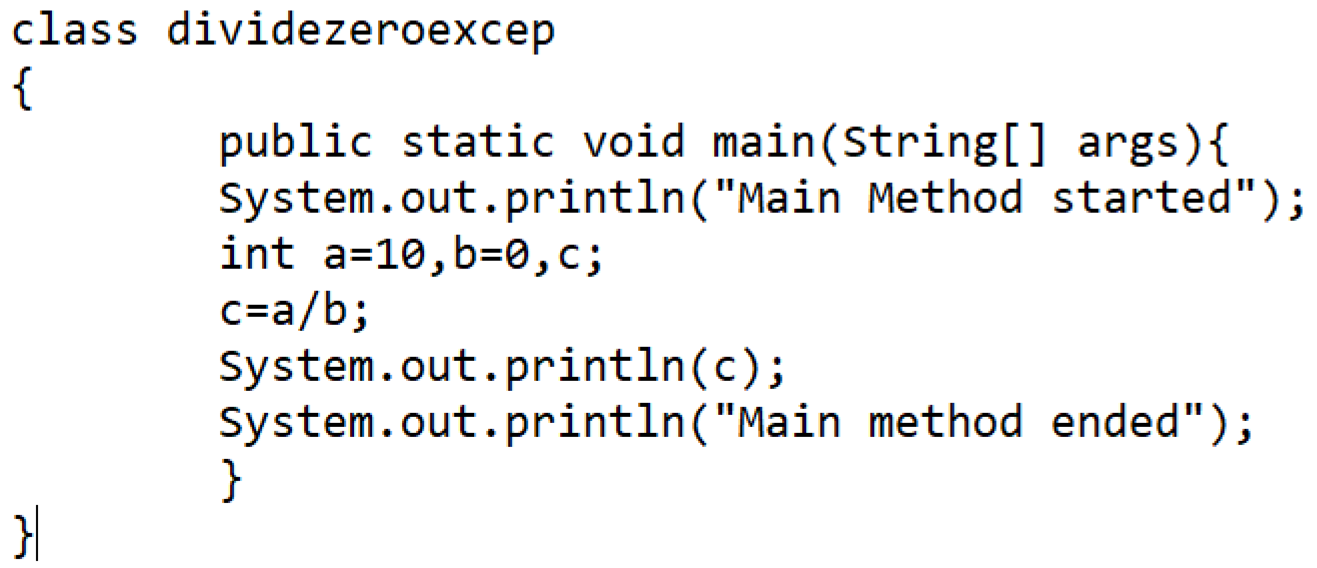
Exception Handling

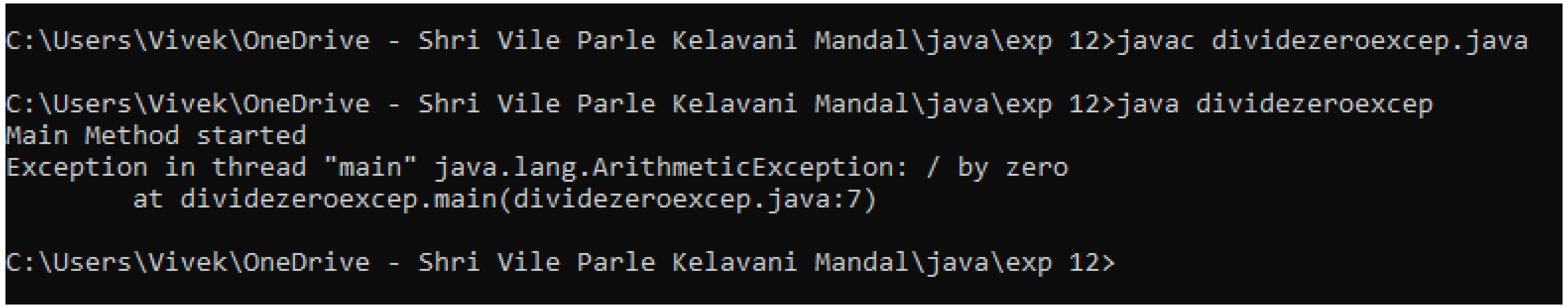
What is Exception?

- A situation which is Unexpected/Unwanted/abnormal that occurred during runtime.
- Reasons of Exceptions

1. Invalid user input
2. code errors
3. Device failure
4. The loss of a network connection,
5. Insufficient memory to run an application,
6. A memory conflict with another program,
7. A program attempting to divide by zero or a user attempting to open files that are unavailable.



```
class dividezeroexcep
{
        public static void main(String[] args){
        System.out.println("Main Method started");
        int a=10,b=0,c;
        c=a/b;
        System.out.println(c);
        System.out.println("Main method ended");
        }
}
```



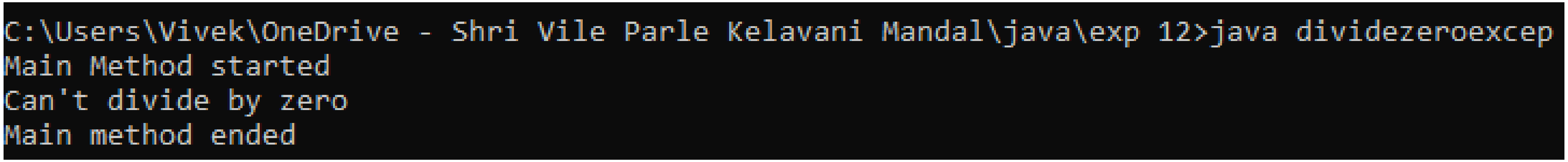
```
C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>javac dividezeroexcep.java

C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>java dividezeroexcep
Main Method started
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at dividezeroexcep.main(dividezeroexcep.java:7)

C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>
```

Handling

- Exception handling is done by writing code when such exception occurs.
- The core advantage of exception handling is **to maintain the normal flow of the application**.

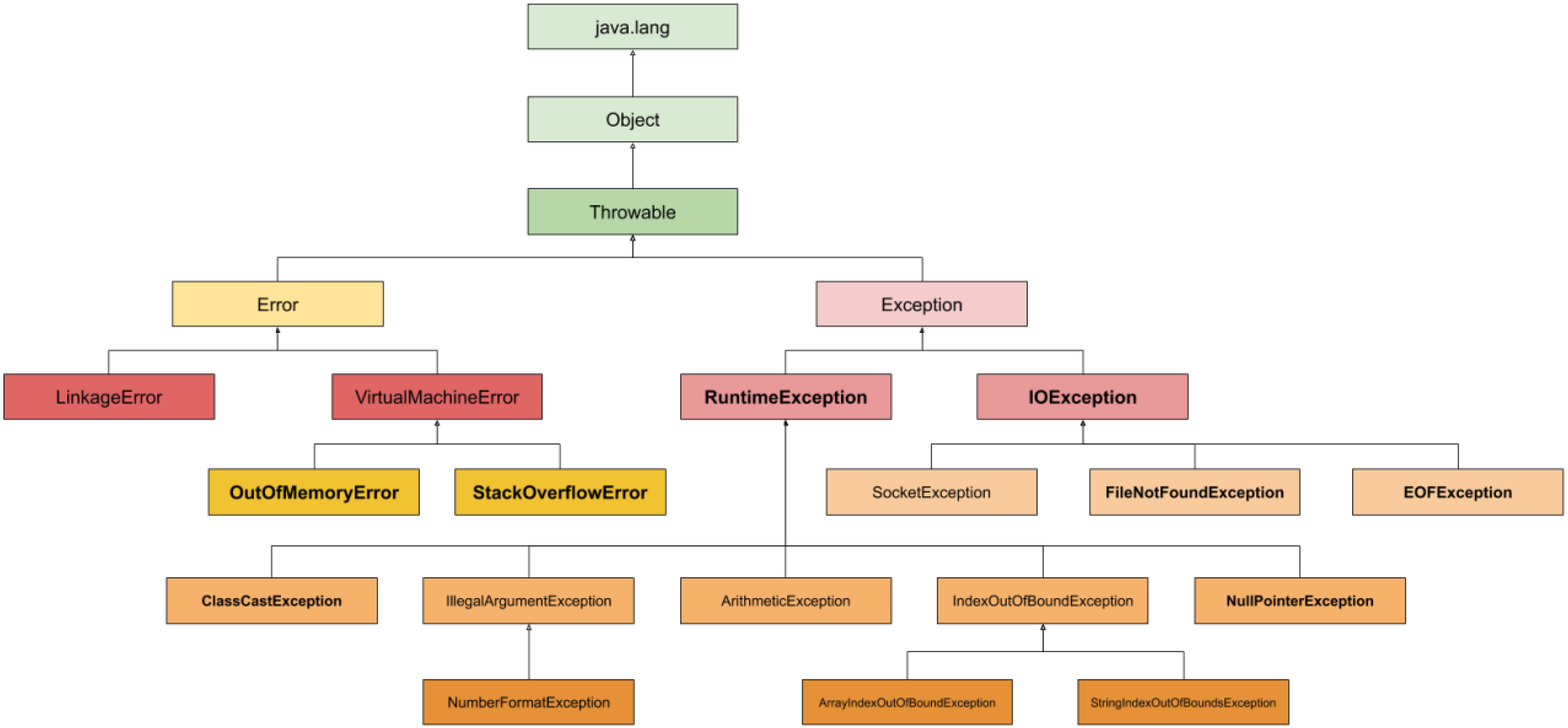
PPT is for reference Only, refer to Text Book for Complete Explanations

```
class dividezeroexcep
{
        public static void main(String[] args){
        System.out.println("Main Method started");
        int a=10,b=0,c;
        try{
        c=a/b;
        System.out.println(c);
        }
        catch(Exception e)
        {
                System.out.println("Can't divide by zero");
        }
                System.out.println("Main method ended");
        }
}
```



```
C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>java dividezeroexcep
Main Method started
Can't divide by zero
Main method ended
```

```
class dividezeroexcep
{
        public static void main(String[] args){
        System.out.println("Main Method started");
        int a=10,b=0,c;
        try{
        c=a/b;
        System.out.println(c);
        }
        catch(ArithmeticException e)
        {
                System.out.println("Can't divide by zero");
        }
                System.out.println("Main method ended");
        }
}
```


java.lang
Object
Throwable
Error
Exception
LinkageError
VirtualMachineError
RuntimeException
IOException
OutOfMemoryError
StackOverflowError
SocketException
FileNotFoundException
EOFException
ClassCastException
IllegalArgumentException
ArithmeticException
IndexOutOfBoundException
NullPointerException
NumberFormatException
ArrayIndexOutOfBoundException
StringIndexOutOfBoundsException

Difference between Error and Exception

1. Error

- It cannot and should not be handled by the developer.
- It usually tends to signal the end of your program
- It signals that the program cannot recover and causes you to exit the program altogether instead of trying to handle it.

PPT is for reference Only, refer to Text Book for Complete Explanations

Difference between Error and Exception Cont..

2. Exceptions

- It can and should be handled by the developer.
- Otherwise, they will surely lead to abnormal termination of the program.
- They are objects of a type ‘System.Exception’ class and arise when non-fatal and recoverable errors occur during run-time

Types of Exceptions

1. **Checked Exceptions:**

- Arise during compilation time.
- The compiler checks for these exceptions and whether the programmer has handled them.
- If not handle checked exceptions, then, it causes a compilation error, and the program will not compile.
- Examples: classNotFound Exception, SQL Exception, IO Exception, etc.

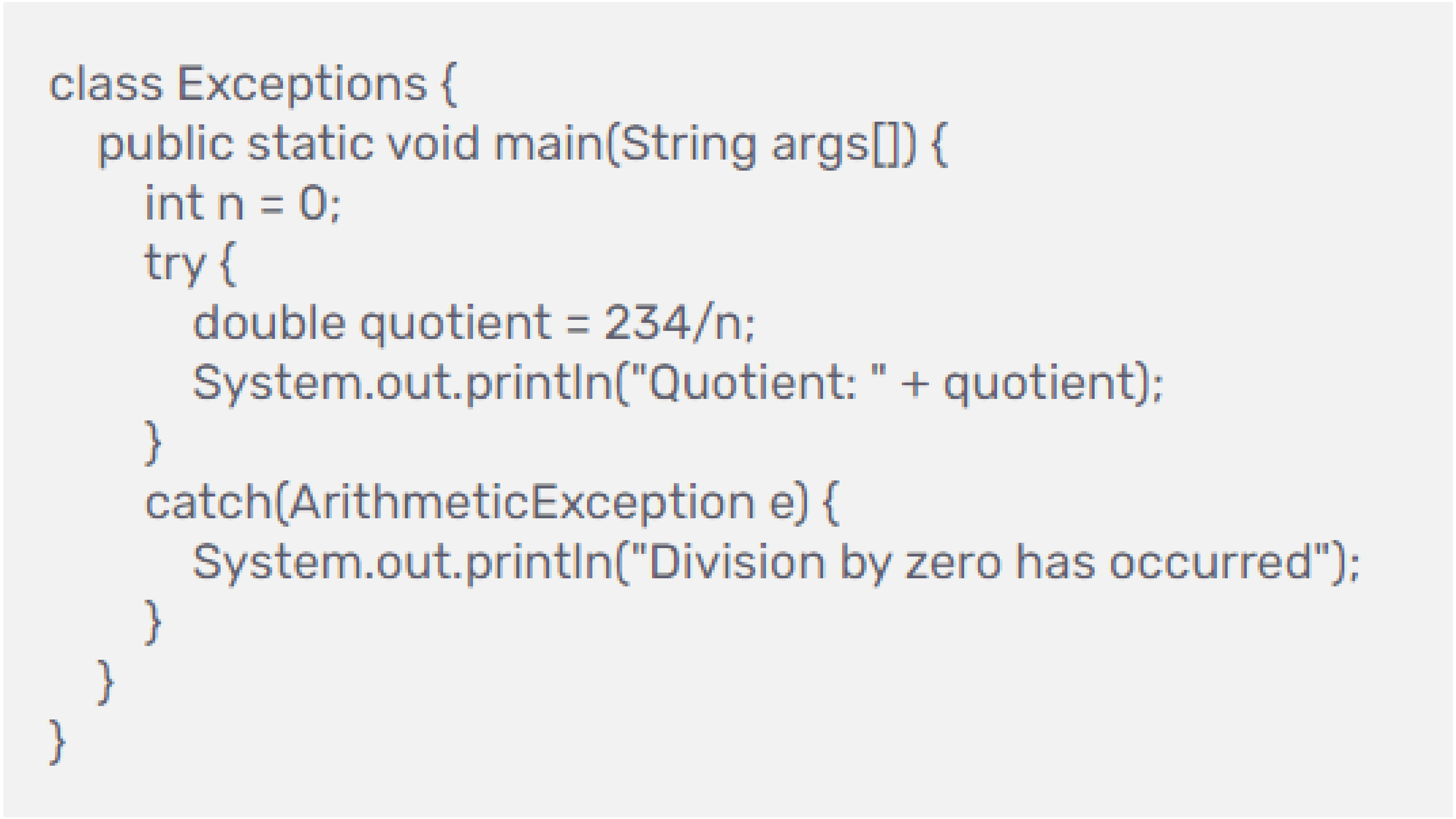
Keywords

- **The Try Block**
- The try block contains those lines which can cause an exception and can have one or more legal lines of code.
- If an exception occurs at a particular statement of the try block, the rest of the statements will not be executed.
- It is recommended to exclude those lines which won't throw an exception.

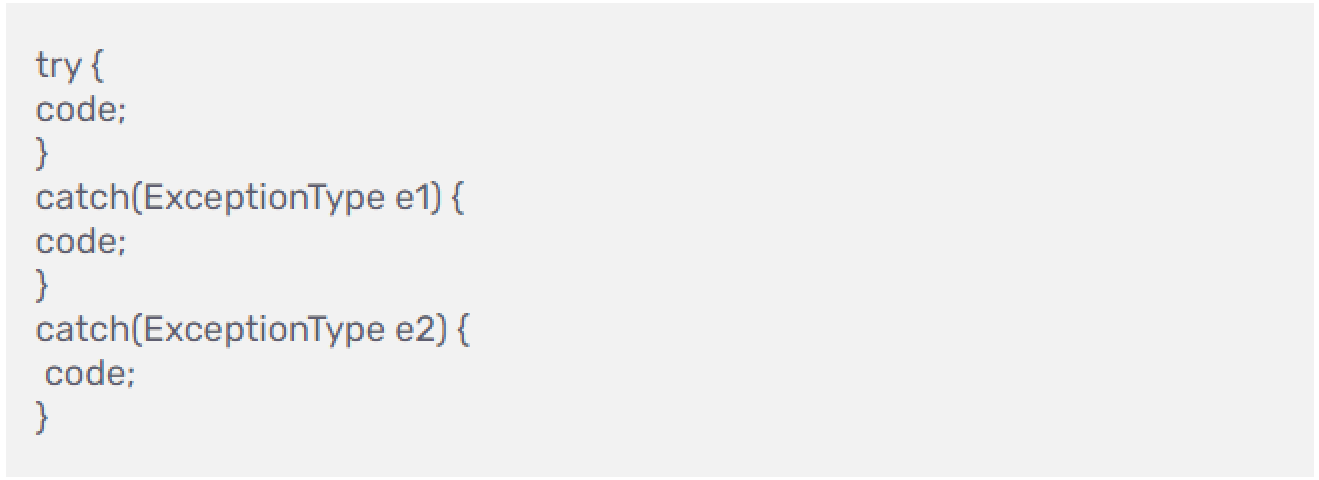
```
try {
 Block of code;
}
catch and finally blocks...
```

Keywords

- **The Catch Block**
- The catch block handles the exception.
- It contains lines of code that will be executed if an exception occurs.
- Each catch block is an exception handler that handles the type of exception specified by the argument.
- The argument type *ExceptionType* declares the type of exception that the handler can handle and must be the name of a class that inherits from the Throwable class.
- The Throwable class is the super-class of all the errors and exceptions in Java.



```
class Exceptions {
  public static void main(String args[]) {
    int n = 0;
    try {
      double quotient = 234/n;
      System.out.println("Quotient: " + quotient);
    }
    catch(ArithmeticException e) {
      System.out.println("Division by zero has occurred");
    }
  }
}
```



```
try {
code;
}
catch(ExceptionType e1) {
code;
}
catch(ExceptionType e2) {
 code;
}
```

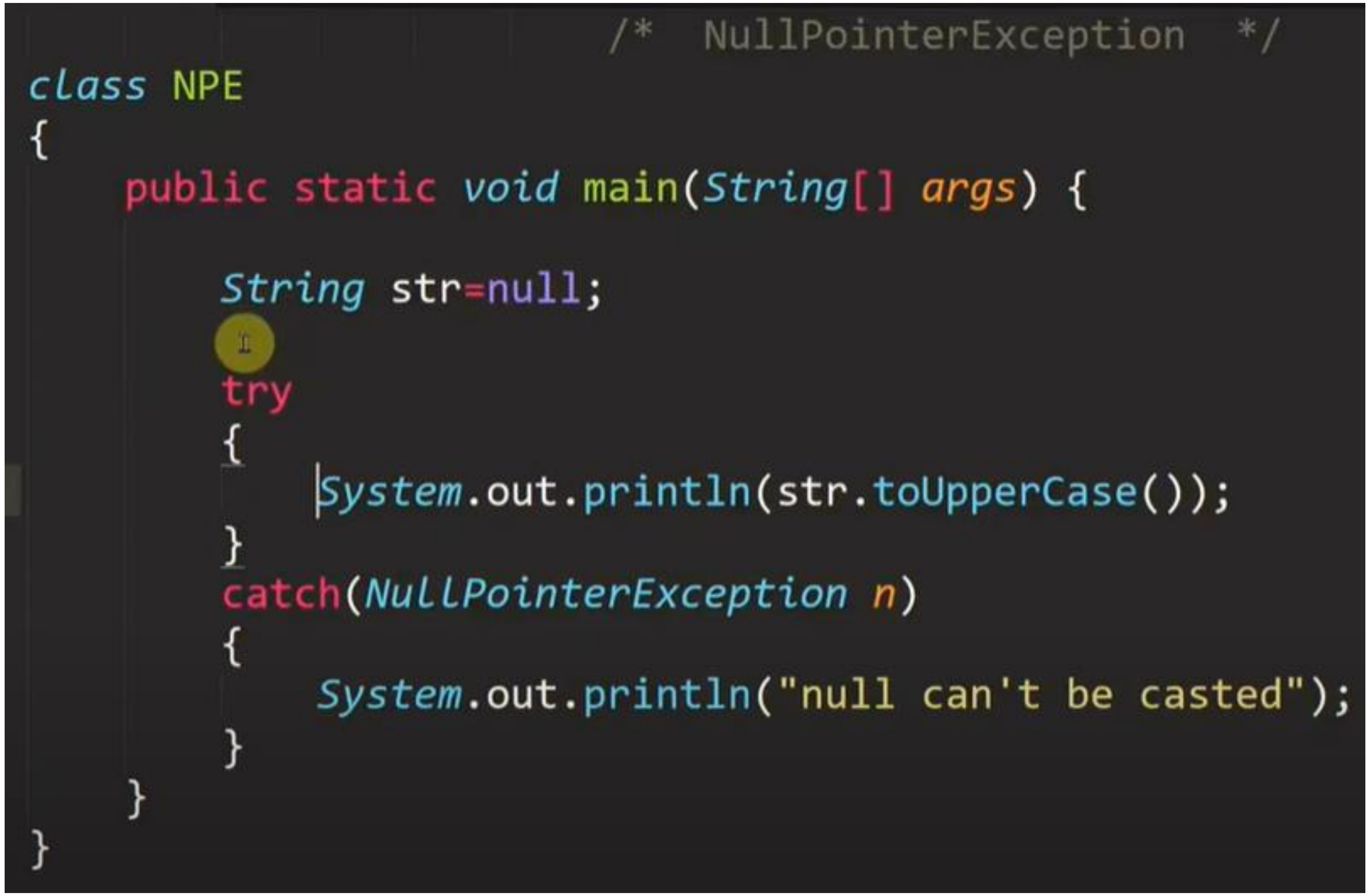
the class ArithmeticException is an instance of a Throwable class and is thrown by the JVM to handle the arithmetic exceptions.

Important Points

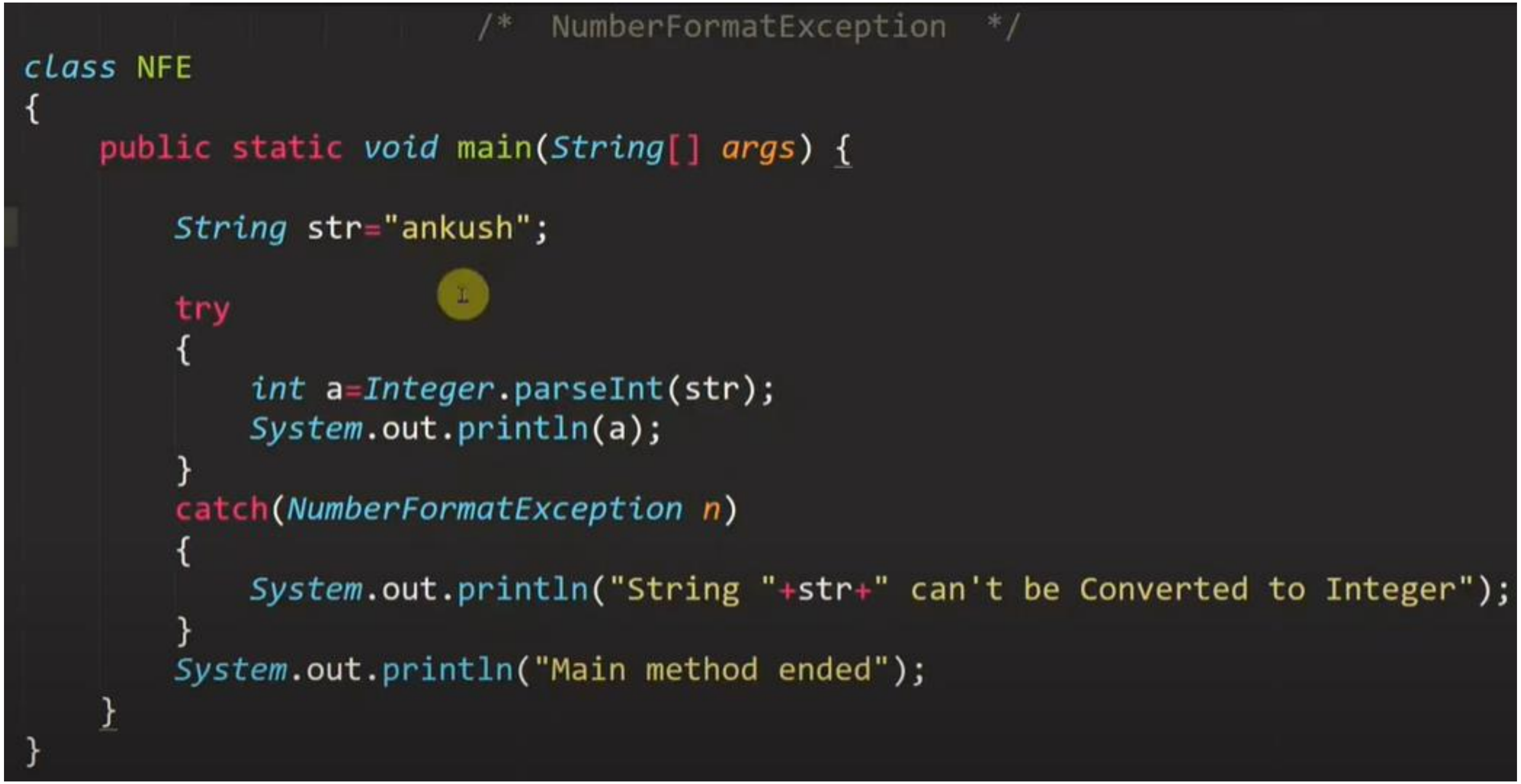
- The try block has to be followed by a catch block.
- No lines of code can come in between Try and Catch.
- There may be multiple catch blocks to one try block.
- Each catch block handles different types of exceptions.
- The catch block is executed as and when the exception matching the exception type in the handler is invoked.

example- catch(ArithmeticException e) will handle only arithmetic exceptions, catch(NullPointException e) will handle only NullPointExceptions.

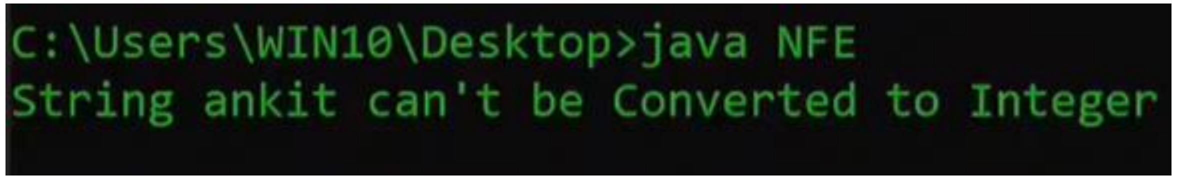
- If no exception occurs, then all the catch blocks will be completely ignored.



```
                                /*  NullPointerException  */
class NPE
{
    public static void main(String[] args) {

        String str=null;

        try
        {
            System.out.println(str.toUpperCase());
        }
        catch(NullPointerException n)
        {
            System.out.println("null can't be casted");
        }
    }
}
```



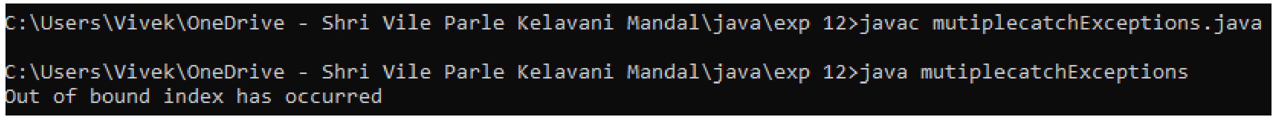
```
                  /*  NumberFormatException  */
class NFE
{
    public static void main(String[] args) {

        String str="ankush";

        try
        {
            int a=Integer.parseInt(str);
            System.out.println(a);
        }
        catch(NumberFormatException n)
        {
            System.out.println("String "+str+" can't be Converted to Integer");
        }
        System.out.println("Main method ended");
    }
}
```



```
C:\Users\WIN10\Desktop>java NFE
String ankit can't be Converted to Integer
```

Multiple catch blocks

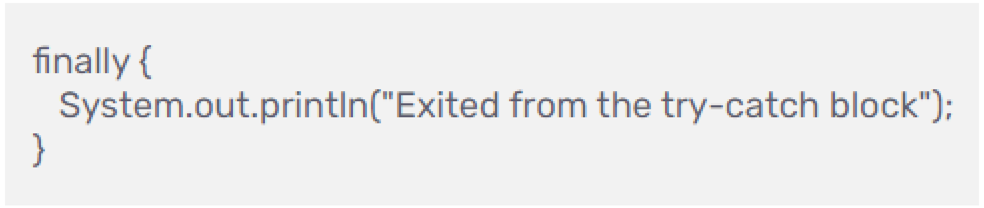
```
class mutiplecatchExceptions {
  public static void main(String args[]) {
    int n = 0;
    int A[] = {1,2,3,4,5};
    try {
      System.out.println("Element: " + A[7]);
      double quotient = 234/n;
      System.out.println("Quotient: " + quotient);
    }
    catch(ArithmeticException e) {
      System.out.println("Division by zero has occurred");
    }
    catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Out of bound index has
occurred");
    }
    catch(Exception e) {
      System.out.println("Some other exception has
occurred");
    }
  }
}
```



```
C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>javac mutiplecatchExceptions.java

C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>java mutiplecatchExceptions
Out of bound index has occurred
```

Finally Block

- ‘finally’ block will be executed whether an exception occurs.
- This block is written after the end of all the catch blocks.
- This block is not just useful in handling exceptions but also helps the programmer to avoid cleaning up the code that may have been bypassed by a continue, return or break statement.
- example, ‘finally’ close the database connection. So that even if there is an exception, we are sure that the database connection will get closed.



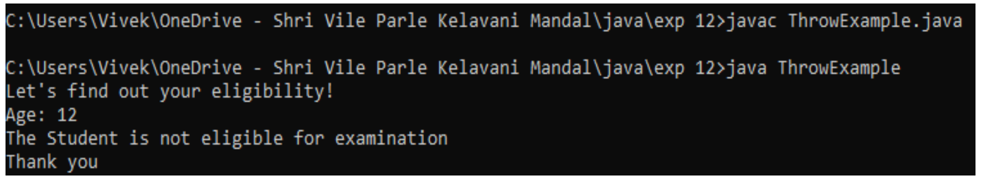
```
finally {
  System.out.println("Exited from the try-catch block");
}
```

PPT is for reference Only, refer to Text Book for Complete Explanations

Throw Block

- Throw keyword to throw custom exceptions or User Defined Exceptions.
- Block can handle exception when thrown by someone or something.
- Either the programmer throws it or the JVM does it.
- We have seen that an instance of the Throwable class has been thrown whenever an exception arises.

- Example to find out whether a student is eligible to sit for an exam based on his/her age. If he/she is older than 15 years, he/she will be eligible for the exam otherwise not. Here, we can explicitly throw an exception with a suitable message with a condition based on age



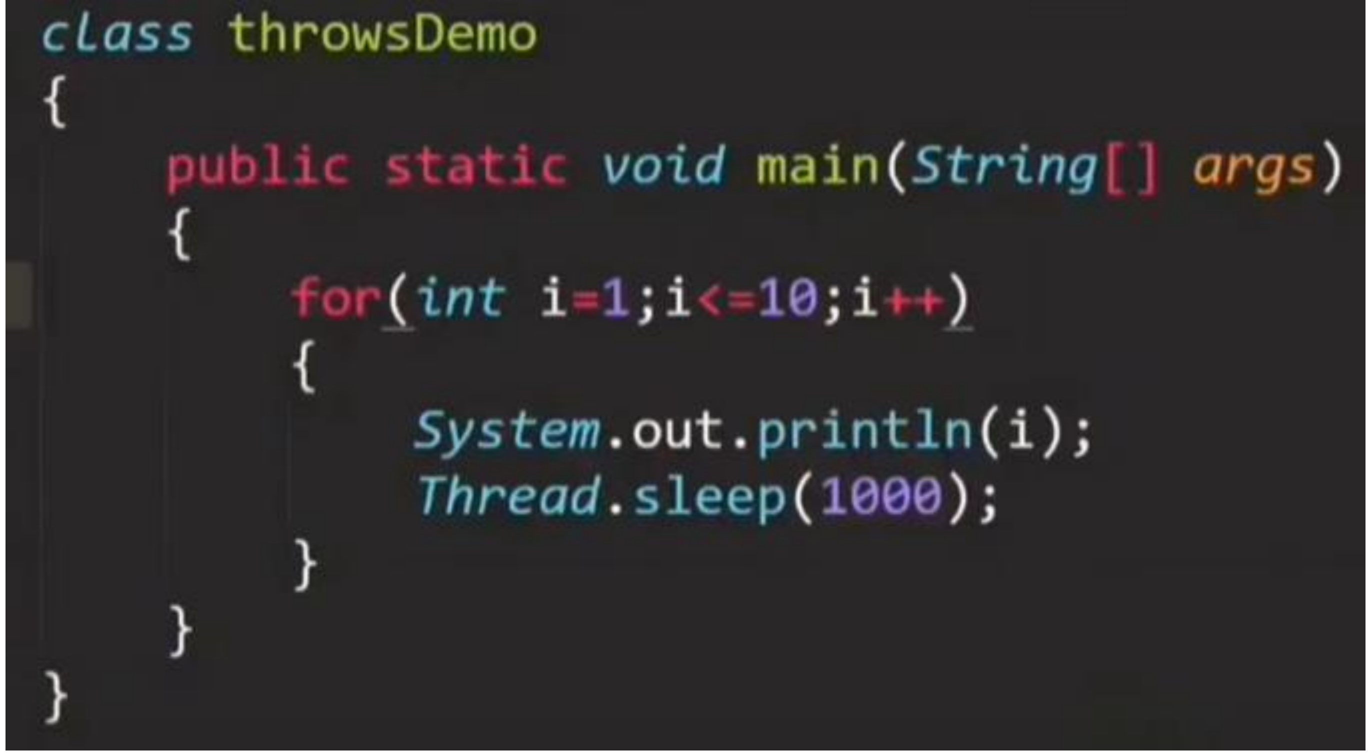
```
C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>javac ThrowExample.java

C:\Users\Vivek\OneDrive - Shri Vile Parle Kelavani Mandal\java\exp 12>java ThrowExample
Let's find out your eligibility!
Age: 12
The Student is not eligible for examination
Thank you
```

```
public class ThrowExample {
  static void checkEligibilty(int age){
   System.out.println("Age: " + age);
   try{
     if(age<15) {
       throw new ArithmeticException("The Student is not
eligible for examination");
     }
     else {
       System.out.println("The Student is Eligible!");
     }
   }
   catch(ArithmeticException e) {
     System.out.println(e.getMessage());
   }
  }
  public static void main(String args[]){
    System.out.println("Let's find out your eligibility!");
    checkEligibilty(12);
    System.out.println("Thank you");
 }
}
```

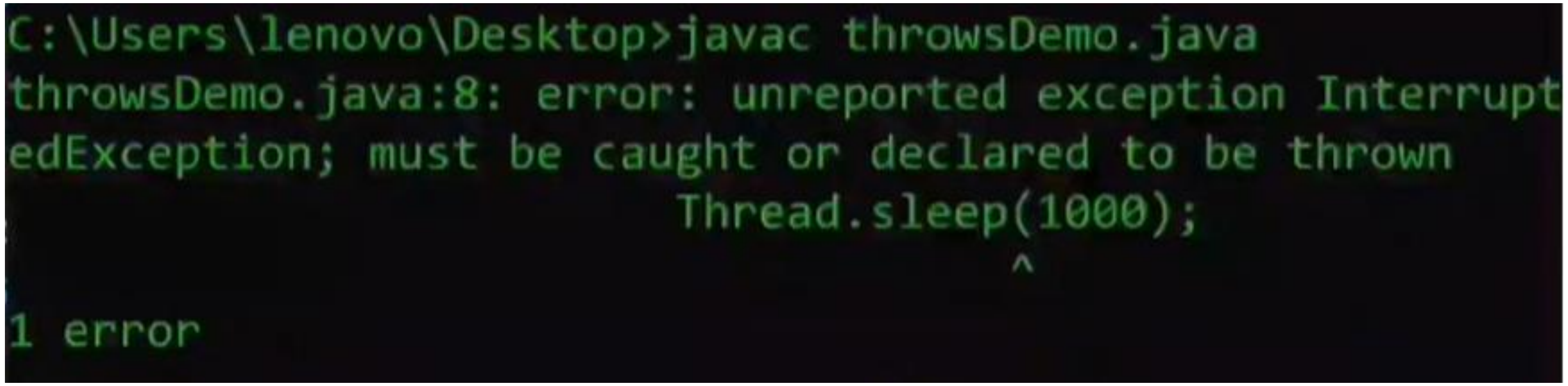
Throws Keyword

- The throws keyword is used to indicate that a particular method may throw an exception of a specific type.
- The caller handles using a try-catch block.
- The throws keyword declares an exception and if not handled, could cause compilation errors.
- This keyword is used to handle only ‘checked exceptions’, i.e., compile-time exceptions.

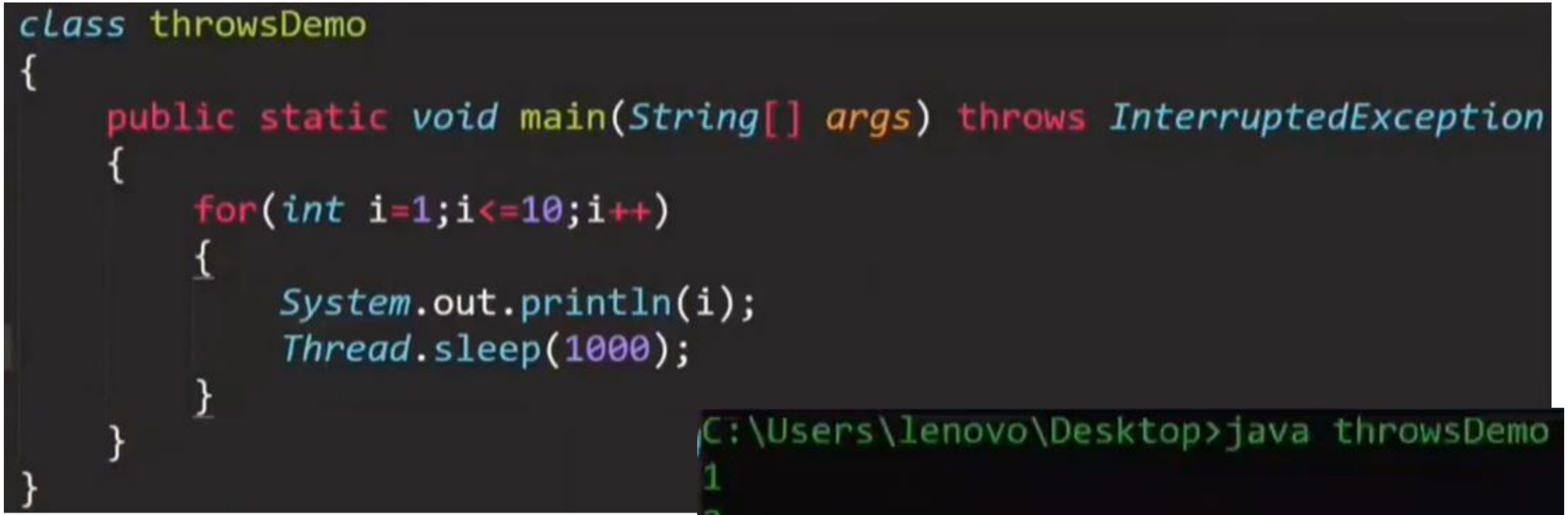
```
import java.io.*;
class ThrowsExample {
   public static void main(String args[]) throws IOException {
      throw new IOException("Exception has arised.");
   }
}
```



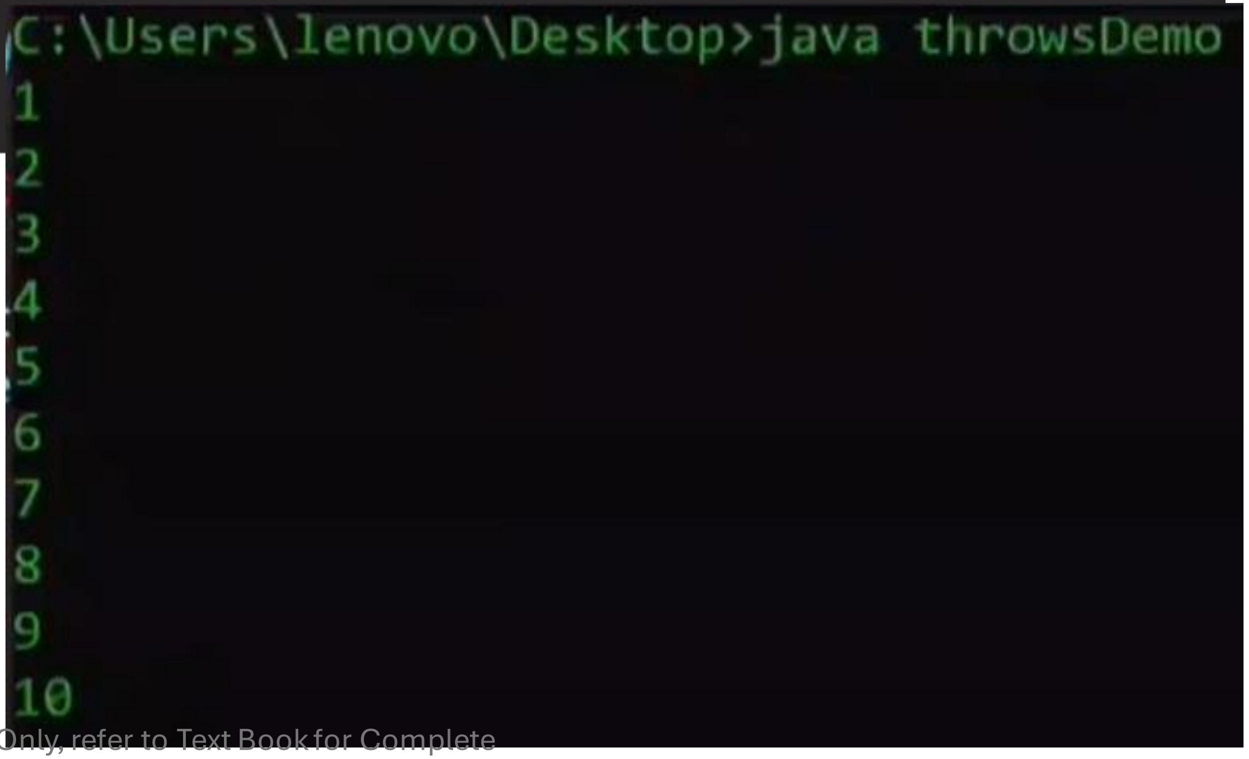
```java
class throwsDemo
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println(i);
            Thread.sleep(1000);
        }
    }
}
```



```
C:\Users\lenovo\Desktop>javac throwsDemo.java
throwsDemo.java:8: error: unreported exception Interrupt
edException; must be caught or declared to be thrown
                        Thread.sleep(1000);
                                    ^
1 error
```



```java
class throwsDemo
{
    public static void main(String[] args) throws InterruptedException
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println(i);
            Thread.sleep(1000);
        }
    }
}
```



```
C:\Users\lenovo\Desktop>java throwsDemo
1
2
3
4
5
6
7
8
9
10
```

JVM will handle the interruption

InputStreamReader

- InputStreamReader acts as a bridge between byte streams and character streams. It reads bytes from an input stream and decodes them into characters based on a specified character encoding. This is important because:
- Programs often deal with text data, which is represented as characters.
- Underlying input sources like files or network connections typically provide data as bytes.

BufferedReader

- BufferedReader is a buffered character-input stream that sits on top of an InputStreamReader. It adds a buffer to improve the performance of reading text data. Here's how it works:
- **Buffering:** BufferedReader internally maintains a character buffer. Instead of reading characters from the underlying InputStreamReader one by one, it reads a block of characters at once and stores them in the buffer.
- **Methods:** BufferedReader inherits methods from Reader and provides additional methods like readLine() to read entire lines of text efficiently.

Throws Example

```
import java.io.*;
class Example {
   public void artOfTesting(int n) throws
IOException, ClassNotFoundException{
      System.out.println("Inside artOfTesting");
      if(n<0)
         throw new IOException("IOException
occurred");
      else
         throw new
ClassNotFoundException("ClassNotFoundExcepti
on occurred");
   }
}
```

```
class ThrowsExample {
   public static void main(String args[]) {
      Example E = new Example();
      try {
         E.artOfTesting(-1);
      }
      catch(IOException e) {
         System.out.println(e.getMessage());
      }
      catch(ClassNotFoundException e) {
         System.out.println(e.getMessage());
      }
      finally {
         System.out.println("Inside main method");
      }
   }
}
```