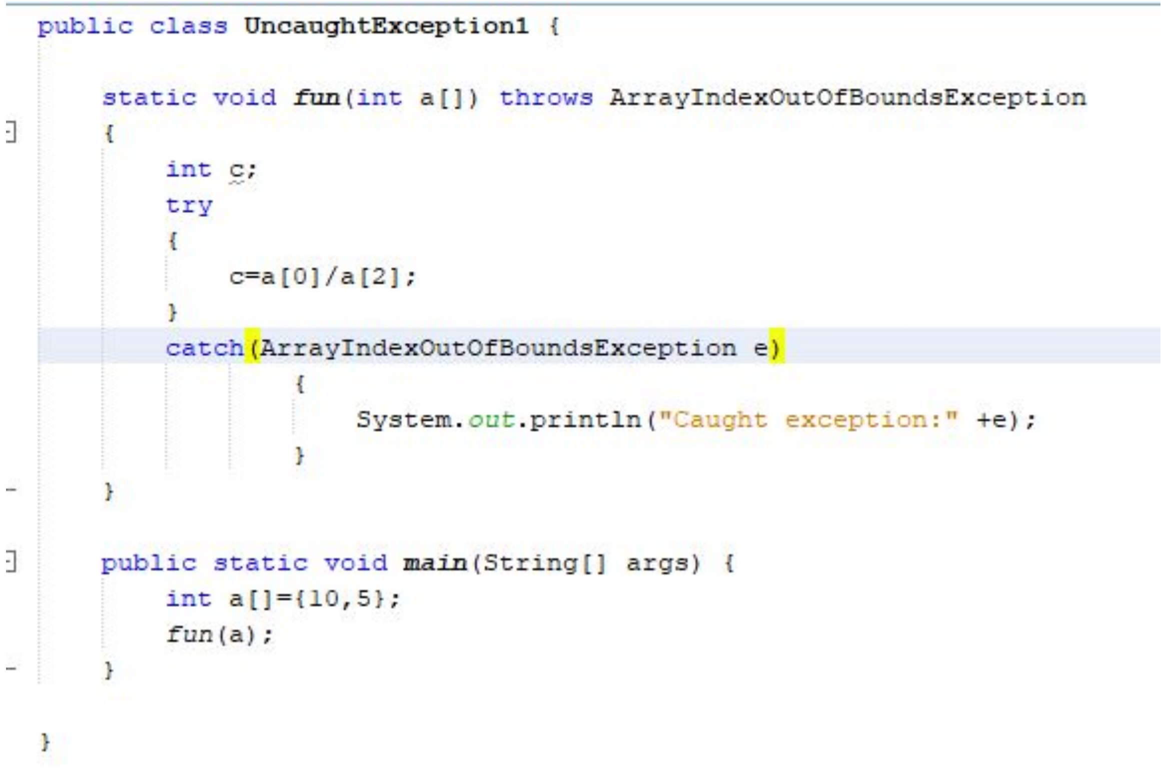
output



```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2
        at uncaughtexception1.UncaughtException1.fun(UncaughtException1.java:20)
        at uncaughtexception1.UncaughtException1.main(UncaughtException1.java:25)
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

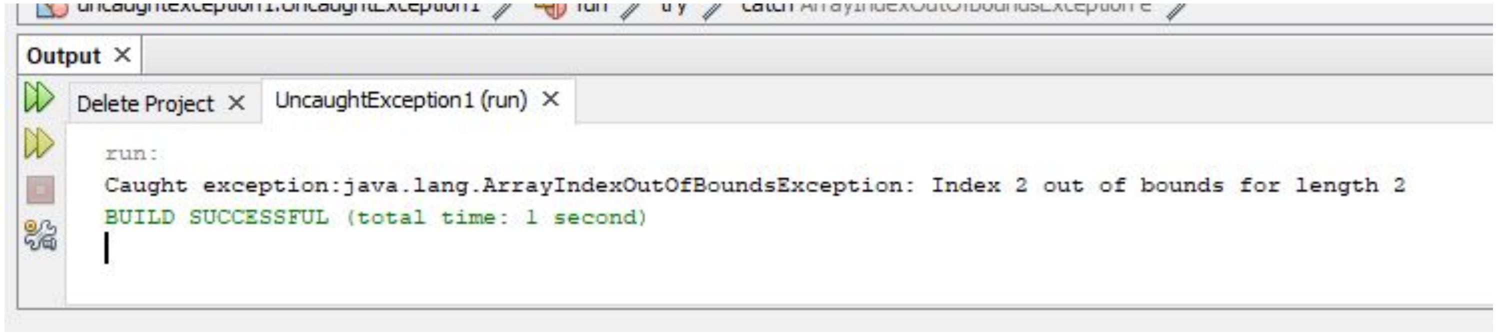
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

[i]DEMO OF ArrayIndexOutOfBoundsException



```
public class UncaughtException1 {

    static void fun(int a[]) throws ArrayIndexOutOfBoundsException
    {
        int c;
        try
        {
            c=a[0]/a[2];
        }
        catch(ArrayIndexOutOfBoundsException e)
            {
                System.out.println("Caught exception:" +e);
            }
    }

    public static void main(String[] args) {
        int a[]={10,5};
        fun(a);
    }

}
```

OUTPUT



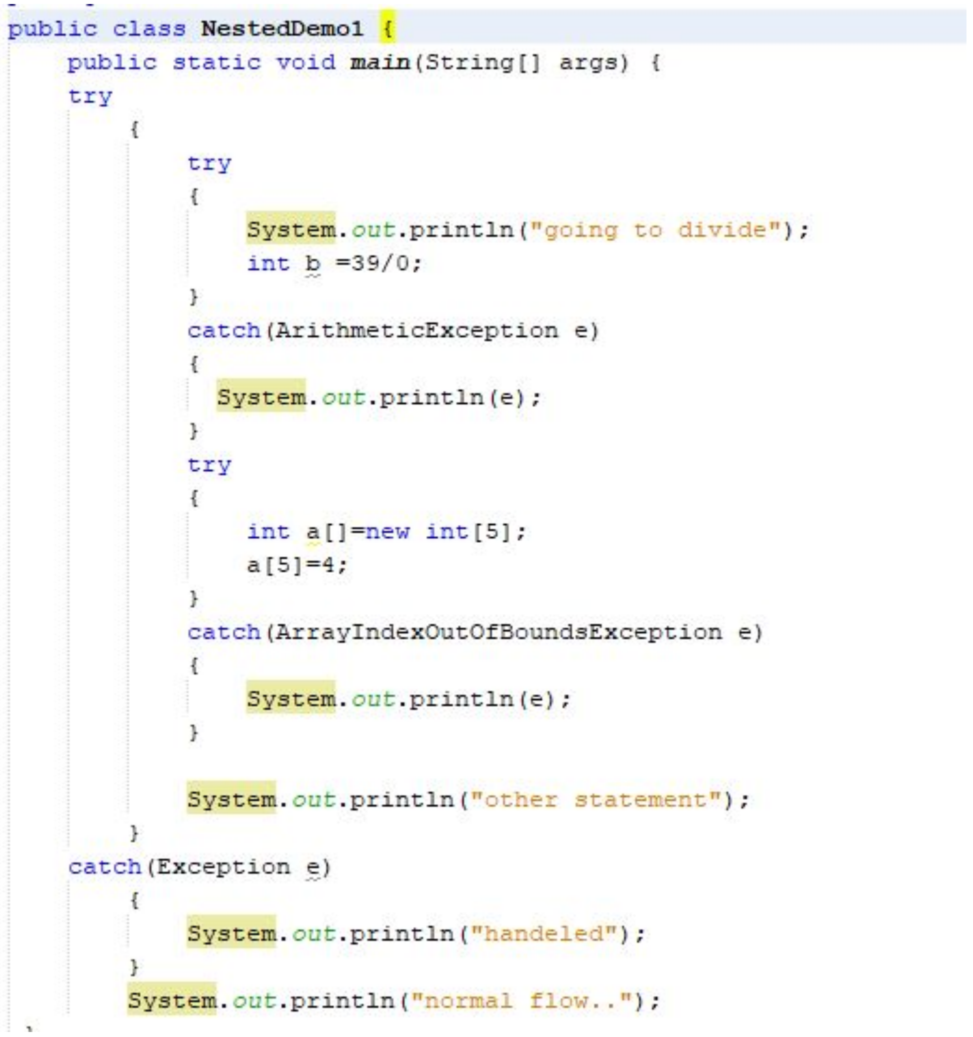
```
run:
Caught exception:java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2
BUILD SUCCESSFUL (total time: 1 second)
```

Nested Try Statements

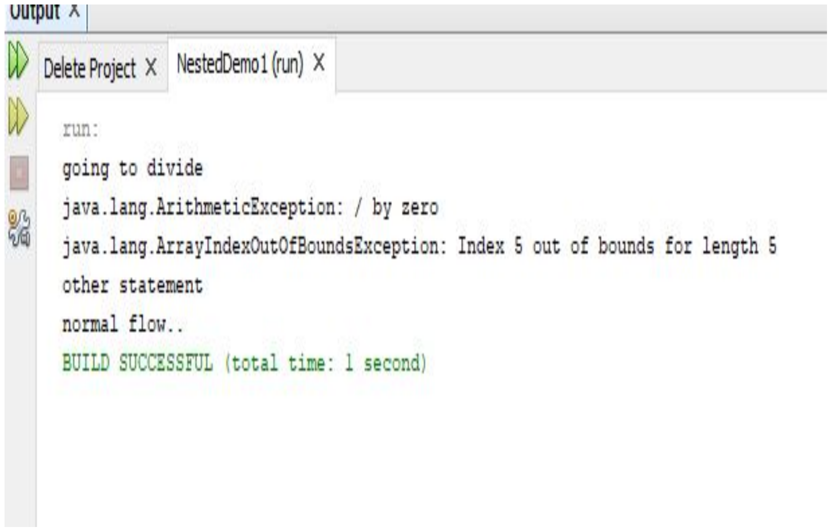
The try block within a try block is known as nested try block in java.

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;
      try
    {
       statement ;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
```



```
public class NestedDemo1 {
    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("handeled");
        }
        System.out.println("normal flow..");
```



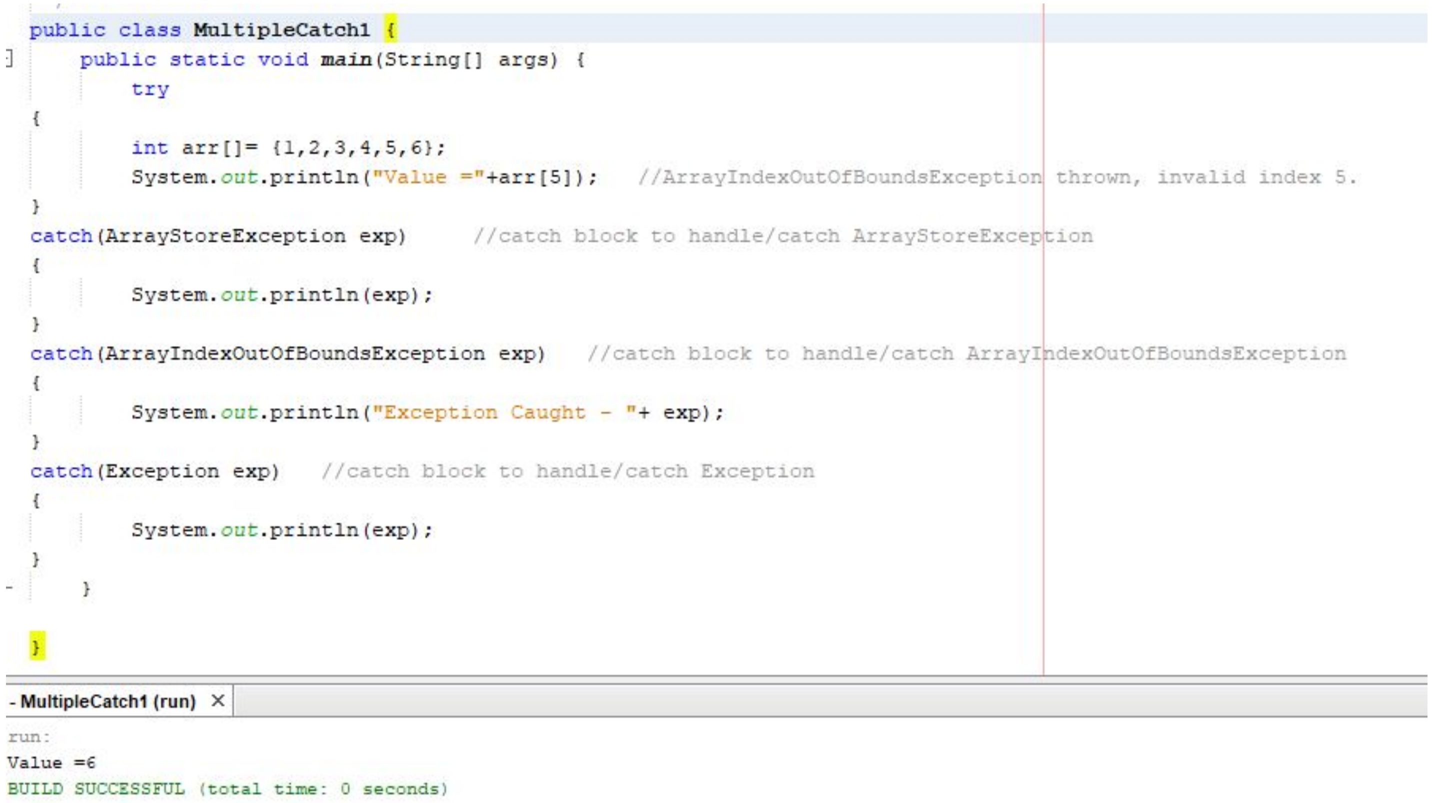
```
run:
going to divide
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
other statement
normal flow..
BUILD SUCCESSFUL (total time: 1 second)
```

Multiple Catch Block

- ➔ As I mentioned above, a single try block can have any number of catch blocks.
- ➔ we had to catch only one exception type in each catch block. So whenever we needed to handle more than one specific exception, but take same action for all exceptions, then we had to have more than one catch block containing the same code.

SYNTAX:

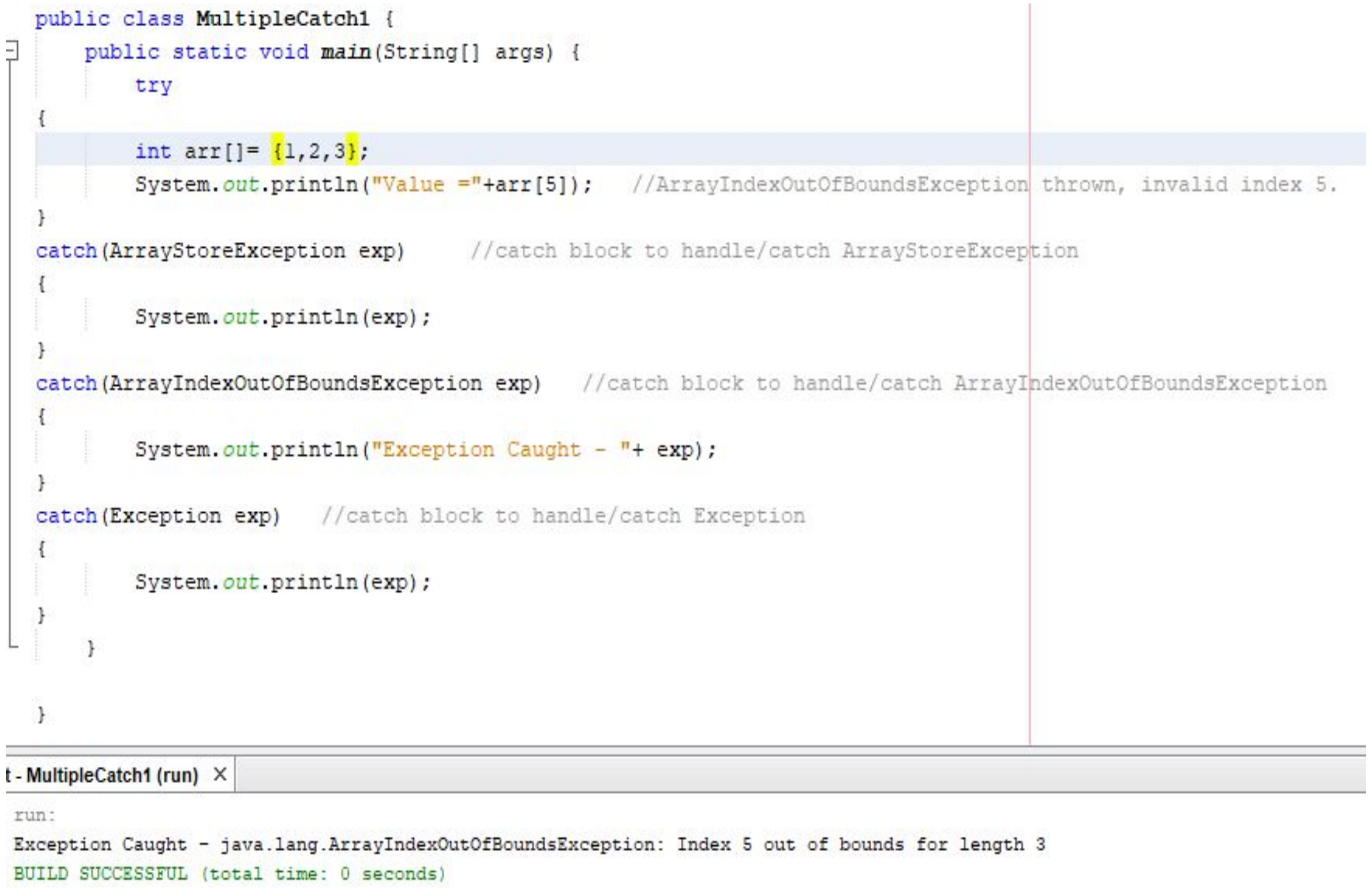
```
try

{

//Exception gets generated here

}

catch (ExceptionType name)

 {

//Exception is handled here

 }
catch (ExceptionType name)

 {

//Exception is handled here

 }
```



```
public class MultipleCatch1 {
    public static void main(String[] args) {
        try
{
        int arr[]= {1,2,3,4,5,6};
        System.out.println("Value ="+arr[5]);   //ArrayIndexOutOfBoundsException thrown, invalid index 5.
}
catch(ArrayStoreException exp)      //catch block to handle/catch ArrayStoreException
{
        System.out.println(exp);
}
catch(ArrayIndexOutOfBoundsException exp)   //catch block to handle/catch ArrayIndexOutOfBoundsException
{
        System.out.println("Exception Caught - "+ exp);
}
catch(Exception exp)   //catch block to handle/catch Exception
{
        System.out.println(exp);
}
    }

}
```

- MultipleCatch1 (run) ×

```
run:
Value =6
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
public class MultipleCatch1 {
    public static void main(String[] args) {
        try
{
        int arr[]= {1,2,3};
        System.out.println("Value ="+arr[5]);   //ArrayIndexOutOfBoundsException thrown, invalid index 5.
}
catch(ArrayStoreException exp)     //catch block to handle/catch ArrayStoreException
{
        System.out.println(exp);
}
catch(ArrayIndexOutOfBoundsException exp)   //catch block to handle/catch ArrayIndexOutOfBoundsException
{
        System.out.println("Exception Caught - "+ exp);
}
catch(Exception exp)   //catch block to handle/catch Exception
{
        System.out.println(exp);
}
    }

}
```

t - MultipleCatch1 (run)

```
run:
Exception Caught - java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Using Throws

According to the Java Compiler - "we must either catch checked exceptions by providing appropriate try-catch block or we should declare them, using *throws*."

Hence, when a **method** doesn't want to catch/handle one or more **checked exceptions** by providing an appropriate **try-catch** block within, it **must** use **throws** keyword in its method signature to declare that it does not handle but only throws such **checked exceptions**.

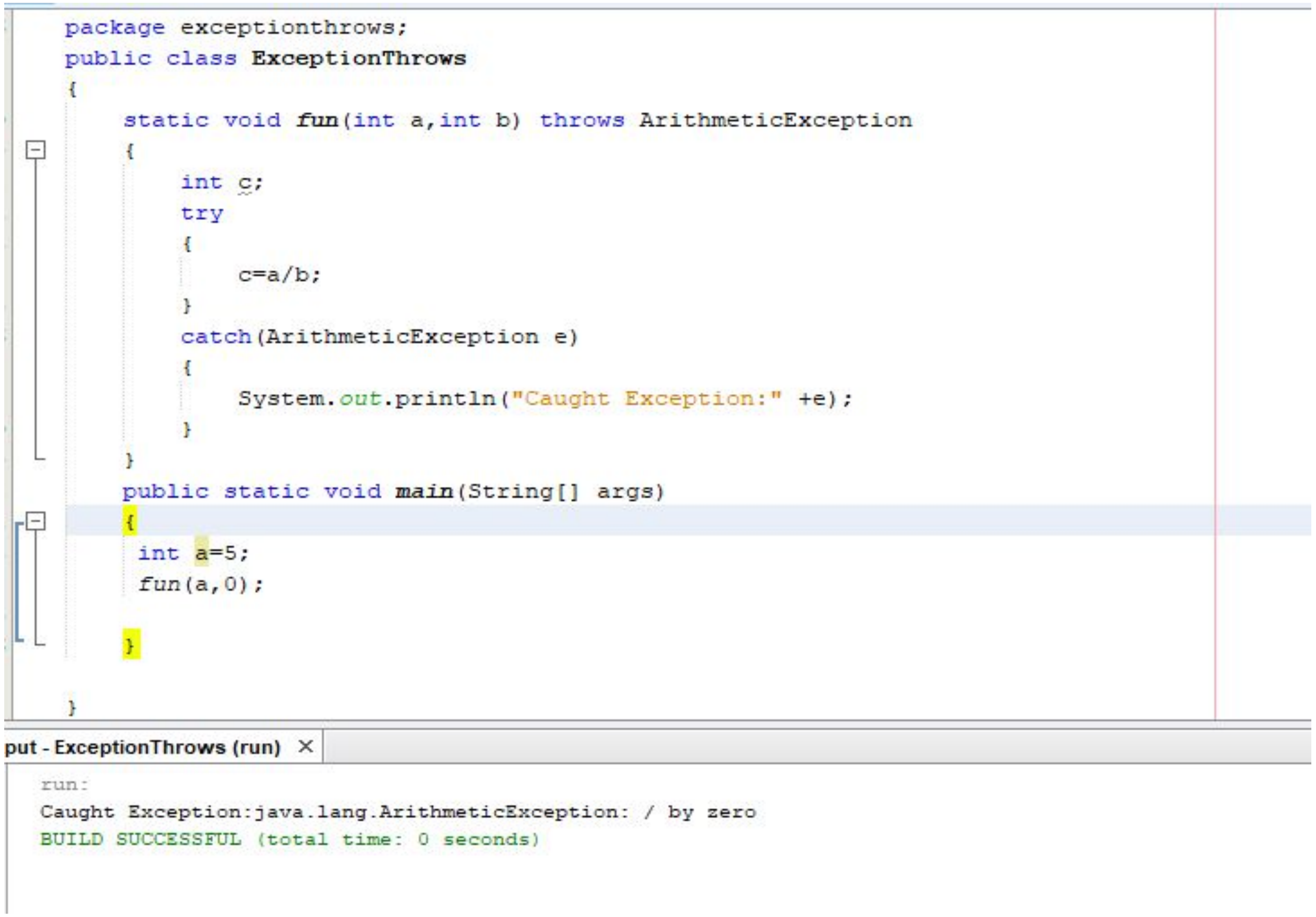
SYNTAX:

[i]

```
//Declaring one checked exception
using throws keyword

public void method1() throws
IOException //method signature{}
```

[ii]

```
method_name(parameter_list)throws
exception_list

{}
```



```
package exceptionthrows;
public class ExceptionThrows
{
    static void fun(int a,int b) throws ArithmeticException
    {
        int c;
        try
        {
            c=a/b;
        }
        catch(ArithmeticException e)
        {
            System.out.println("Caught Exception:" +e);
        }
    }
    public static void main(String[] args)
    {
     int a=5;
     fun(a,0);

    }

}
```

```
run:
Caught Exception:java.lang.ArithmeticException: / by zero
BUILD SUCCESSFUL (total time: 0 seconds)
```

Using throw

➔ you may use the **throw** keyword when you explicitly want to throw an exception. The keyword throw is followed by an object of the exception class that you want to throw. Using **throw** keyword, an exception can be explicitly thrown from within the two places in a Java program -

- *try-block* or,
- *catch-block*.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Throw in Try Block

To throw an exception out of **try block**, keyword **throw** is followed by creating a new object of exception class that we wish to throw. For example -

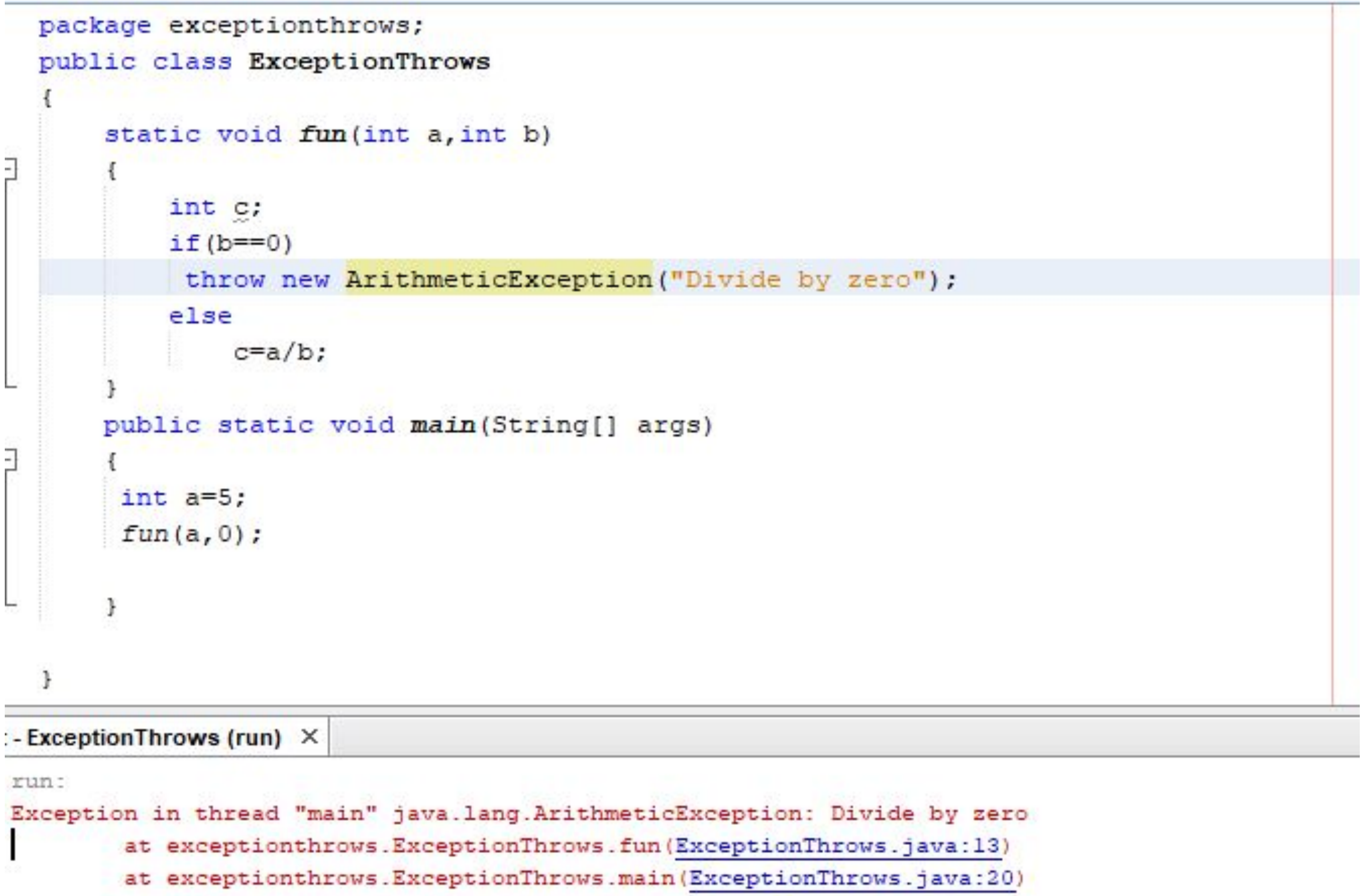
```
try

{

// An object of IOException exception class is created using "new" keyword.

        throw new IOException();

}
```

Throw in Catch Block

To throw an exception out of catch block, keyword throw is followed by object reference of the exception class that was caught by catch block. For example -

```
catch(IOException io)

{

        throw io;       // An existing exception class object referenced by "io"  of type "IOException",  is thrown.

}
```

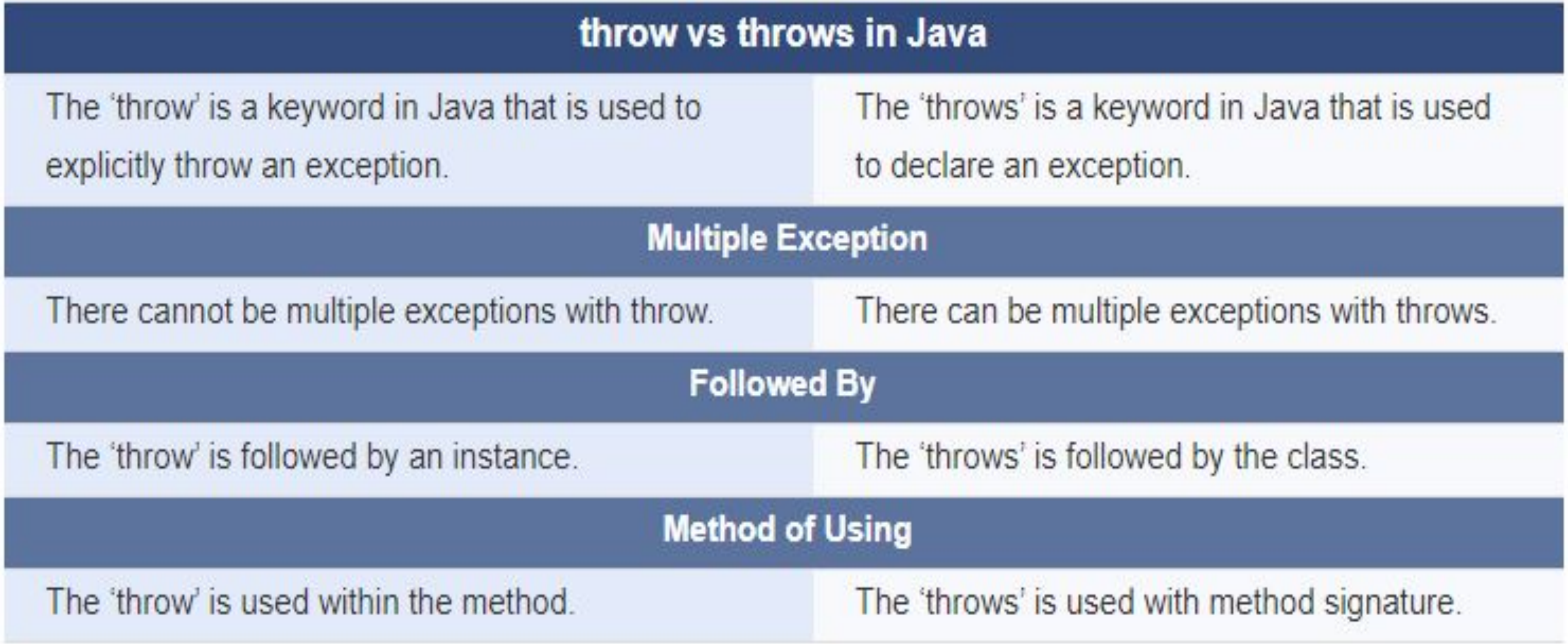
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY



```
package exceptionthrows;
public class ExceptionThrows
{
    static void fun(int a,int b)
    {
        int c;
        if(b==0)
         throw new ArithmeticException("Divide by zero");
        else
            c=a/b;
    }
    public static void main(String[] args)
    {
     int a=5;
     fun(a,0);

    }

}
```

```
run:
Exception in thread "main" java.lang.ArithmeticException: Divide by zero
        at exceptionthrows.ExceptionThrows.fun(ExceptionThrows.java:13)
        at exceptionthrows.ExceptionThrows.main(ExceptionThrows.java:20)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

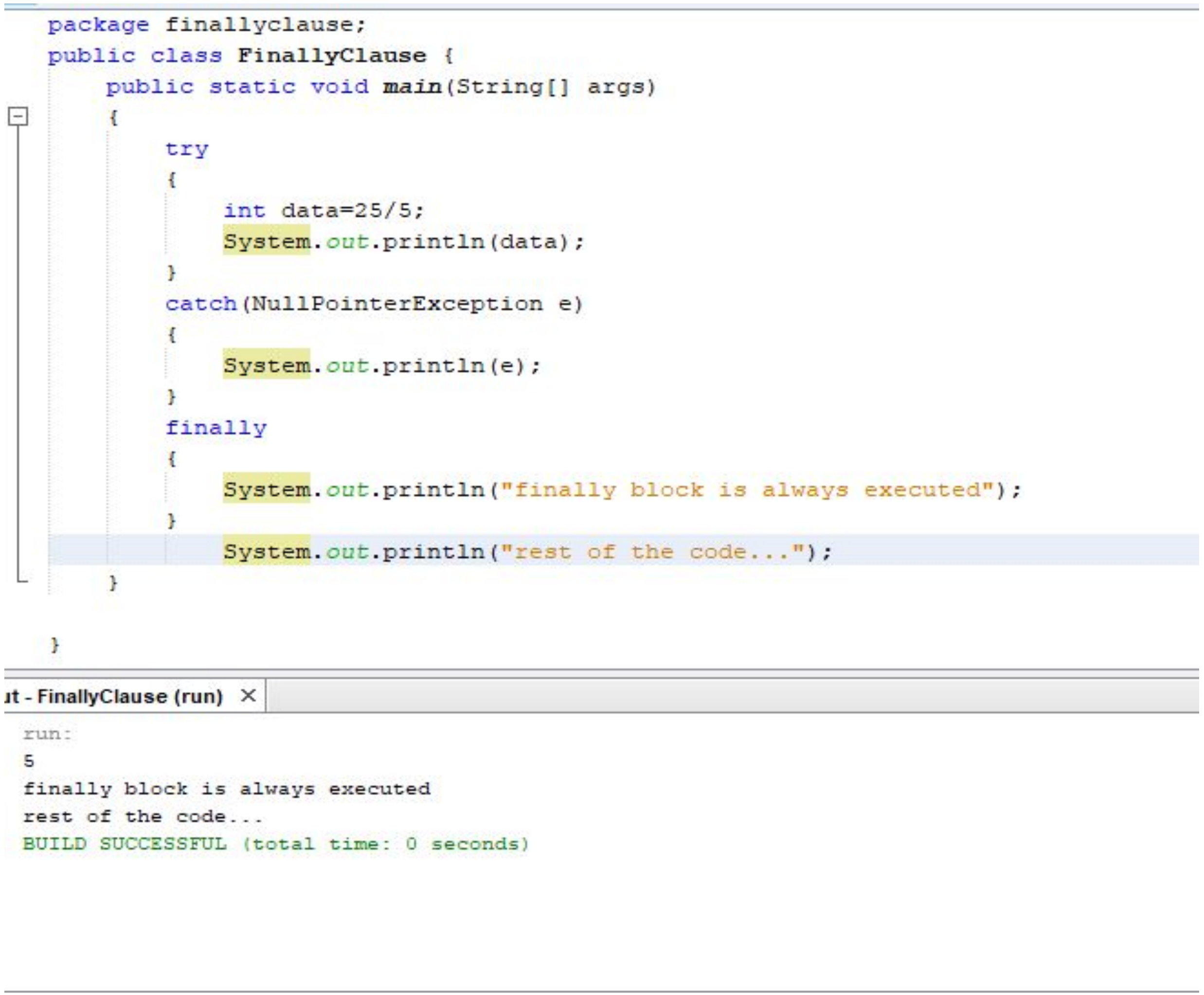


throw vs throws in Java	
The 'throw' is a keyword in Java that is used to explicitly throw an exception.	The 'throws' is a keyword in Java that is used to declare an exception.
Multiple Exception	
There cannot be multiple exceptions with throw.	There can be multiple exceptions with throws.
Followed By	
The 'throw' is followed by an instance.	The 'throws' is followed by the class.
Method of Using	
The 'throw' is used within the method.	The 'throws' is used with method signature.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Finally Clause

- **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
- Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

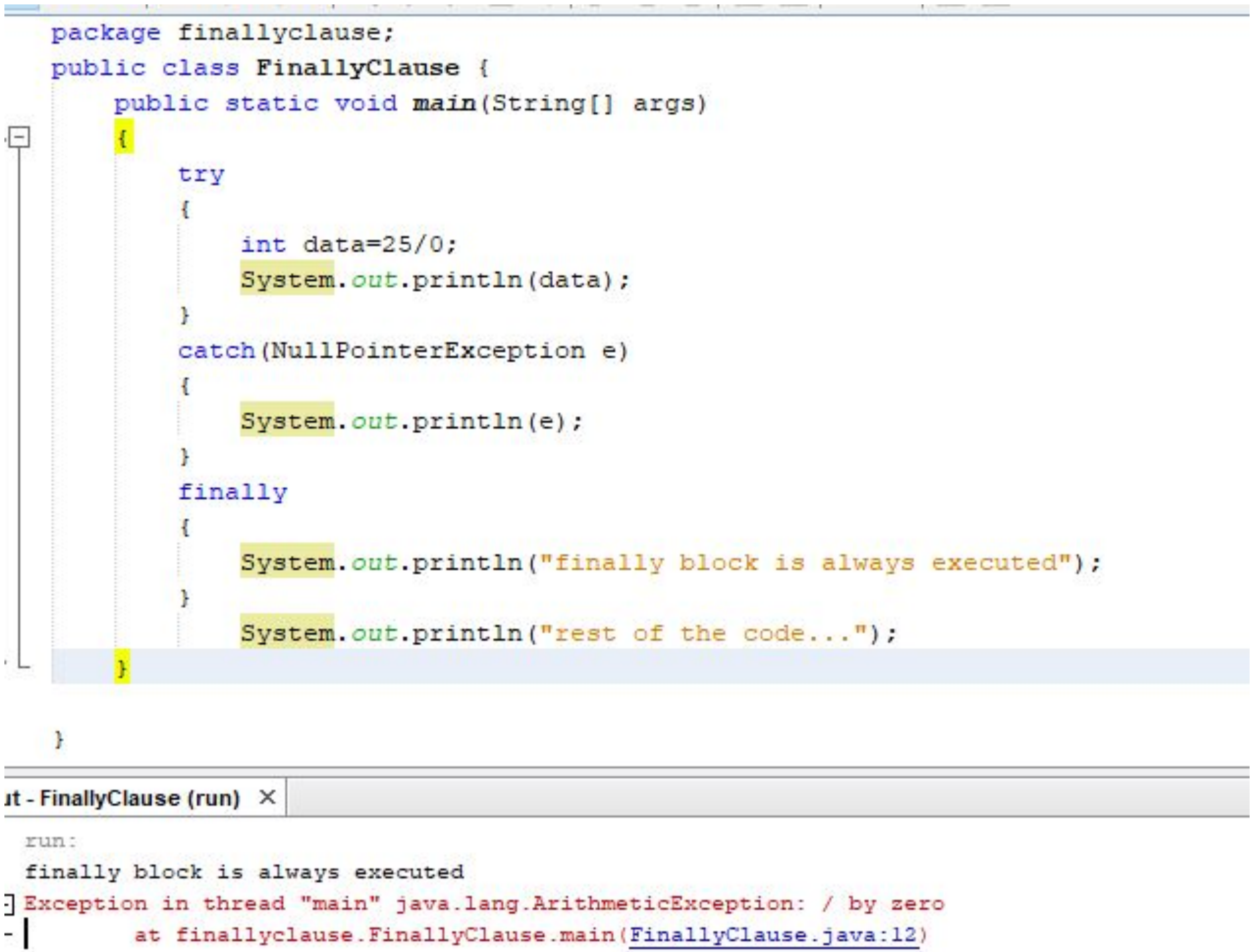


```
package finallyclause;
public class FinallyClause {
    public static void main(String[] args)
    {
        try
        {
            int data=25/5;
            System.out.println(data);
        }
        catch(NullPointerException e)
        {
            System.out.println(e);
        }
     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

}
```

ut - FinallyClause (run) ✕

```
run:
5
finally block is always executed
rest of the code...
BUILD SUCCESSFUL (total time: 0 seconds)
```

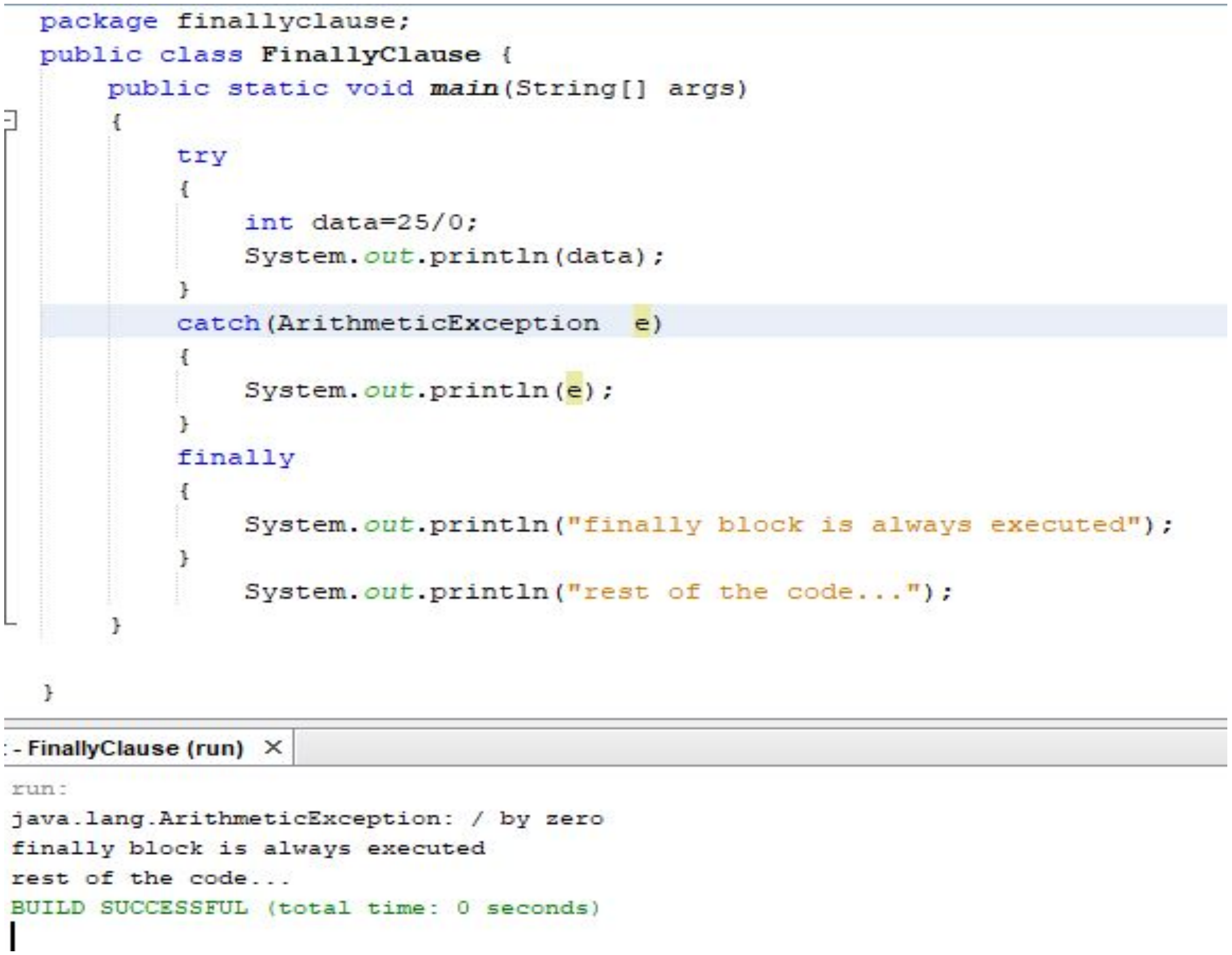
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```
package finallyclause;
public class FinallyClause {
    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...");
    }

}
```

```
run:
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at finallyclause.FinallyClause.main(FinallyClause.java:12)
```

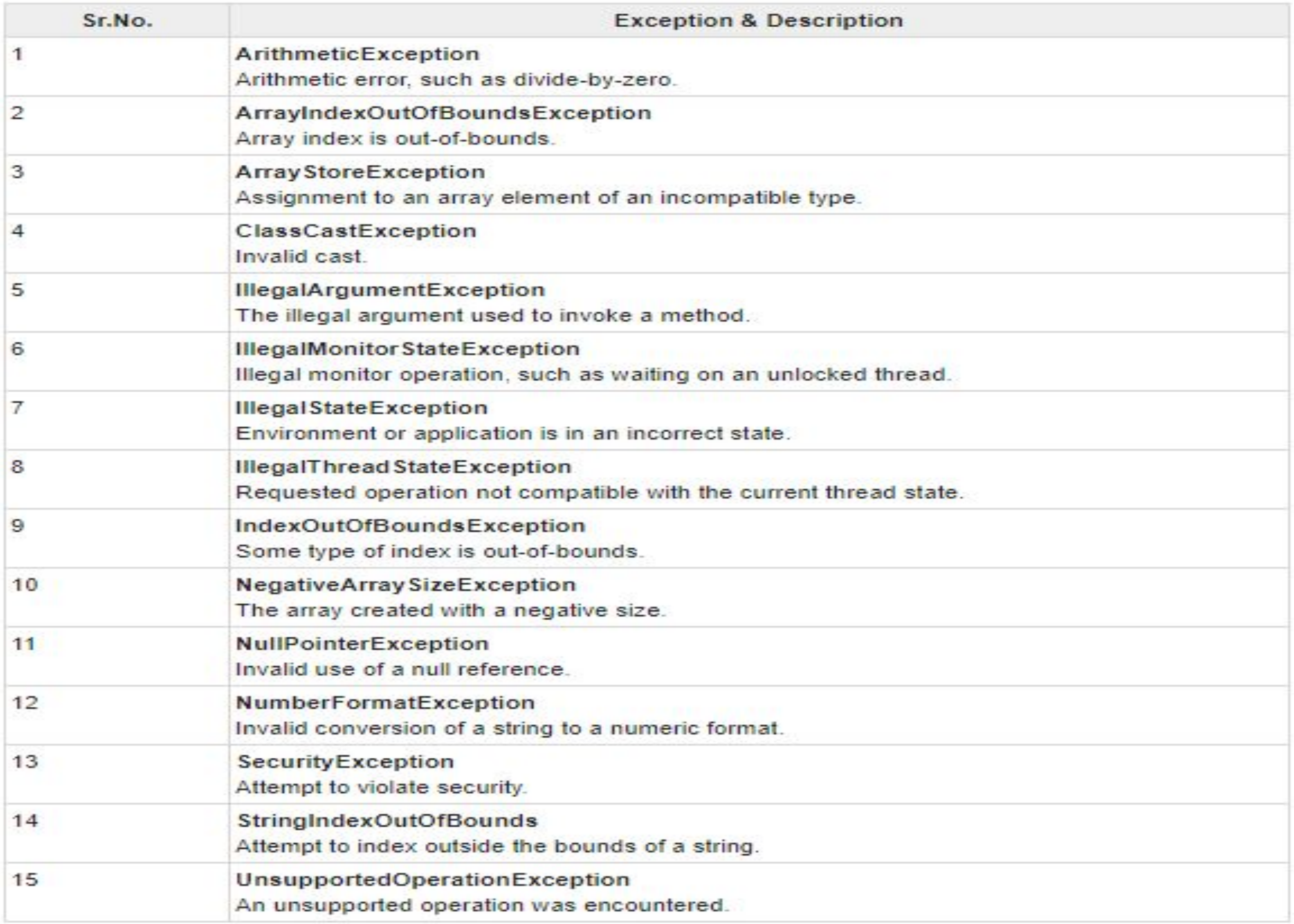
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```
package finallyclause;
public class FinallyClause {
    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...");
    }

}
```

```
run:
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...
BUILD SUCCESSFUL (total time: 0 seconds)
```

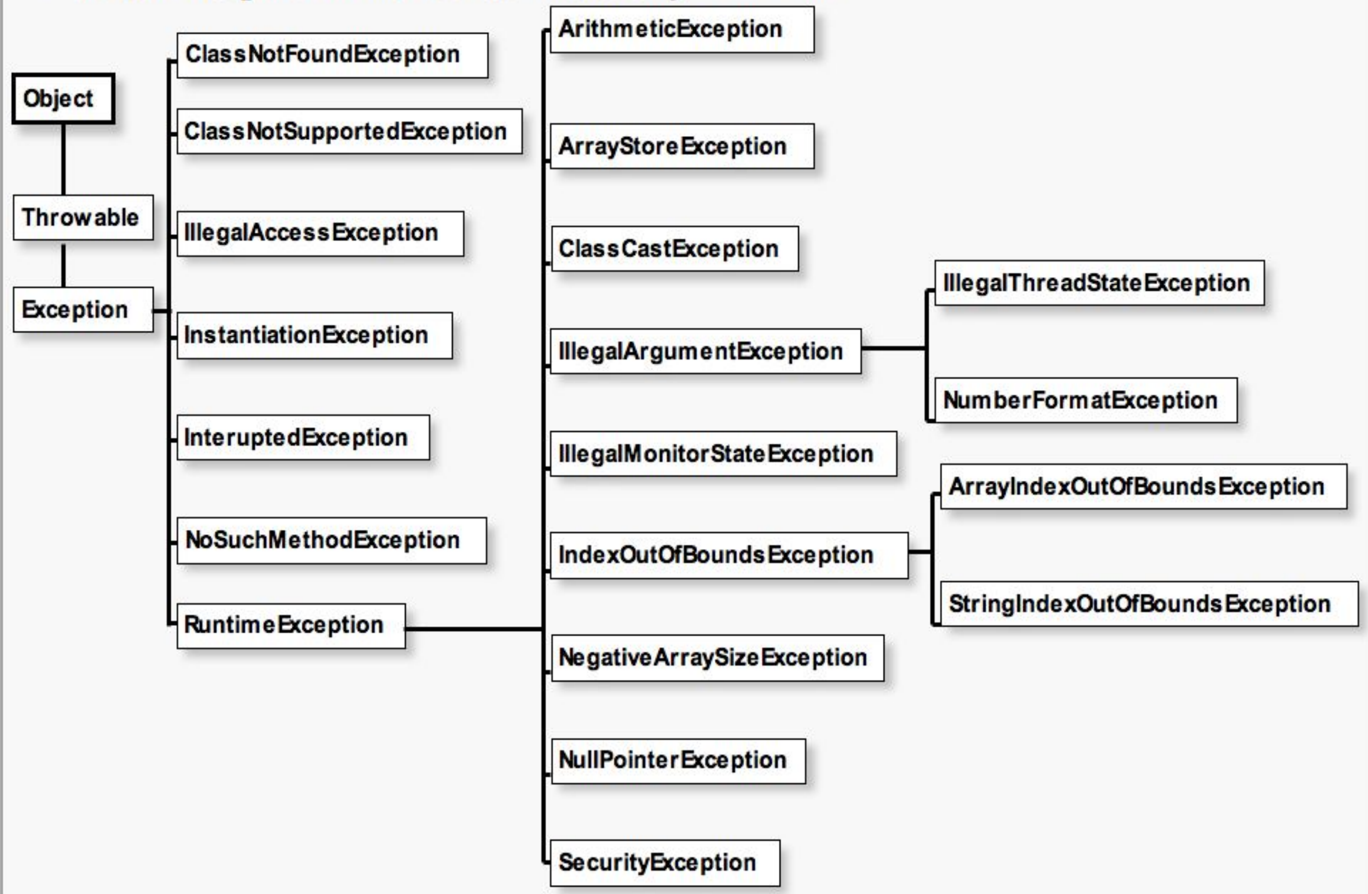
Build In Exceptions



Sr.No.	Exception & Description
1	**ArithmeticException** Arithmetic error, such as divide-by-zero.
2	**ArrayIndexOutOfBoundsException** Array index is out-of-bounds.
3	**ArrayStoreException** Assignment to an array element of an incompatible type.
4	**ClassCastException** Invalid cast.
5	**IllegalArgumentException** The illegal argument used to invoke a method.
6	**IllegalMonitorStateException** Illegal monitor operation, such as waiting on an unlocked thread.
7	**IllegalStateException** Environment or application is in an incorrect state.
8	**IllegalThreadStateException** Requested operation not compatible with the current thread state.
9	**IndexOutOfBoundsException** Some type of index is out-of-bounds.
10	**NegativeArraySizeException** The array created with a negative size.
11	**NullPointerException** Invalid use of a null reference.
12	**NumberFormatException** Invalid conversion of a string to a numeric format.
13	**SecurityException** Attempt to violate security.
14	**StringIndexOutOfBounds** Attempt to index outside the bounds of a string.
15	**UnsupportedOperationException** An unsupported operation was encountered.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Java Exception Inheritance Hierarchy Java.lang



AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Throwable Class

The java.lang.Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.

Class Declaration

Following is the declaration for java.lang.Throwable class –

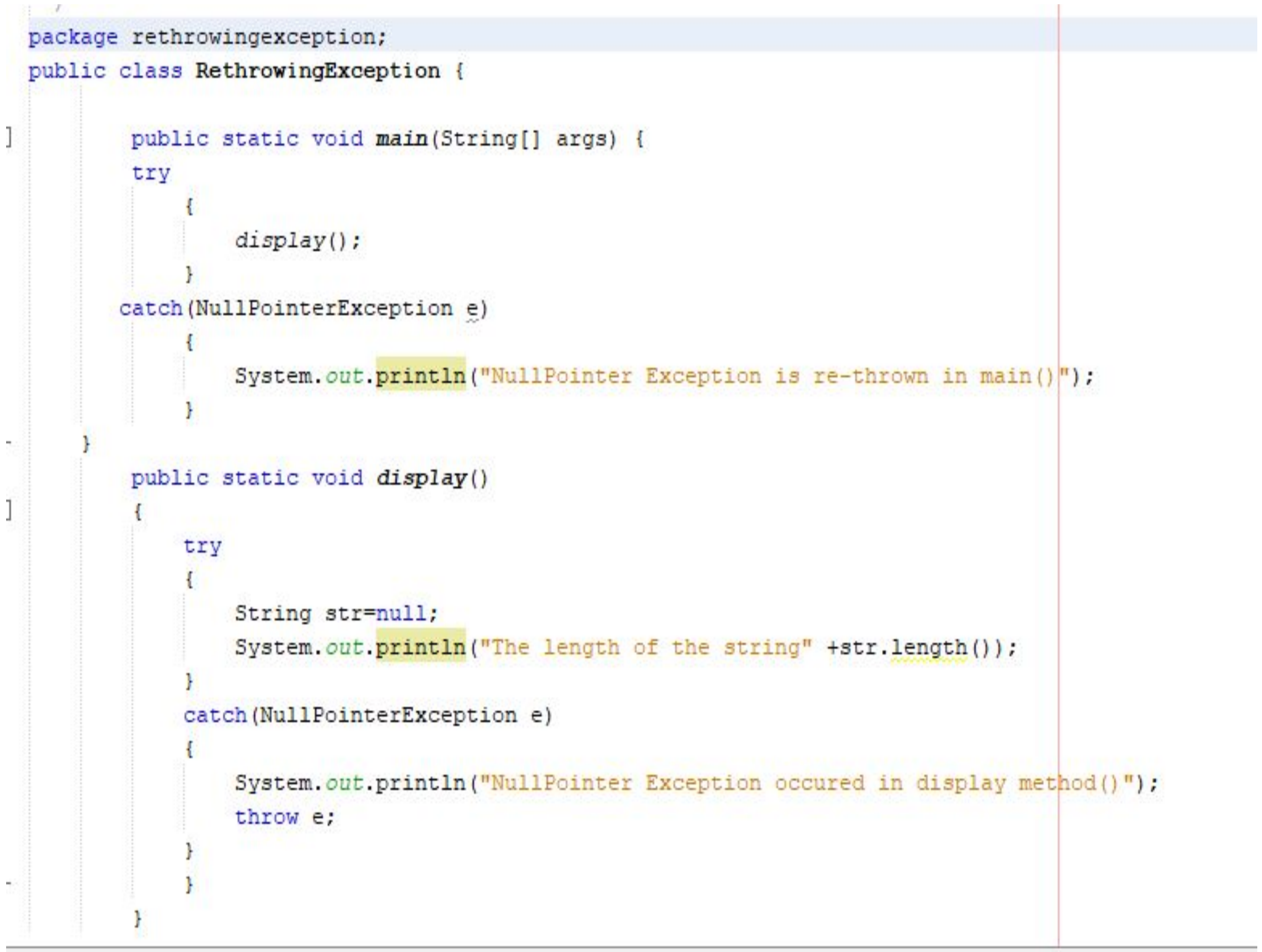
```
public class Throwable

   extends Object

       implements Serializable
```

Class constructors

Sr.No.	Constructor & Description
1	**Throwable()** This constructs a new throwable with null as its detail message.
2	**Throwable(String message)** This constructs a new throwable with the specified detail message.
3	**Throwable(String message, Throwable cause)** This constructs a new throwable with the specified detail message and cause.
4	**Throwable(Throwable cause)** This constructs a new throwable with the specified cause and a detail message of (cause==null ? null : cause.toString()) (which typically contains the class and detail message of cause).

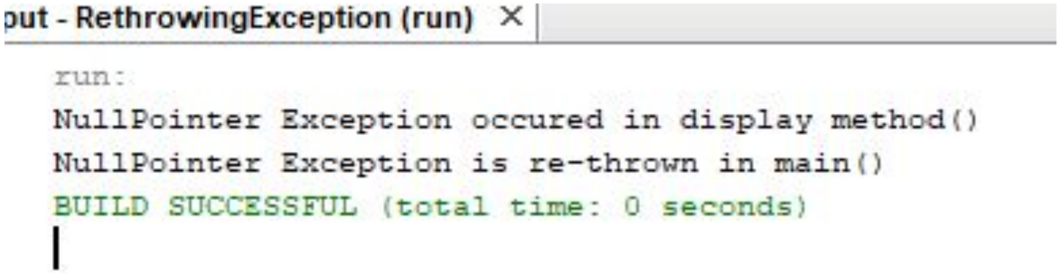
Re-throwing Exception:

➔ An exception can be rethrown in a catch block. This action will cause the exception to be passed to the calling method. If the rethrow operation occurs in the main method then the exception is passed to the JVM and displayed on the console. The purpose of the rethrow operation is to get the attention of the outside world that an exception has occurred and at the same time perform any contingency logic (such as logging) in the catch block.



```
package rethrowingexception;
public class RethrowingException {

        public static void main(String[] args) {
        try
            {
                display();
            }
        catch(NullPointerException e)
            {
                System.out.println("NullPointer Exception is re-thrown in main()");
            }
    }
        public static void display()
        {
            try
            {
                String str=null;
                System.out.println("The length of the string" +str.length());
            }
            catch(NullPointerException e)
            {
                System.out.println("NullPointer Exception occured in display method()");
                throw e;
            }
            }
        }
```

OUTPUT:



```
run:
NullPointer Exception occured in display method()
NullPointer Exception is re-thrown in main()
BUILD SUCCESSFUL (total time: 0 seconds)
```

Chained Exception

- Chained Exceptions allows to relate one exception with another exception, i.e one exception describes cause of another exception.
- For example, consider a situation division operation a/b in which a b is 0 which throws an ArithmeticException because of an attempt to divide by zero but the actual cause of exception was an I/O error (bcz wrong input value to variable b)which caused the divisor to be zero. The method will throw only ArithmeticException to the caller. So the caller would not come to know about the actual cause of exception. Chained Exception is used in such type of situations.

CONT...

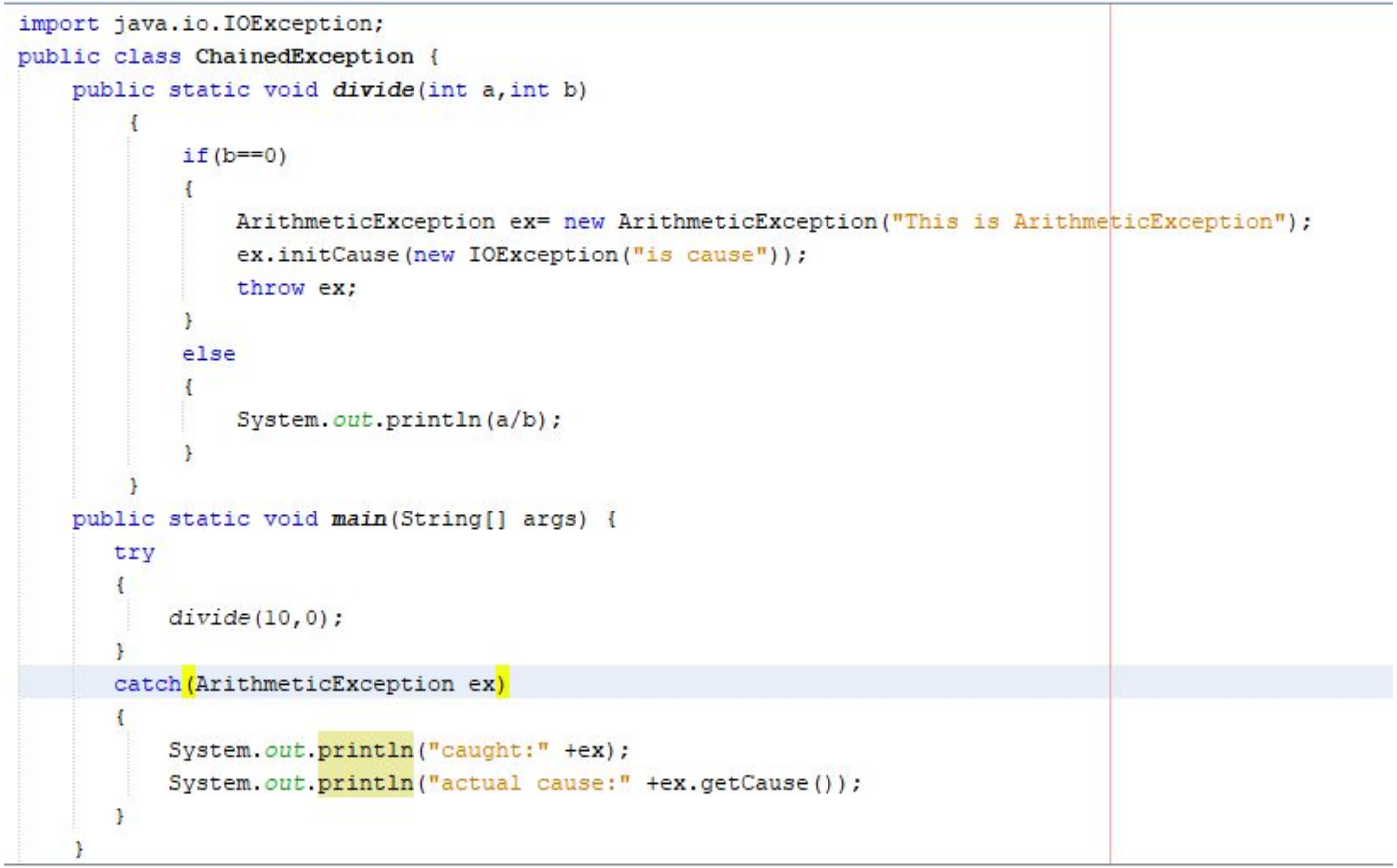
The throwable class supports chained exception using the following methods:

Constructors

1. **Throwable(Throwable cause)** - the cause is the current exception.
2. **Throwable(String msg, Throwable cause)** - msg is the exception message, the cause is the current exception.

Methods

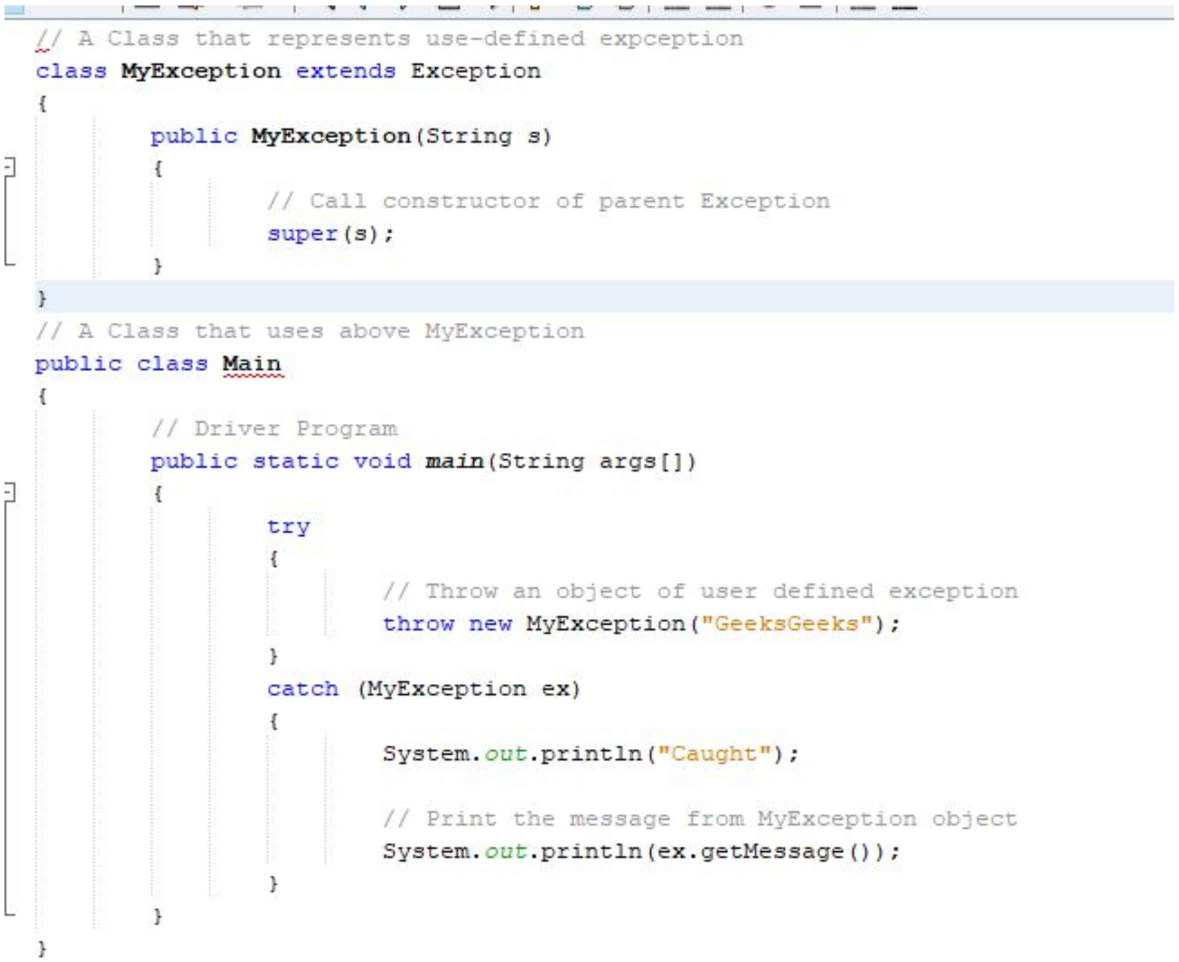
1. **getCause** - returns actual cause.
2. **initCause(Throwable cause)** - sets the cause for calling an exception.


```
import java.io.IOException;
public class ChainedException {
    public static void divide(int a,int b)
        {
            if(b==0)
            {
                ArithmeticException ex= new ArithmeticException("This is ArithmeticException");
                ex.initCause(new IOException("is cause"));
                throw ex;
            }
            else
            {
                System.out.println(a/b);
            }
        }
    public static void main(String[] args) {
        try
        {
            divide(10,0);
        }
        catch(ArithmeticException ex)
        {
            System.out.println("caught:" +ex);
            System.out.println("actual cause:" +ex.getCause());
        }
    }
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Defining Custom Exception Classes

- ➔ Java provides us facility to create our own exceptions which are basically derived classes of Exception. For example MyException in below code extends the Exception class.
- ➔ We pass the string to the constructor of the super class- Exception which is obtained using “getMessage()” function on the object created.



```java
// A Class that represents use-defined expception
class MyException extends Exception
{
        public MyException(String s)
        {
                // Call constructor of parent Exception
                super(s);
        }
}
// A Class that uses above MyException
public class Main
{
        // Driver Program
        public static void main(String args[])
        {
                try
                {
                        // Throw an object of user defined exception
                        throw new MyException("GeeksGeeks");
                }
                catch (MyException ex)
                {
                        System.out.println("Caught");

                        // Print the message from MyException object
                        System.out.println(ex.getMessage());
                }
        }
}
```

VIDEO LINKS FOR EXCEPTION HANDLING

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Introduction to Exception Handling, How Exception is Handled
https://www.youtube.com/watch?v=HDVUos1IlyY

https://www.youtube.com/watch?v=ohpCMpderow

Try Catch in Java with Example, Java Exception Handling
https://www.youtube.com/watch?v=-fXnRkB8FFc

Throw Keyword in Java Exception Handling with Example
https://www.youtube.com/watch?v=j9M69wdEU-0

Finally Keyword in Java with Complete Example
https://www.youtube.com/watch?v=c_Jun_ouCg8

Throws Keyword in Java Exception Handling, Throw Vs. Throws
https://www.youtube.com/watch?v=nsJvdOrjBRA

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CONTT..

Exception Hierarchy in Java

https://www.youtube.com/watch?v=S5K1T4GqdJE

Throwable class and exception class?

https://www.youtube.com/watch?v=o4NfR8AvaQ0

What is chained Exceptions?

https://www.youtube.com/watch?v=O16v69dCUj8

User Defined Exceptions or Custom Exception

https://www.youtube.com/watch?v=vkN2v3LsoEo

https://www.youtube.com/watch?v=NsctooZANVk

I/O

File Class and its Input and Output

- The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination
- The File class contains several methods for working with the path name, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.
- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the getParent() method of this class.
- First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.

How to create a File Object?

➔ A File object is created by passing in a String that represents the name of a file, or a String or another File object. For example,

```
File a = new File("/usr/local/bin/geeks");
```

➔ defines an abstract file name for the geeks file in directory /usr/local/bin. This is an absolute abstract file name.

Absolute Path

- ➔ Simply, a path is absolute if it starts with the root element of the file system. In windows, the root element is a drive e.g. *C:\\*, *D:\\*, while in unix it is denoted by *"/"* character.
- ➔ An absolute path is complete in that no other information is required to locate the file, it usually holds the complete directory list starting from the root node of the file system till reaching the file or directory it denotes.
- ➔ Since absolute path is static and platform dependent, it is a bad practice to locate a file using absolute path inside your program, since you will lose the ability to reuse your program on different machines and platforms.

```
File absoluteFile = new File("D:\\sample-documents\\pdf-sample.pdf");
```

Relative Path

- ➔ A relative path is a path which doesn’t start with the root element of the file system. It is simply the path needed in order to locate the file from within the current directory of your program. It is not complete and needs to be combined with the current directory path in order to reach the requested file.
- ➔ In order to construct a rigid and platform independent program, it is a common convention to use a relative path when locating a file inside your program.

```
File relativeFile = new File("/sample-documents/pdf-sample.pdf");
```

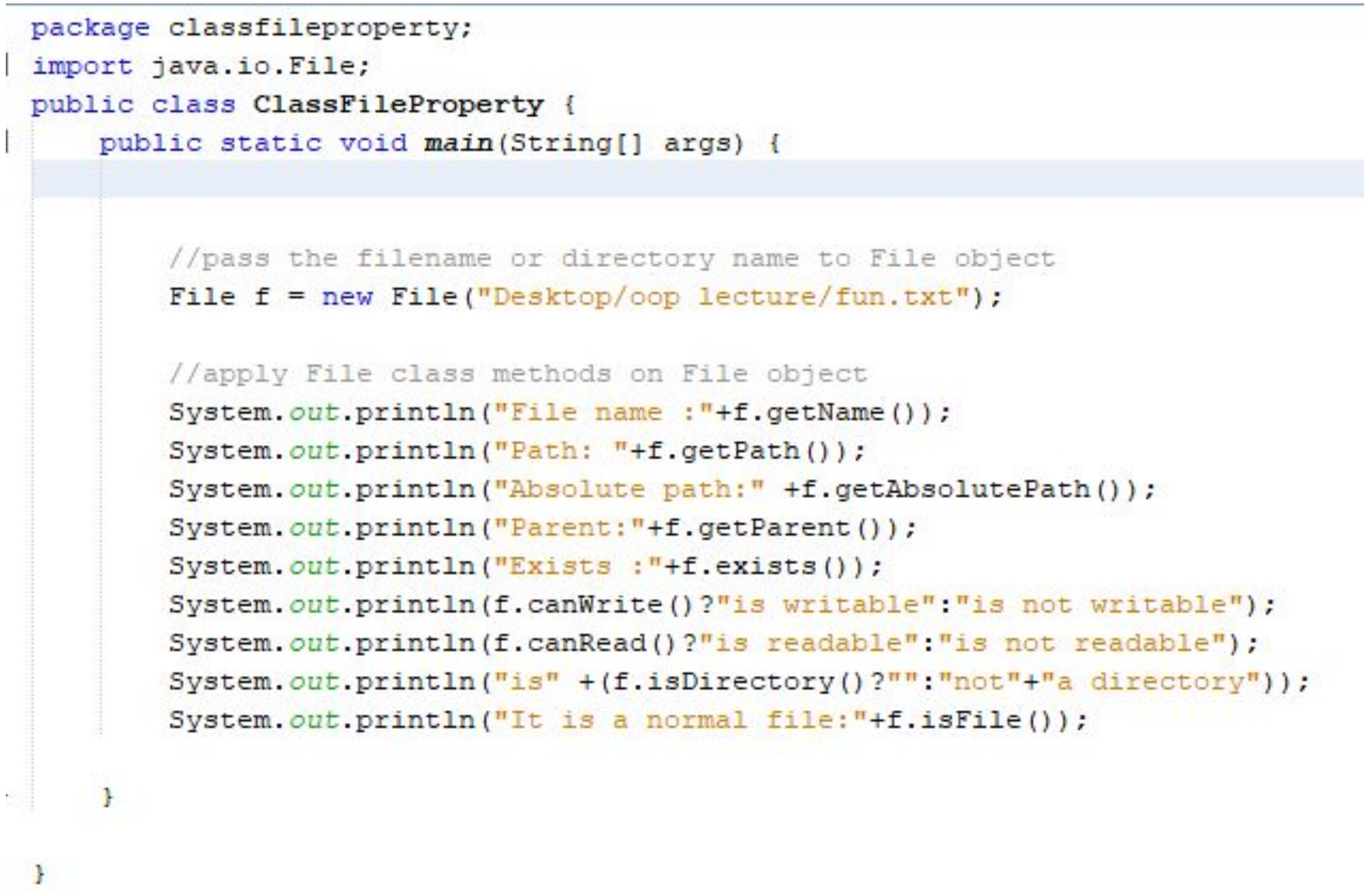
Constructor:

- **File(File parent, String child) :** Creates a new File instance from a parent abstract pathname and a child pathname string.
- **File(String pathname) :** Creates a new File instance by converting the given pathname string into an abstract pathname.
- **File(String parent, String child) :** Creates a new File instance from a parent pathname string and a child pathname string.
- **File(URI uri) :** Creates a new File instance by converting the given file: URI into an abstract pathname.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

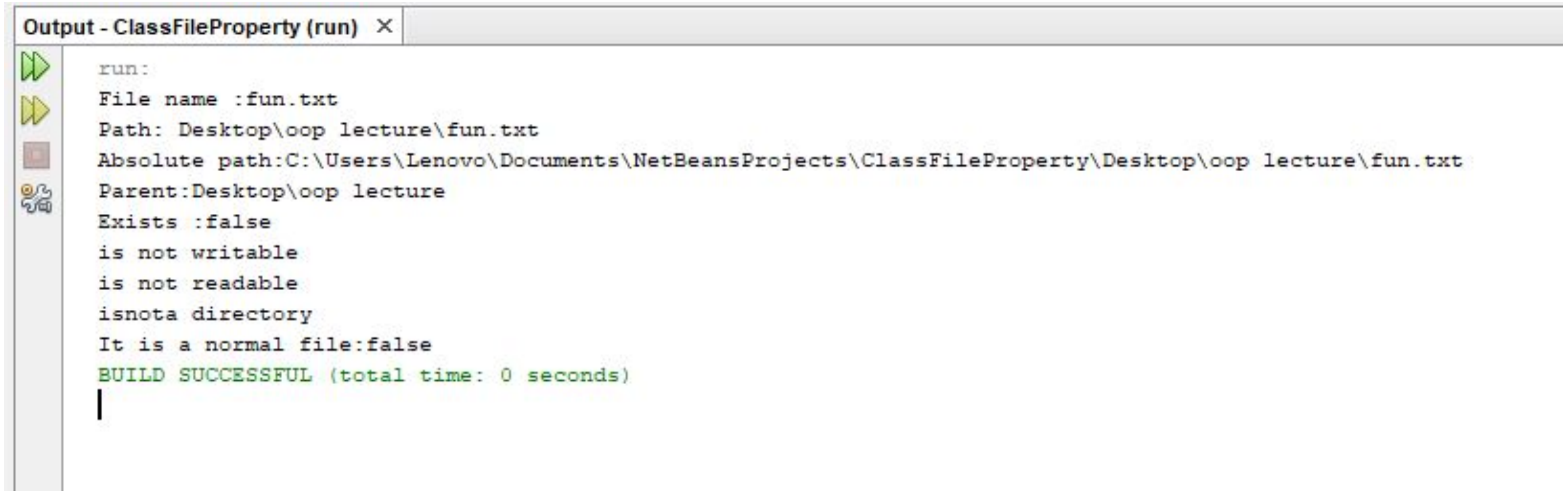
Methods:

1. **String getName() :** Returns the name of the file or directory denoted by this abstract pathname.
2. **String getParent() :** Returns the pathname string of this abstract pathname's parent.
3. **File getParentFile() :** Returns the abstract pathname of this abstract pathname's parent.
4. **String getPath() :** Converts this abstract pathname into a pathname string.
5. **boolean isDirectory() :** Tests whether the file denoted by this pathname is a directory.
6. **boolean isFile() :** Tests whether the file denoted by this abstract pathname is a normal file.
7. **boolean canExecute() :** Tests whether the application can execute the file denoted by this abstract pathname.
8. **boolean canRead()** : Tests whether the application can read the file denoted by this abstract pathname.
9. **boolean canWrite() :** Tests whether the application can modify the file denoted by this abstract pathname.



```java
package classfileproperty;
import java.io.File;
public class ClassFileProperty {
    public static void main(String[] args) {


        //pass the filename or directory name to File object
        File f = new File("Desktop/oop lecture/fun.txt");

        //apply File class methods on File object
        System.out.println("File name :"+f.getName());
        System.out.println("Path: "+f.getPath());
        System.out.println("Absolute path:" +f.getAbsolutePath());
        System.out.println("Parent:"+f.getParent());
        System.out.println("Exists :"+f.exists());
        System.out.println(f.canWrite()?"is writable":"is not writable");
        System.out.println(f.canRead()?"is readable":"is not readable");
        System.out.println("is" +(f.isDirectory()?"":"not"+"a directory"));
        System.out.println("It is a normal file:"+f.isFile());

    }

}
```

output:

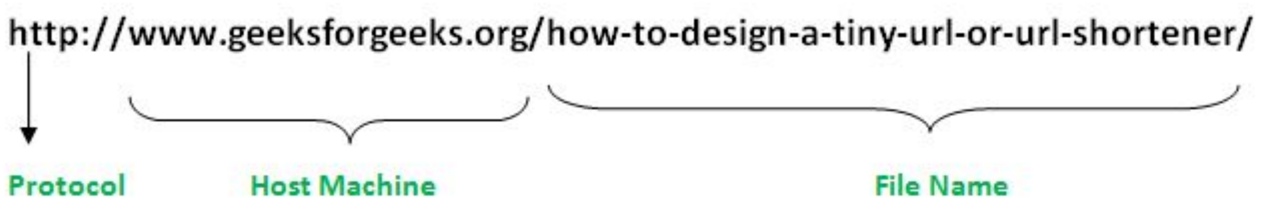


```
run:
File name :fun.txt
Path: Desktop\oop lecture\fun.txt
Absolute path:C:\Users\Lenovo\Documents\NetBeansProjects\ClassFileProperty\Desktop\oop lecture\fun.txt
Parent:Desktop\oop lecture
Exists :false
is not writable
is not readable
isnota directory
It is a normal file:false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Reading Data From Web

As many of you must be knowing that Uniform Resource Locator-URL is a string of text that identifies all the resources on Internet, telling us the address of the resource, how to communicate with it and retrieve something from it.

A Simple URL looks like:

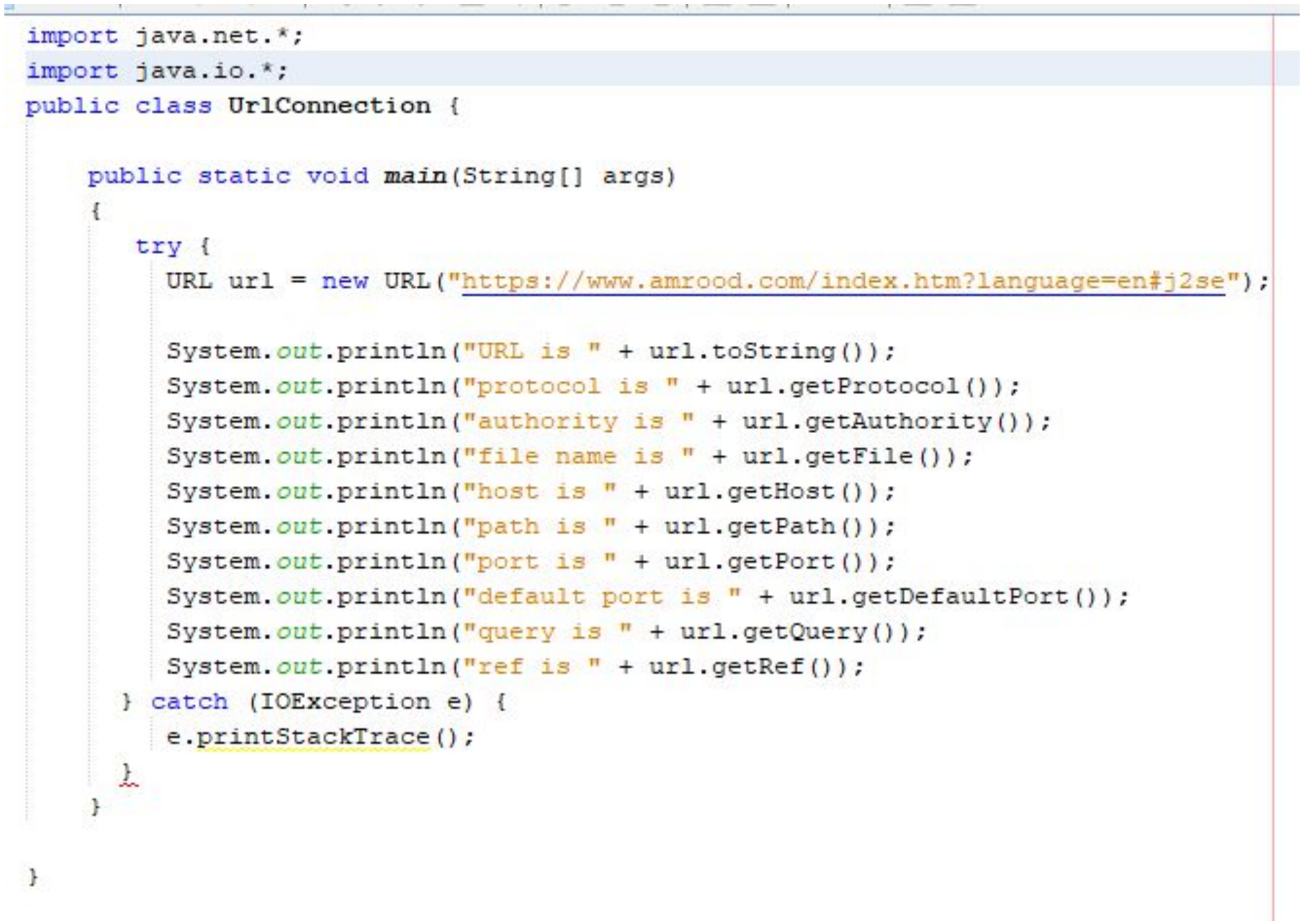

http://www.geeksforgeeks.org/how-to-design-a-tiny-url-or-url-shortener/
Protocol
Host Machine
File Name

Constructor in URL

- ➔ **URL(String protocol, String host, String file):** Creates a URL object from the specified protcol, host, and file name.
- ➔ **URL(String protocol, String host, int port, String file):** Creates a URL object from protocol, host, port and file name.
- ➔ **URL(URL context, String spec):** Creates a URL object by parsing the given spec in the given context.
- ➔ **URL(String protocol, String host, int port, String file, URLStreamHandler handler):-** Creates a URL object from the specified protocol, host, port number, file, and handler.
- ➔ **URL(URL context, String spec, URLStreamHandler handler):-** Creates a URL by parsing the given spec with the specified handler within a specified context.

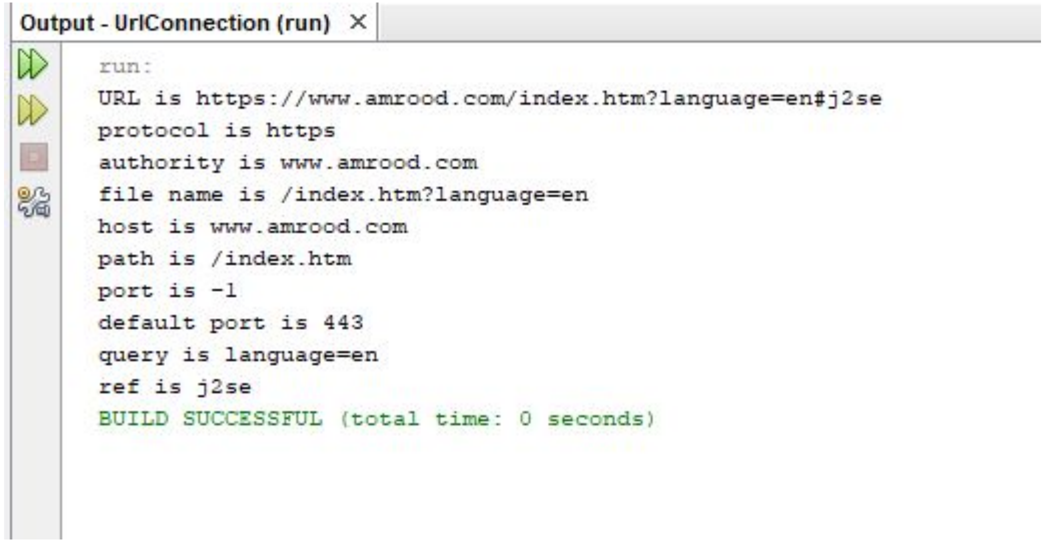
Methods in URL

- **public String getHost():** return the hostname of the URL in IPv6 format.
- **public String getFile():** returns the file name.
- **public int getPort():** returns the port associated with the protocol specified by the URL.
- **public int getDefaultPort:** returns the default port used.
- **public String getProtocol():** returns the protocol used by the URL.
- **public String toString():** As in any class, toString() returns the string representation of the given URL object.
- **public String getAuthority():** returns the authority part of URL or null if empty.
- **public String getPath():** returns the path of the URL, or null if empty.



```
import java.net.*;
import java.io.*;
public class UrlConnection {

    public static void main(String[] args)
    {
       try {
         URL url = new URL("https://www.amrood.com/index.htm?language=en#j2se");

         System.out.println("URL is " + url.toString());
         System.out.println("protocol is " + url.getProtocol());
         System.out.println("authority is " + url.getAuthority());
         System.out.println("file name is " + url.getFile());
         System.out.println("host is " + url.getHost());
         System.out.println("path is " + url.getPath());
         System.out.println("port is " + url.getPort());
         System.out.println("default port is " + url.getDefaultPort());
         System.out.println("query is " + url.getQuery());
         System.out.println("ref is " + url.getRef());
      } catch (IOException e) {
         e.printStackTrace();
      }
    }

}
```

output:


Output - UrlConnection (run)

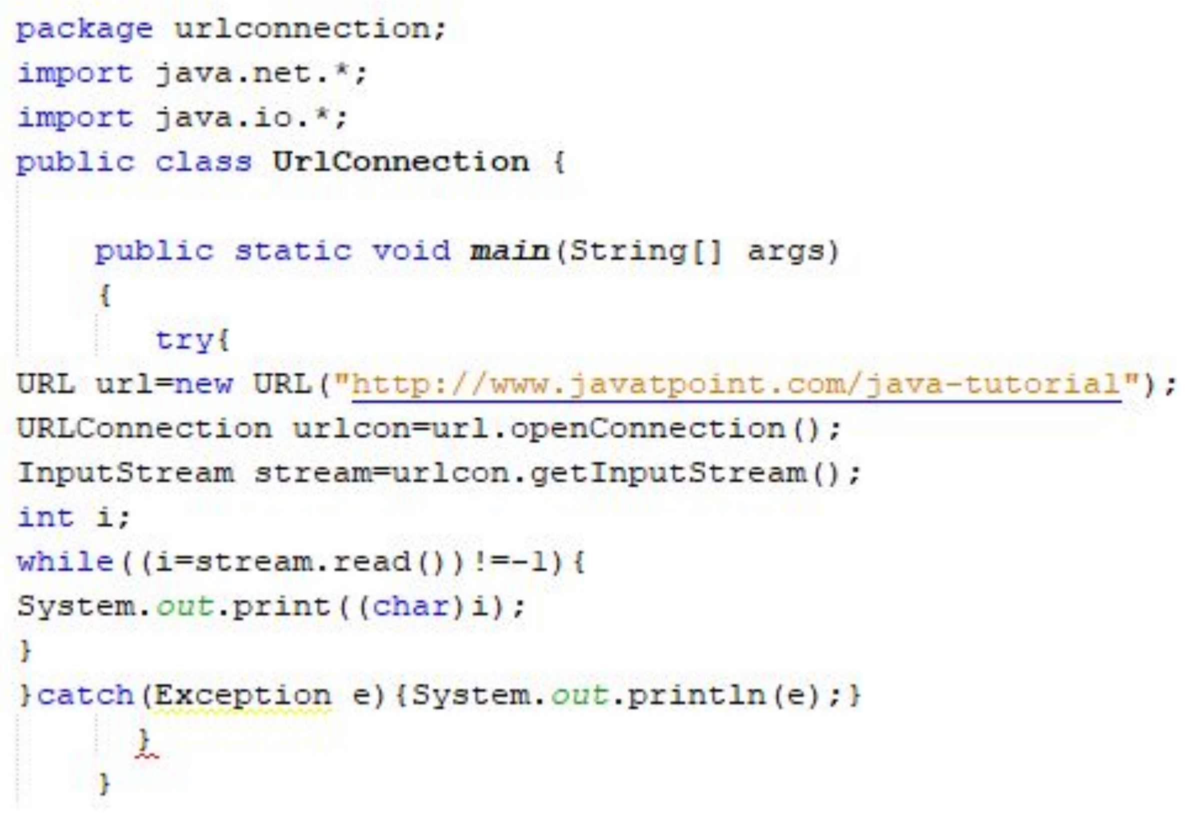
```
run:
URL is https://www.amrood.com/index.htm?language=en#j2se
protocol is https
authority is www.amrood.com
file name is /index.htm?language=en
host is www.amrood.com
path is /index.htm
port is -1
default port is 443
query is language=en
ref is j2se
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

URL Connection Class

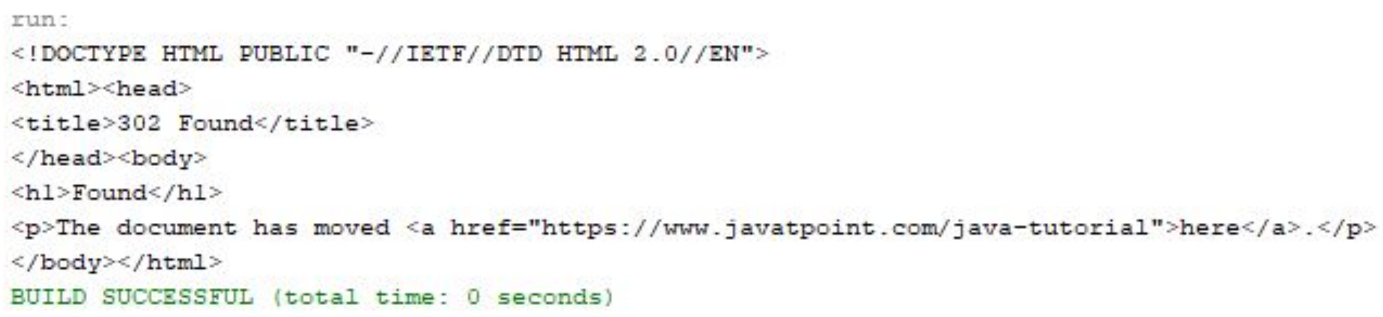
- ➔ The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.
- ➔ How to get the object of URLConnection class
- ➔ The openConnection() method of URL class returns the object of URLConnection class. Syntax:

```
public URLConnection openConnection()throws IOException{}
```



```
package urlconnection;
import java.net.*;
import java.io.*;
public class UrlConnection {

    public static void main(String[] args)
    {
        try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon=url.openConnection();
InputStream stream=urlcon.getInputStream();
int i;
while((i=stream.read())!=-1){
System.out.print((char)i);
}
}catch(Exception e){System.out.println(e);}
        }
    }
```

output:



```
run:
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://www.javatpoint.com/java-tutorial">here</a>.</p>
</body></html>
BUILD SUCCESSFUL (total time: 0 seconds)
```

ABSTRACT CLASSES

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

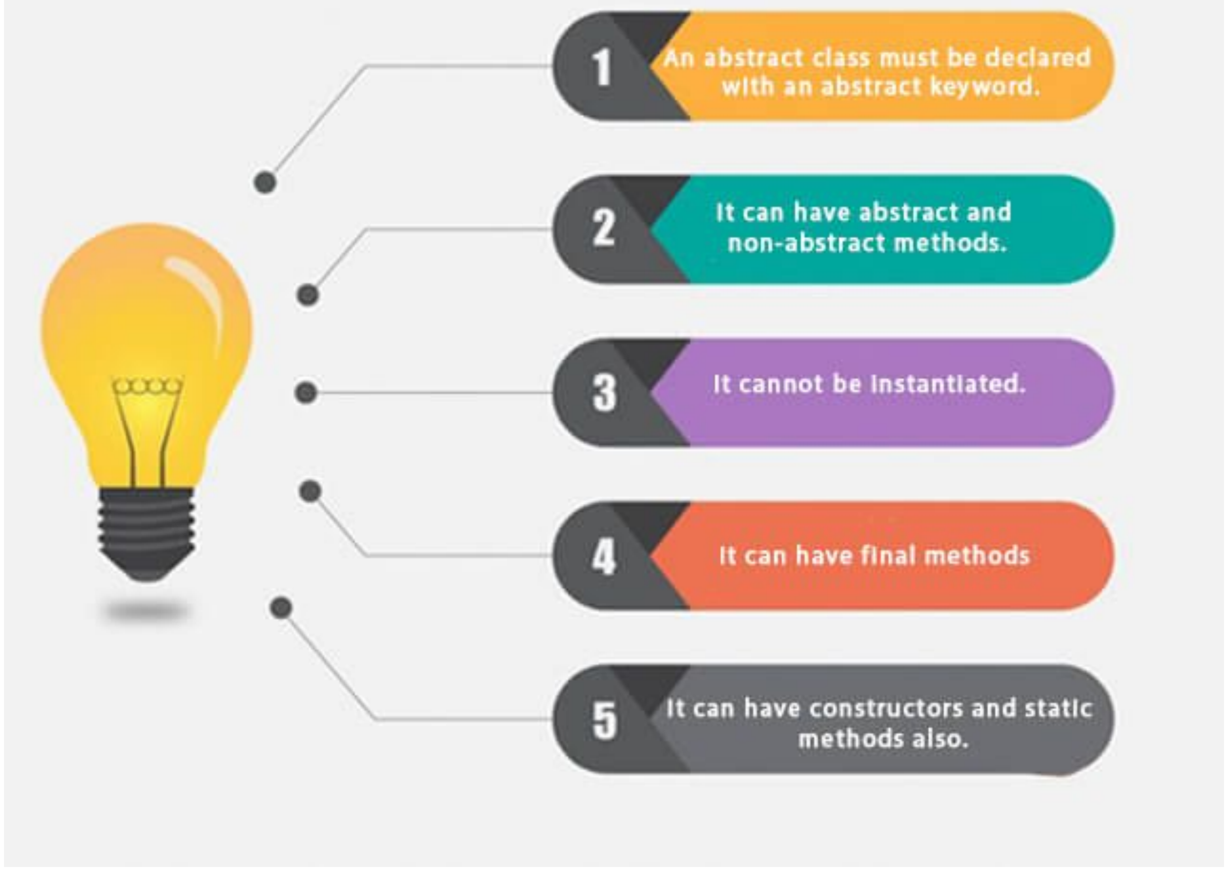
Abstract Classes:

- ➔ A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember:

- ➔ An abstract class must be declared with an abstract keyword.
- ➔ It can have abstract and non-abstract methods.
- ➔ It cannot be instantiated.
- ➔ It can have constructors and static methods also.
- ➔ It can have final methods which will force the subclass not to change the body of the method.

Rules for Java Abstract class



1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

Example of abstract class

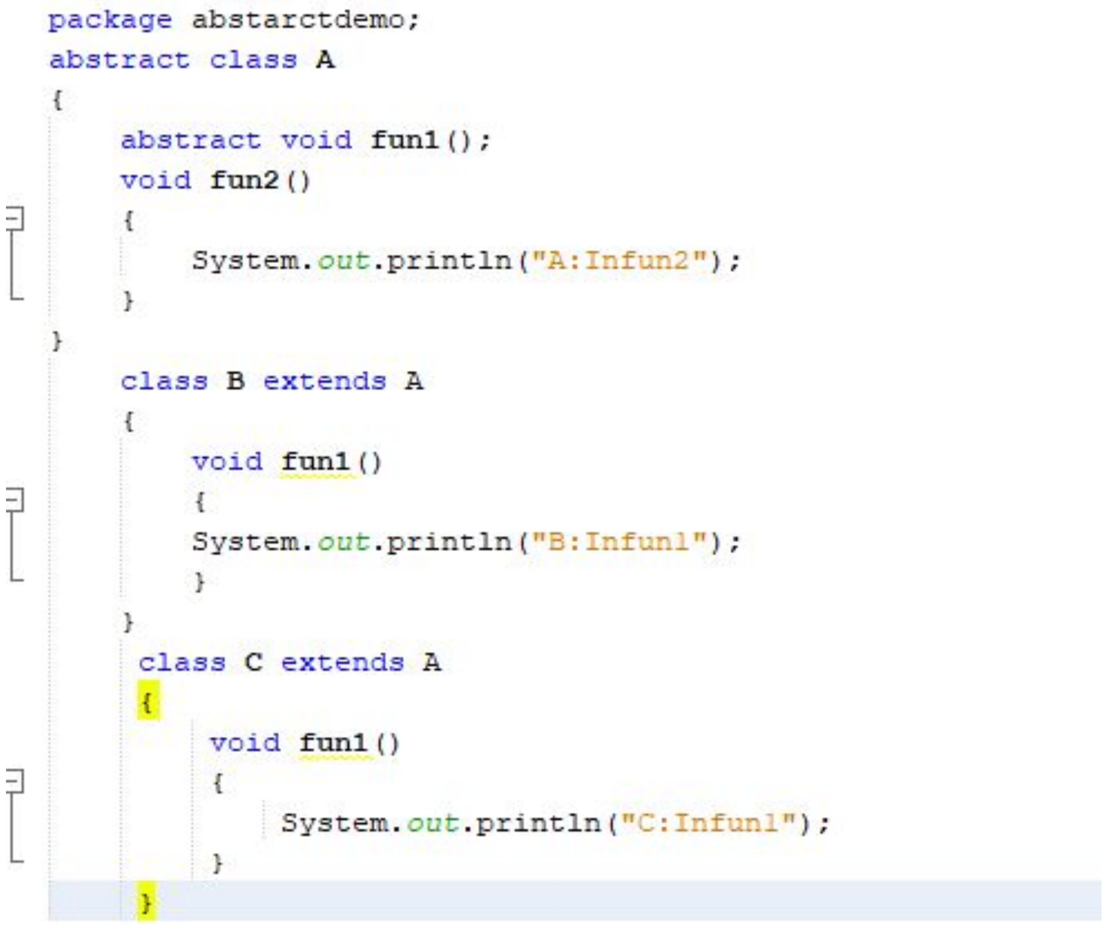
abstract class A{}

Abstract Method in Java

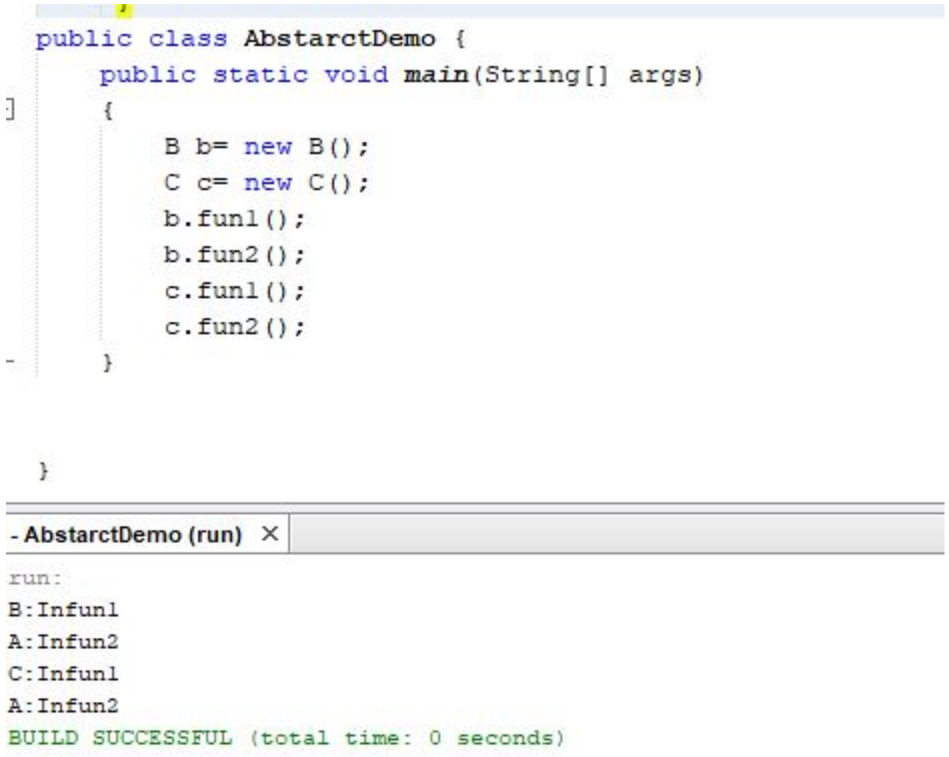
A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

abstract void printStatus();



```java
package abstarctdemo;
abstract class A
{
    abstract void fun1();
    void fun2()
    {
        System.out.println("A:Infun2");
    }
}
    class B extends A
    {
        void fun1()
        {
        System.out.println("B:Infun1");
        }
    }
     class C extends A
     {
         void fun1()
         {
             System.out.println("C:Infun1");
         }
     }
```

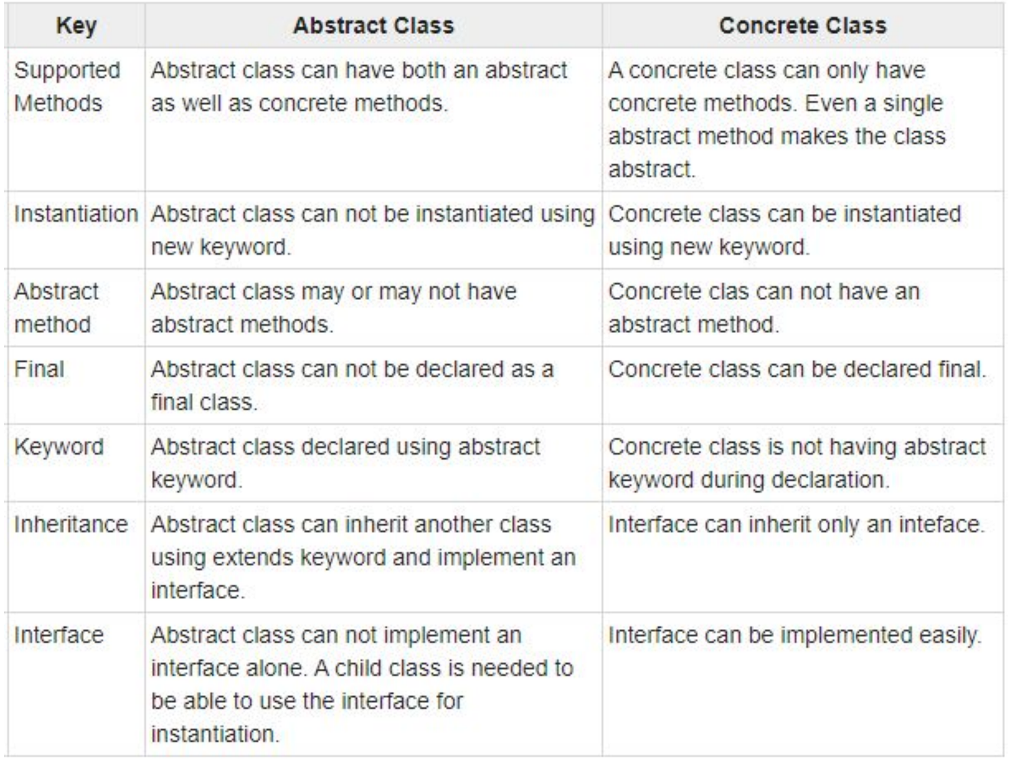


```
public class AbstarctDemo {
    public static void main(String[] args)
    {
        B b= new B();
        C c= new C();
        b.fun1();
        b.fun2();
        c.fun1();
        c.fun2();
    }


}
```

- AbstarctDemo (run)

```
run:
B:Infun1
A:Infun2
C:Infun1
A:Infun2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Abstact Class v/s Concrete Class



Key	Abstract Class	Concrete Class
Supported Methods	Abstract class can have both an abstract as well as concrete methods.	A concrete class can only have concrete methods. Even a single abstract method makes the class abstract.
Instantiation	Abstract class can not be instantiated using new keyword.	Concrete class can be instantiated using new keyword.
Abstract method	Abstract class may or may not have abstract methods.	Concrete clas can not have an abstract method.
Final	Abstract class can not be declared as a final class.	Concrete class can be declared final.
Keyword	Abstract class declared using abstract keyword.	Concrete class is not having abstract keyword during declaration.
Inheritance	Abstract class can inherit another class using extends keyword and implement an interface.	Interface can inherit only an inteface.
Interface	Abstract class can not implement an interface alone. A child class is needed to be able to use the interface for instantiation.	Interface can be implemented easily.

Interfaces

Interfaces

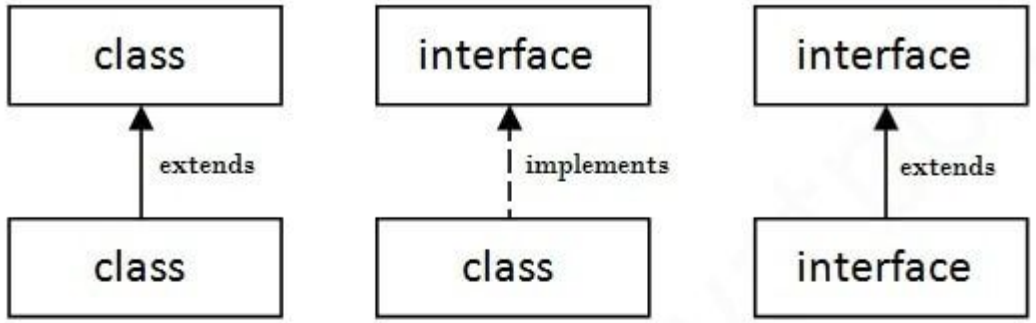
The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

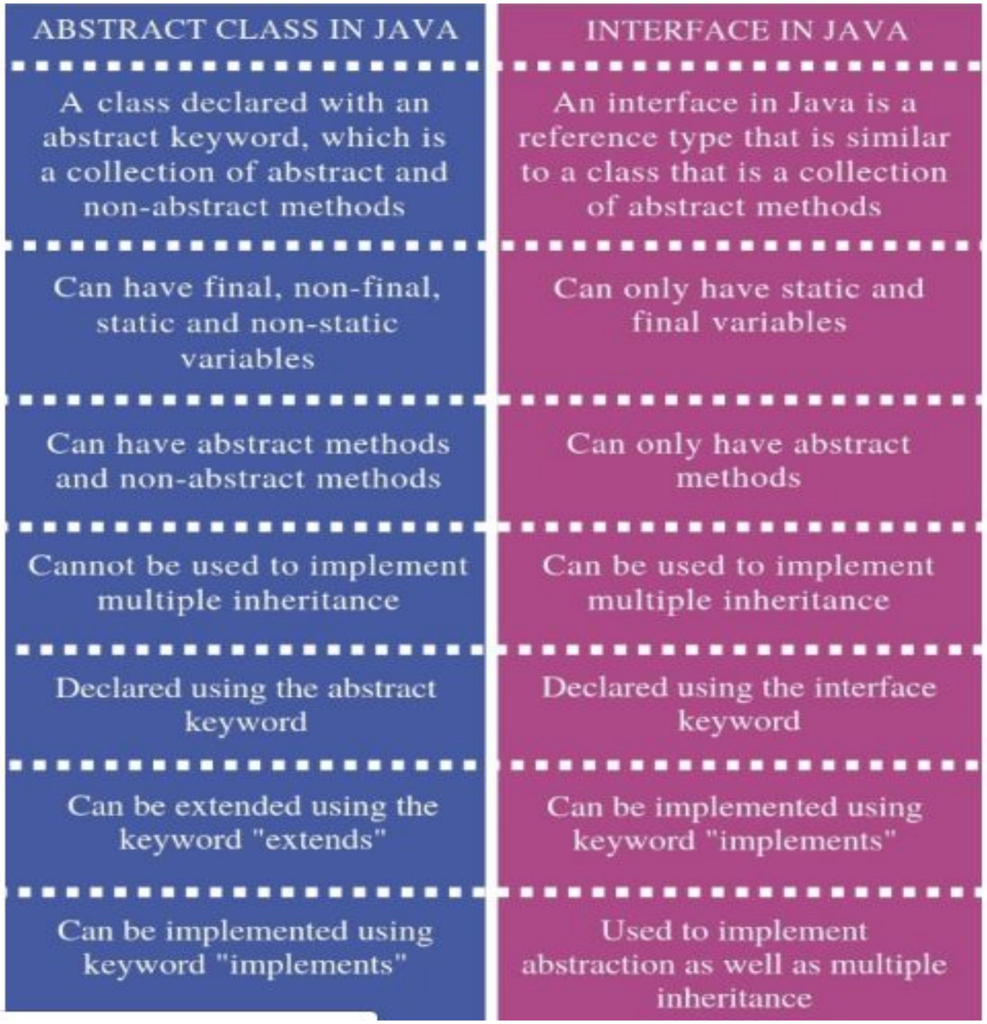
Syntax:

```
interface <interface_name>{

    // declare constant fields

    // declare methods that abstract

    // by default.

public void method1(); /

   public void method2();

}
```

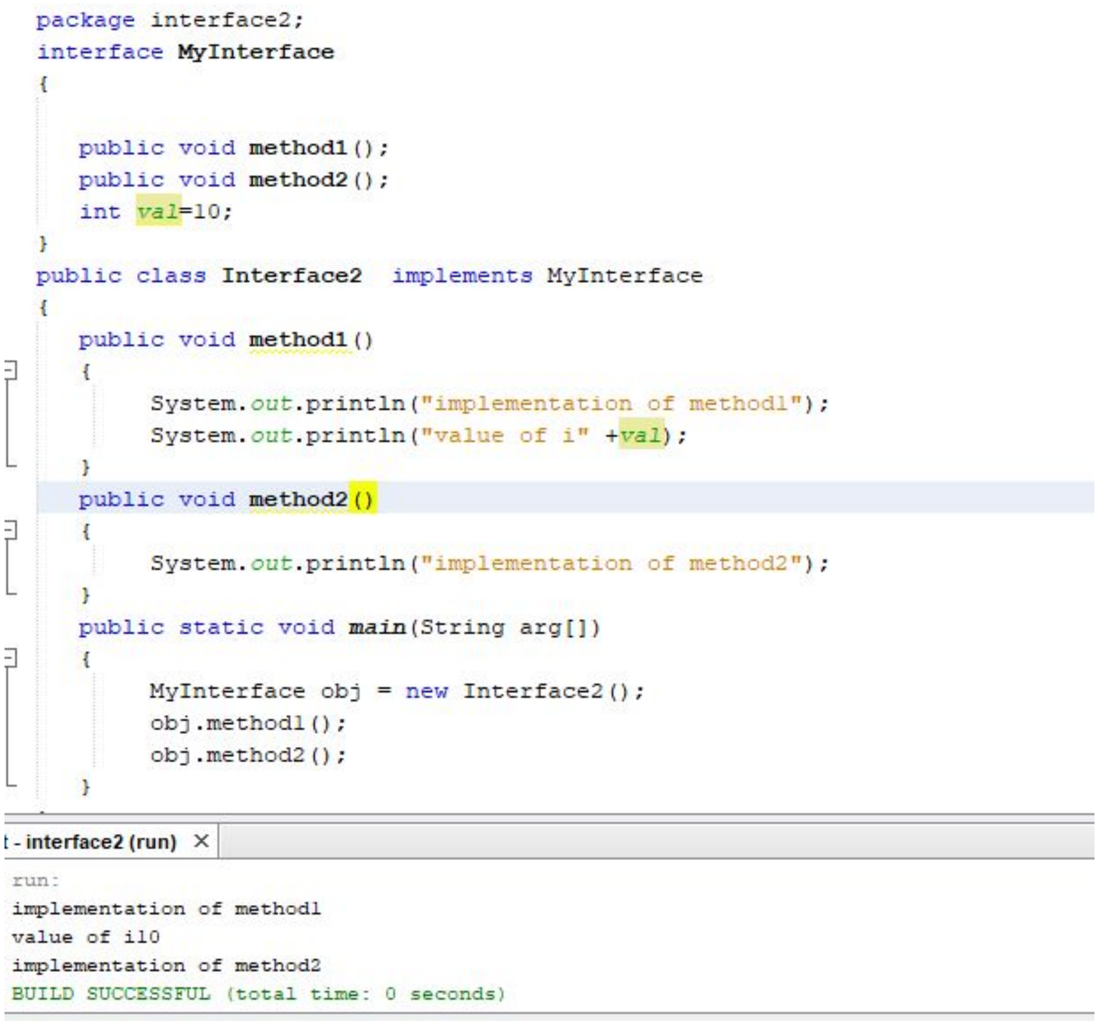


syntax:

```
acess_modifier interface name_of_interface

{

return_type method_name1(parameter1,parameter2,......,parametern);

return_type method_name2(parameter1,parameter2,......,parametern);

type static final variable_name=value;

}
```



ABSTRACT CLASS IN JAVA	INTERFACE IN JAVA
A class declared with an abstract keyword, which is a collection of abstract and non-abstract methods	An interface in Java is a reference type that is similar to a class that is a collection of abstract methods
Can have final, non-final, static and non-static variables	Can only have static and final variables
Can have abstract methods and non-abstract methods	Can only have abstract methods
Cannot be used to implement multiple inheritance	Can be used to implement multiple inheritance
Declared using the abstract keyword	Declared using the interface keyword
Can be extended using the keyword "extends"	Can be implemented using keyword "implements"
Can be implemented using keyword "implements"	Used to implement abstraction as well as multiple inheritance

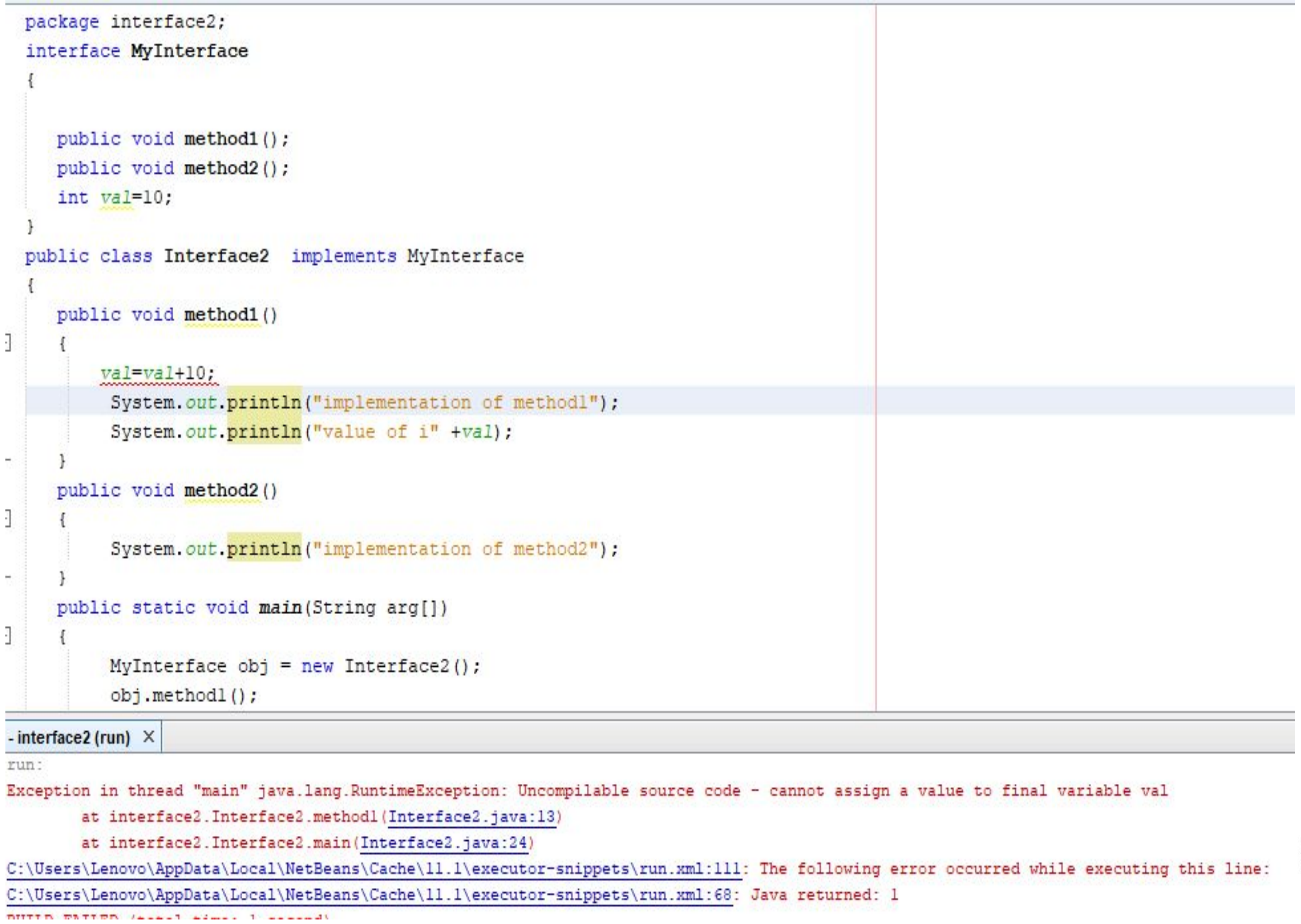


```java
package interface2;
interface MyInterface
{

    public void method1();
    public void method2();
    int val=10;
}
public class Interface2  implements MyInterface
{
    public void method1()
    {
        System.out.println("implementation of method1");
        System.out.println("value of i" +val);
    }
    public void method2()
    {
        System.out.println("implementation of method2");
    }
    public static void main(String arg[])
    {
        MyInterface obj = new Interface2();
        obj.method1();
        obj.method2();
    }
```

t - interface2 (run)

```
run:
implementation of method1
value of i10
implementation of method2
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```java
package interface2;
interface MyInterface
{

   public void method1();
   public void method2();
   int val=10;
}
public class Interface2  implements MyInterface
{
   public void method1()
   {
       val=val+10;
        System.out.println("implementation of method1");
        System.out.println("value of i" +val);
   }
   public void method2()
   {
        System.out.println("implementation of method2");
   }
   public static void main(String arg[])
   {
        MyInterface obj = new Interface2();
        obj.method1();
```

- interface2 (run) ×

```
run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - cannot assign a value to final variable val
        at interface2.Interface2.method1(Interface2.java:13)
        at interface2.Interface2.main(Interface2.java:24)
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\Lenovo\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 1 second)
```

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Syntax:

```
Interface Interface_name2 extends Interface_name1

{

//body of interface

}
```

Example:

Interface A

{

Int val=10;

}

Interface B extends A

{

Void print_val();

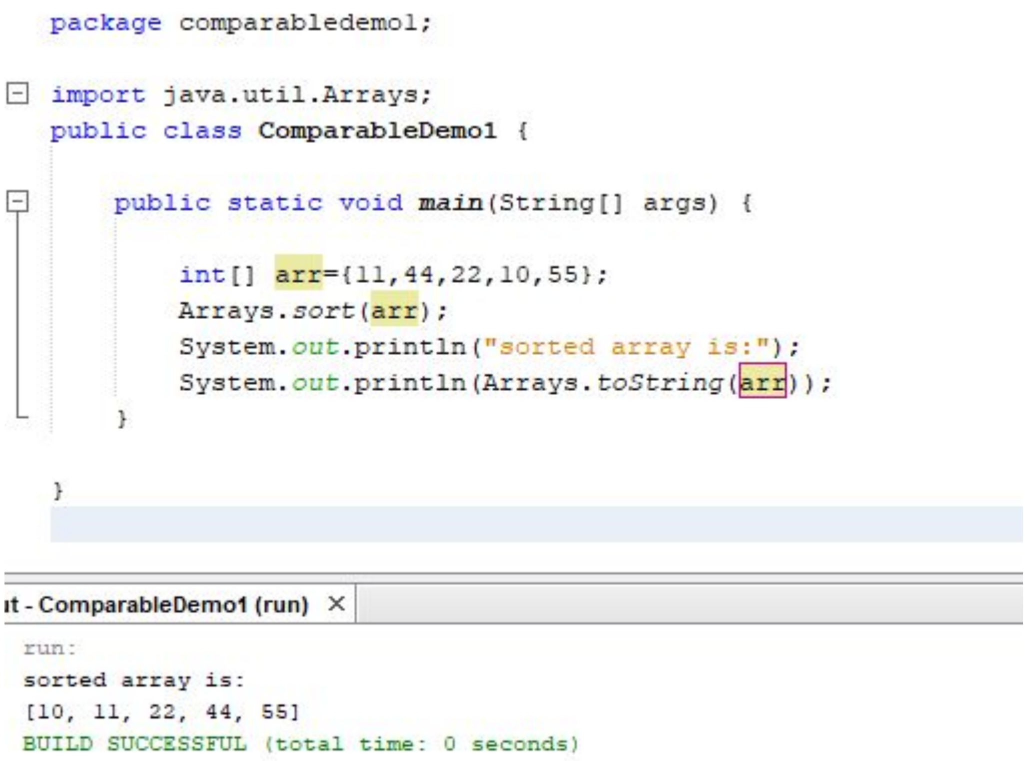
}

```
Interface A
{
Int val=10;
}
Interface B extends A
{
Void print_val();
}
interfaces C  extends A,B{
…...
}
```

Comparable Interface

- ➔ Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.
- ➔ If we use Arrays and List objects then these objects can be sorted automatically by Collection.sort method which is basically implements comparable interfaces.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



```
package comparabledemo1;

import java.util.Arrays;
public class ComparableDemo1 {

    public static void main(String[] args) {

        int[] arr={11,44,22,10,55};
        Arrays.sort(arr);
        System.out.println("sorted array is:");
        System.out.println(Arrays.toString(arr));
    }

}
```

ut - ComparableDemo1 (run)

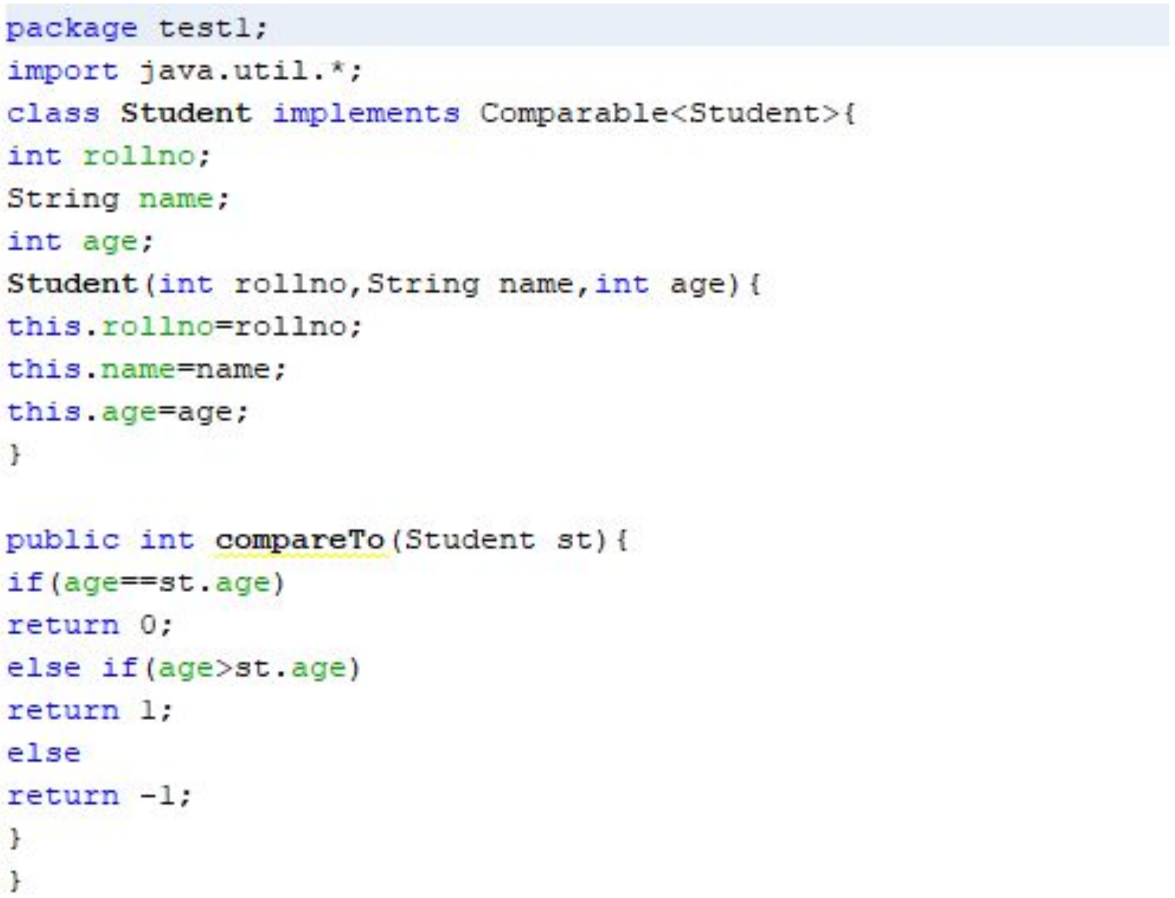
```
run:
sorted array is:
[10, 11, 22, 44, 55]
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

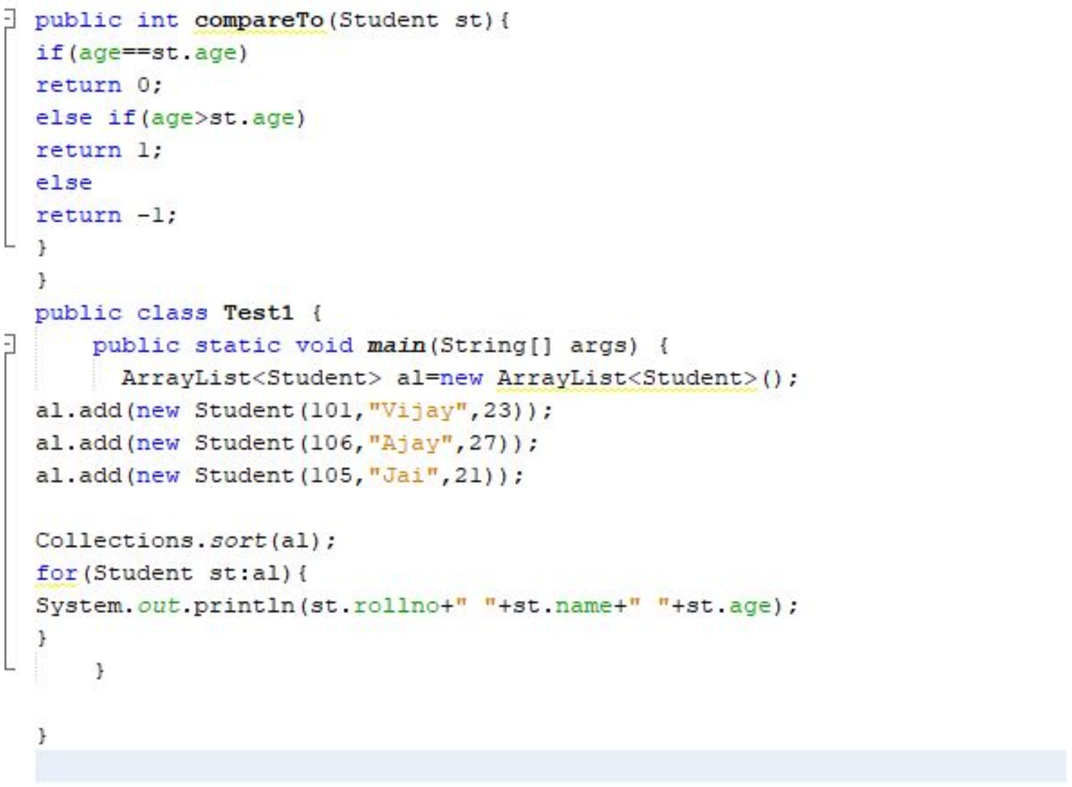
compareTo(Object obj) method

public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns

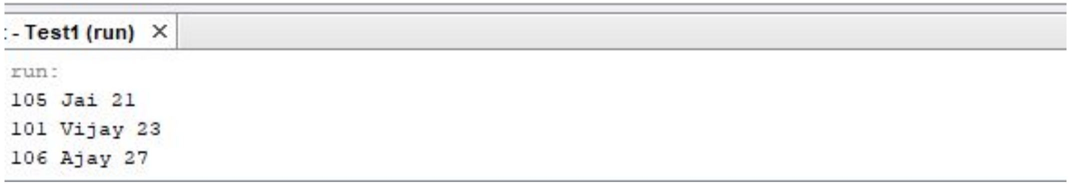
- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.



```java
package test1;
import java.util.*;
class Student implements Comparable<Student>{
int rollno;
String name;
int age;
Student(int rollno,String name,int age){
this.rollno=rollno;
this.name=name;
this.age=age;
}

public int compareTo(Student st){
if(age==st.age)
return 0;
else if(age>st.age)
return 1;
else
return -1;
}
}
```



```
public int compareTo(Student st){
if(age==st.age)
return 0;
else if(age>st.age)
return 1;
else
return -1;
}
}
public class Test1 {
    public static void main(String[] args) {
      ArrayList<Student> al=new ArrayList<Student>();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

Collections.sort(al);
for(Student st:al){
System.out.println(st.rollno+" "+st.name+" "+st.age);
}
    }

}
```



```
run:
105 Jai 21
101 Vijay 23
106 Ajay 27
```

Cloneable Interface

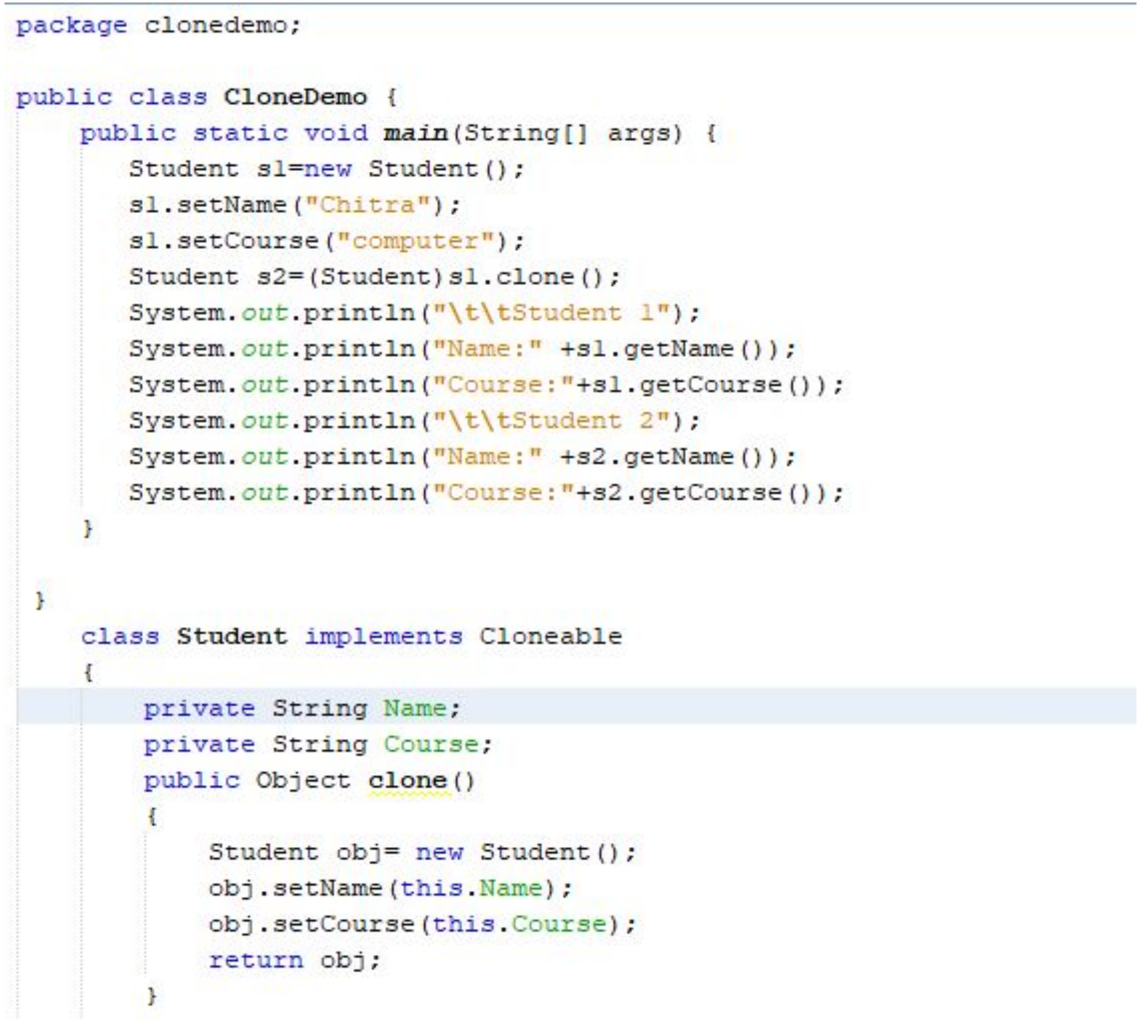
- ➔ The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.
- ➔ The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

What if we don’t implement Cloneable interface?

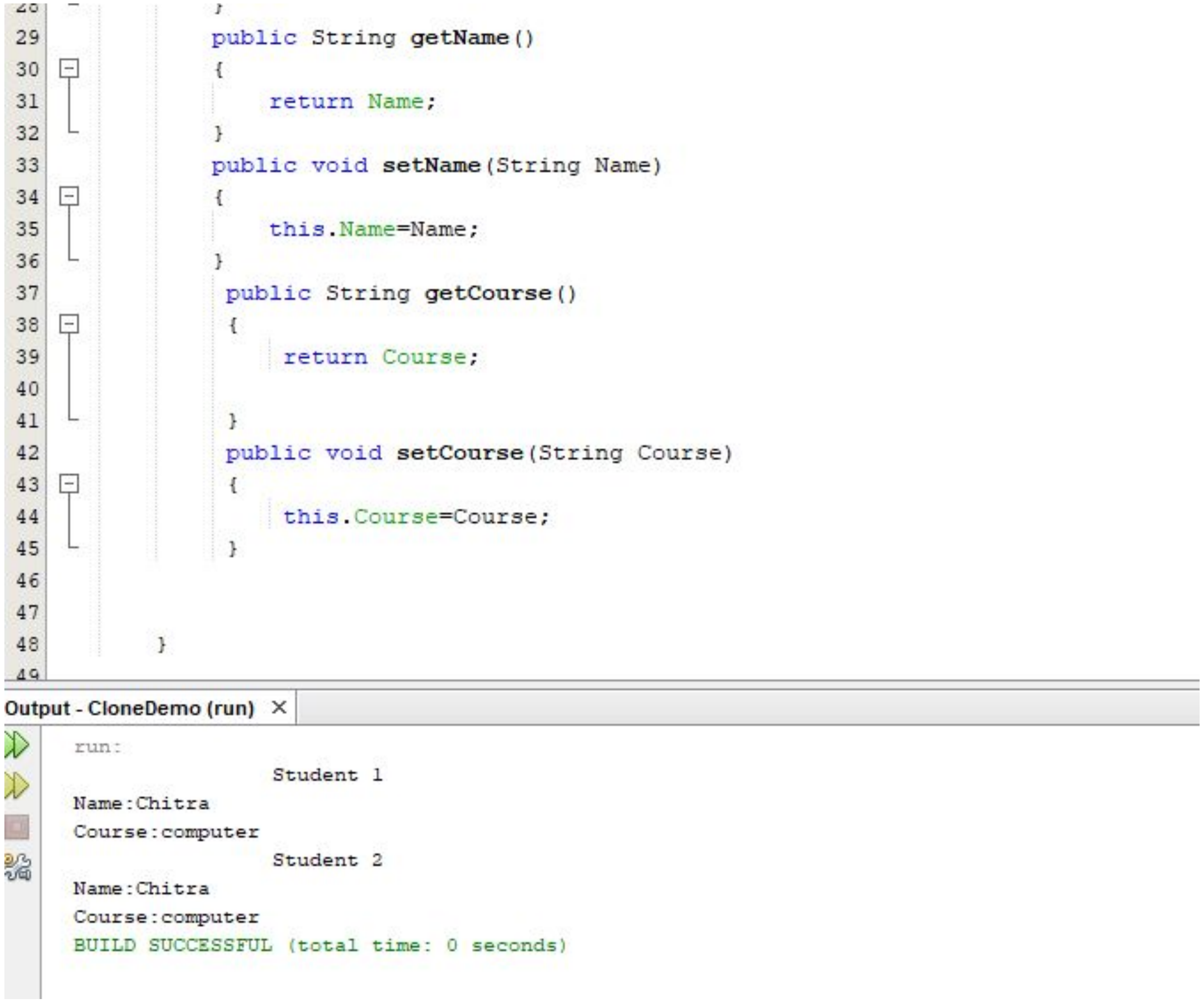
The program would throw CloneNotSupportedException if we don’t implement the Cloneable interface. A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class.

Does clone object and original object point to the same location in memory

The answer is no. The clone object has its own space in the memory where it copies the content of the original object. That’s why when we change the content of original object after cloning, the changes does not reflect in the clone object. Lets see this with the help of an example.



```java
package clonedemo;

public class CloneDemo {
    public static void main(String[] args) {
        Student s1=new Student();
        s1.setName("Chitra");
        s1.setCourse("computer");
        Student s2=(Student)s1.clone();
        System.out.println("\t\tStudent 1");
        System.out.println("Name:" +s1.getName());
        System.out.println("Course:"+s1.getCourse());
        System.out.println("\t\tStudent 2");
        System.out.println("Name:" +s2.getName());
        System.out.println("Course:"+s2.getCourse());
    }


 }
    class Student implements Cloneable
    {
        private String Name;
        private String Course;
        public Object clone()
        {
            Student obj= new Student();
            obj.setName(this.Name);
            obj.setCourse(this.Course);
            return obj;
        }
```



```
        }
        public String getName()
        {
            return Name;
        }
        public void setName(String Name)
        {
            this.Name=Name;
        }
         public String getCourse()
         {
             return Course;

         }
         public void setCourse(String Course)
         {
             this.Course=Course;
         }


    }
```

Output - CloneDemo (run)

```
run:
                  Student 1
Name:Chitra
Course:computer
                  Student 2
Name:Chitra
Course:computer
BUILD SUCCESSFUL (total time: 0 seconds)
```

VIDEO LINKS FOR I/O,ABSTRACT CLASS AND INTERFACES

Example of URL Class, How to Parse URL in Java
https://www.youtube.com/watch?v=DuFyhu5_GPs

Java URLConnection Class – How to Read/Write File on URL
https://www.youtube.com/watch?v=5-2kNYxWxOg

Abstract Class and Abstract Methods in Java
https://www.youtube.com/watch?v=TZa9omZ-uTg

Interface in Java with Example, Multiple Inheritance in Java using Interface
https://www.youtube.com/watch?v=1MDf8p-Tkk0

Java Comparable interface
https://www.youtube.com/watch?v=swEvHhN9l8k

Cloneable Interface

https://www.youtube.com/watch?v=b2uFL4BFDYg

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY