EXCEPTIONS

Exception Handling

Exceptions

- Java exception: An event that disrupts the normal flow of the program
- Java exception is an object that describes an exceptional condition that has occurred in a piece of code
- When an exceptional condition arises
  - An object representing that exception is created and is thrown in the method that caused the error
  - Method may handle the exception itself (exception is caught and processed) or pass it on
- Generation of exceptions:
  - Either by the Java run-time system (relate to errors that violate the rules of Java language or the constraints of the Java execution environment)
  - Or manually by the code (used to report some error condition to the caller of the method)

Exception Handling

- **Using *try* and *catch*** (Continued …)
  - Syntax:

```
try {
        // block of code to monitor errors
}
catch (ExceptionType1 exOb) {
        // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
        // exception handler for ExceptionType2
}
// …
finally {
        // block of code to be executed after try block ends
}
```

Note: *Java finally block is always executed whether exception is handled or not.*

Exception - Keywords

- Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**
- Program statements that you want to monitor for exceptions are contained within a **try** block
- If an exception occurs within the **try** block, it is thrown. Your code can catch this exception (using **catch**) and handle it in some rational manner
- System-generated exceptions are automatically thrown by the Java runtime system
- To manually throw an exception, use the keyword **throw**
- Any exception that is thrown out of a method must be specified as such by a **throws** clause
- Any code that absolutely must be executed after a **try** block completes is put in a **finally** block.

Exception Handling

- **Exceptions** (Continued ...)

Exception Type	**Cause of exception**
ArithmeticException	Math errors such as division by zero
ArrayIndexOutOfBoundsException	Bad array indexes
ArrayStoreException	Try to store the wrong type of data in an array
FileNotFoundException	Attempt to access a non-existent file
IOException	General I/O failures (unable to read from a file)
NullPointerException	Referencing a null object
NumberFormatException	Conversion between strings and number fails
OutOfMemoryException	No enough memory to allocate a new object
StackOverflowException	System runs out of stack space
StringIndexOutOfBoundsException	Access a non-existent character position in a string

Exception Handling

- **Exceptions** (Continued …)
  - Some examples

1. ArithmeticException

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

2. NullPointerException

```
String s=null;
System.out.println(s.length());
```

3. NumberFormatException

```
String s="abc";
int i=Integer.parseInt(s)
```

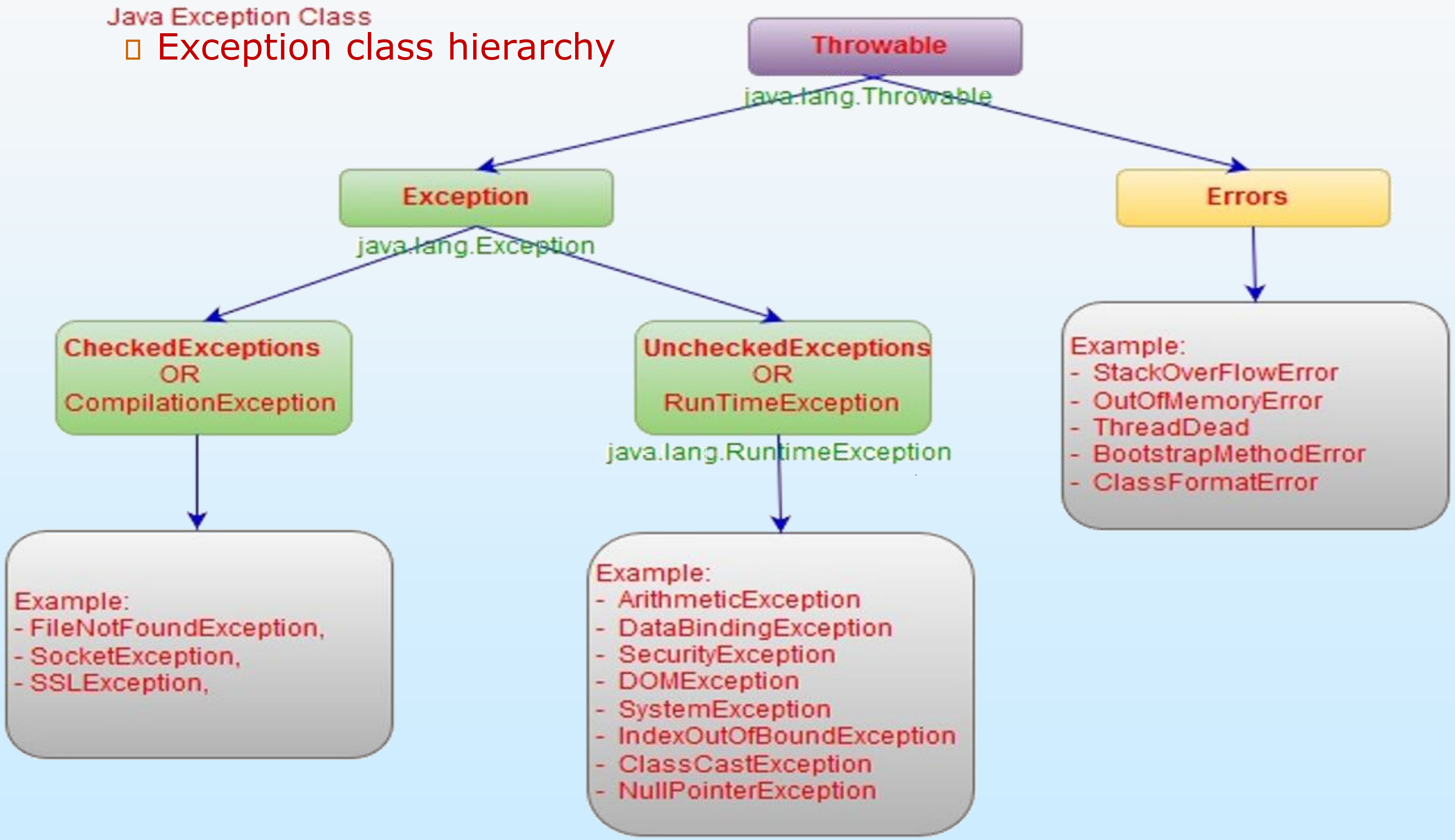
4. ArrayIndexOutOfBoundsException

```
int a[]=new int[5];
a[10]=50;
```

Exception Handling

Exception Types

- Exception class hierarchy



Exception Handling

Exception Types (Continued …)

- Built-in class *Object*
  - All other classes are subclasses of *Object*
  - A reference variable of type *Object* can refer to an object of any other classes
- Built-in class *Throwable* is a subclass of *Object*
  - All exception types are its subclasses
  - 2 subclasses of *Throwable* class
    1. Exception (used for exceptional conditions that user programs should catch, including user-defined exceptions)
       - RuntimeException is a subclass of Exception
       - Examples: Division by zero, Invalid array indexing …
    2. Error (exceptions that are not expected to be caught under normal circumstances; used to indicate run-time errors)
       - Examples: Stack overflow, NoClassDefFoundError

.

Exception Handling

Exception Types (Continued …)

- Checked: Exceptions that are checked at compile time.
  - If some code within a method throws a checked exception then
    - Either the method must handle the exception
    - Or it must specify the exception using *throws* keyword.
- Unchecked: Exceptions that are not checked at compiled time.
  - It is not forced by the compiler to either handle or specify the exception.

Exception Handling

Exception Types (Continued …)

- Example for checked exceptions (1)

```
1.  import java.io.*;
2.  class Checked {
3.      public static void main(String args[]) {
4.          FileReader file = new FileReader("a.txt");
5.          BufferedReader fileInput = new BufferedReader(file);
6.          System.out.println (fileInput.readLine());
7.          fileInput.close();
8.      }
9.  }
```

- Compiler gives errors in lines 4, 6 and 7:

```
unreported exception EXCEPTION; must be caught or declared to be thrown
      Line 4: EXCEPTION = java.io.FileNotFoundException
      Line 6: EXCEPTION = java.io.IOException
      Line 7: EXCEPTION = java.io.IOException
```

Exception Handling

- **Exception Types** (Continued …)
  - Example for checked exceptions (2)

```
import java.io.*;
class Checked {
        public static void main(String args[]) throws IOException {
                FileReader file = new FileReader("a.txt");
                BufferedReader fileInput = new BufferedReader(file);
                System.out.println (fileInput.readLine());
                fileInput.close();
        }
}
```

- With this modification, Compiler does not give any errors

Exception Handling

- **Exception Types** (Continued …)
  - Example for unchecked exceptions

```
class UnChecked {
        public static void main(String args[]) {
                int d = 0;
                int a = 42 / d;
        }
}
```

- Compiler does not give any error;
- During run-time, we get an error indicating division by zero.

Exception Handling

Uncaught Exceptions

- Example:

```
class Exc0 {
        public static void main (String args[]) {
                int d = 0;
                int a = 42 / d;
        }
}
```

- Run-time system detects divide-by-zero error, constructs a new exception object and throws it
- Since it is not caught (by exception handler), Exc0 terminates
- It is caught by default handler provided by Java run-time

```
java.lang.ArithmeticException: / by zero
        at Exc0.main(Exc0.java:4)
```

Exception Handling

- **Uncaught Exceptions** (Continued …)
  - Example (Modified):

```
1.    class Exc1 {
2.          static void subroutine () {
3.                int d = 0;
4.                int a = 42 / d;
5.          }
6.          public static void main (String args[]) {
7.                Exc1.subroutine ();
8.          }
9.    }
```

- Processed by default exception handler

```
java.lang.ArithmeticException: / by zero
      at Exc1.subroutine(Exc1.java:4)
      at Exc1.main(Exc1.java:7)
```

Exception Handling

- **Using *try* and *catch***
  - Benefits of handling an exception
    - Allows us to fix the error
    - Prevents the program from abnormal termination
  - Method of handling an exception
    - Enclose the code to be monitored for error in a *try* block
    - Immediately after the *try* block, include a *catch* clause that specifies the exception type to be caught

Exception Handling

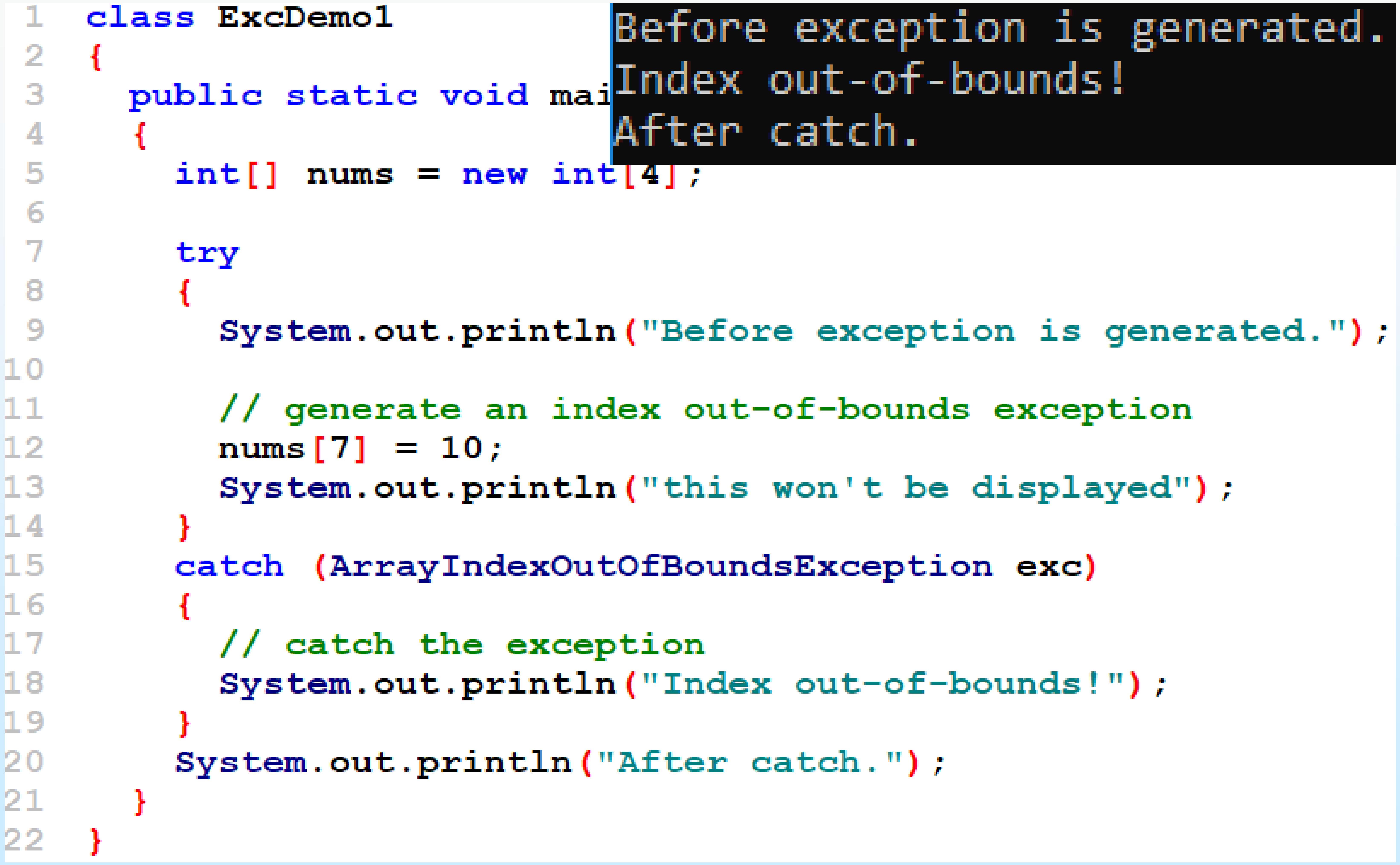
- **Using *try* and *catch*** (Continued …)
  - Example:

```
class Exc {
    public static void main (String args[]) {
        int d, a;
        try {                                   // monitor a block of code.
            d = 0;
            a = 42 / d;
            System.out.println ("This will not be printed if d = 0");
        } catch (ArithmeticException e)     // catch divide-by-zero error
            { System.out.println ("Division by zero"); }
        System.out.println ("After catch statement");
    }
}
```

Output:

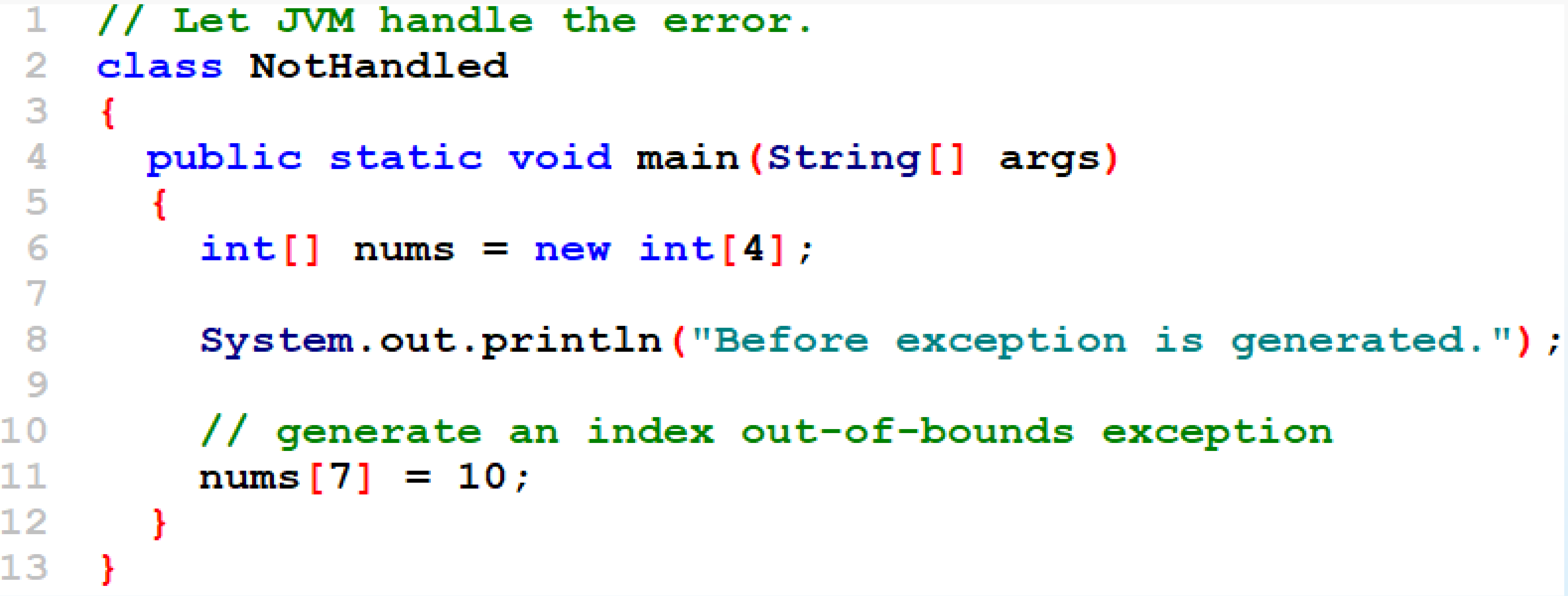
```
Division by zero
After catch statement
```

Demonstrate ArrayIndexOutOfBoundsException.

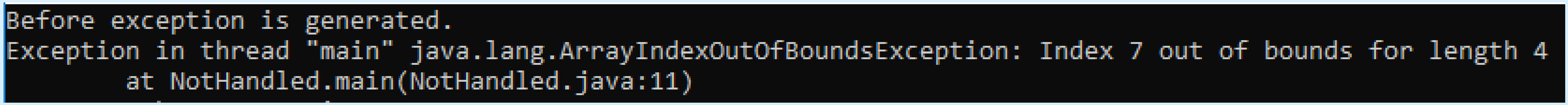


```java
class ExcDemo1
{
  public static void mai
  {
    int[] nums = new int[4];

    try
    {
      System.out.println("Before exception is generated.");

      // generate an index out-of-bounds exception
      nums[7] = 10;
      System.out.println("this won't be displayed");
    }
    catch (ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
      System.out.println("Index out-of-bounds!");
    }
    System.out.println("After catch.");
  }
}
```

```
Before exception is generated.
Index out-of-bounds!
After catch.
```



```
1   // Let JVM handle the error.
2   class NotHandled
3   {
4     public static void main(String[] args)
5     {
6       int[] nums = new int[4];
7
8       System.out.println("Before exception is generated.");
9
10      // generate an index out-of-bounds exception
11      nums[7] = 10;
12    }
13  }
```



```
Before exception is generated.
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 4
        at NotHandled.main(NotHandled.java:11)
```

```java
class ExcTypeMismatch      // This won't work!
{
  public static void main(String[] args)
  {
    in

    try {
      System.out.println("Before exception is generated.");

      //generate an index out-of-bounds exception
      nums[7] = 10;
      System.out.println("this won't be displayed");
    }

    /* Can't catch an array boundary error with an
       ArithmeticException. */
    catch (ArithmeticException exc) {
      // catch the exception
      System.out.println("Index out-of-bounds!");
    }
    System.out.println("After catch.");
  }
}
```

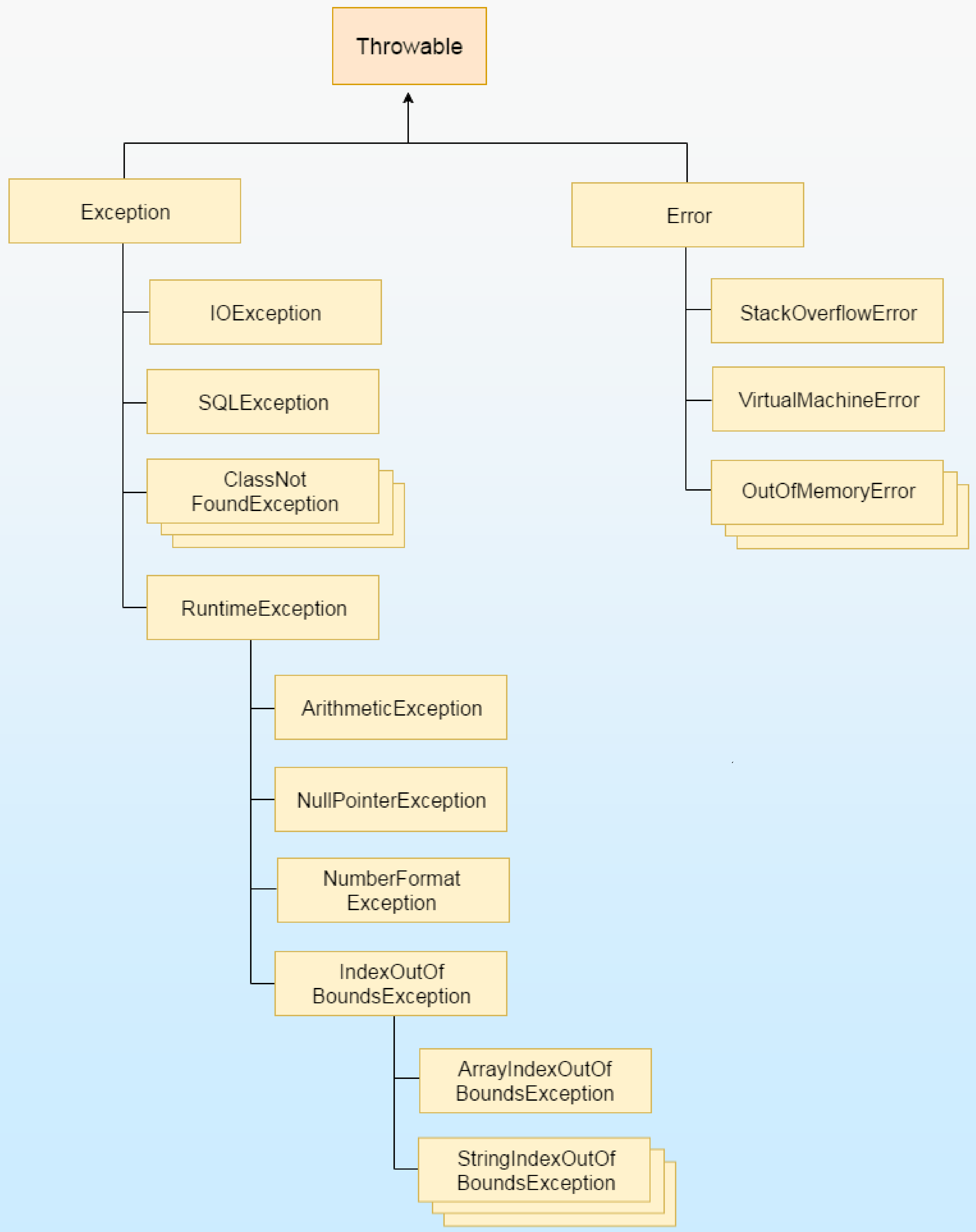
```
Before exception is generated.
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 4
        at ExcTypeMismatch.main(ExcTypeMismatch.java:11)
```

Exception Handling

Java’s Built-in Exception

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero
ArrayIndexOutOfBoundsException	Array index is out-of-bounds
ArrayStoreException	Assignment to an array element of an incompatible type
IllegalArgumentException	Illegal argument used to invoke a method
NegativeArraySizeException	Array created with a negative size
NullPointerException	Invalid use of a null reference
NumberFormatException	Invalid conversion of a string to numeric format
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string

Some unchecked exceptions


Throwable
Exception
Error
IOException
SQLException
ClassNot FoundException
RuntimeException
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException
StackOverflowError
VirtualMachineError
OutOfMemoryError

Question-1

```
public  static  void  main ( String [ ] args )
{
        int [ ] numer = {  4, 8, 16, 32, 64, 128  };
        int [ ] denom = { 2, 0, 4, 4, 0, 8  };

        for( int i = 0; i < numer.length; i++ )
        {
                System.out.println( numer [ i ] / denom[ i ] );
        }
}
```

```java
class ExcDemo3
{
  public static void main(String[] args)
  {
    int[] numer = { 4, 8, 16, 32, 64, 128 };
    int[] denom = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<numer.length; i++)
    {
      try
      {
        System.out.println(numer[i] + "
                           denom[i] + " is " +
                           numer[i]/denom[i]);
      }
      catch (ArithmeticException exc)
      {
        // catch the exception
        System.out.println("Can't divide by Zero!");
      }
    }
  }
}
```

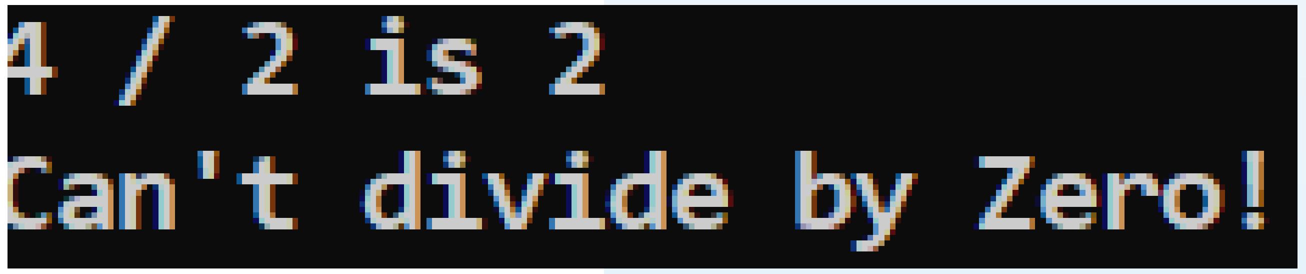
OUTPUT

```
4 / 2 is 2
Can't divide by Zero!
16 / 4 is 4
32 / 4 is 8
Can't divide by Zero!
128 / 8 is 16
```

Question-2: What is the output of the following code

```java
class ExcDemo3
{
  public static void main(String[] args)
  {
     int[] numer = { 4, 8, 16, 32, 64, 128 };
     int[] denom = { 2, 0, 4, 4, 0, 8 };

     try
     {
         for(int i=0; i<numer.length; i++)
         {
             System.out.println(numer[i] + " / " +
                                denom[i] + " is " +
                                numer[i]/denom[i]);
         }
     }
     catch (ArithmeticException exc)
     {
         // catch the exception
         System.out.println("Can't divide by Zero!");
     }
  }
}
```

OUTPUT



```
4 / 2 is 2
Can't divide by Zero!
```

Exception Handling

- **Multiple *catch* clauses**
  - More than one *catch*, each catching a different type of exception
    - Inspected in order
    - First one that matches with the generated exception is executed
    - Remaining are bypassed; execution continues after the try/catch block
  - Example:

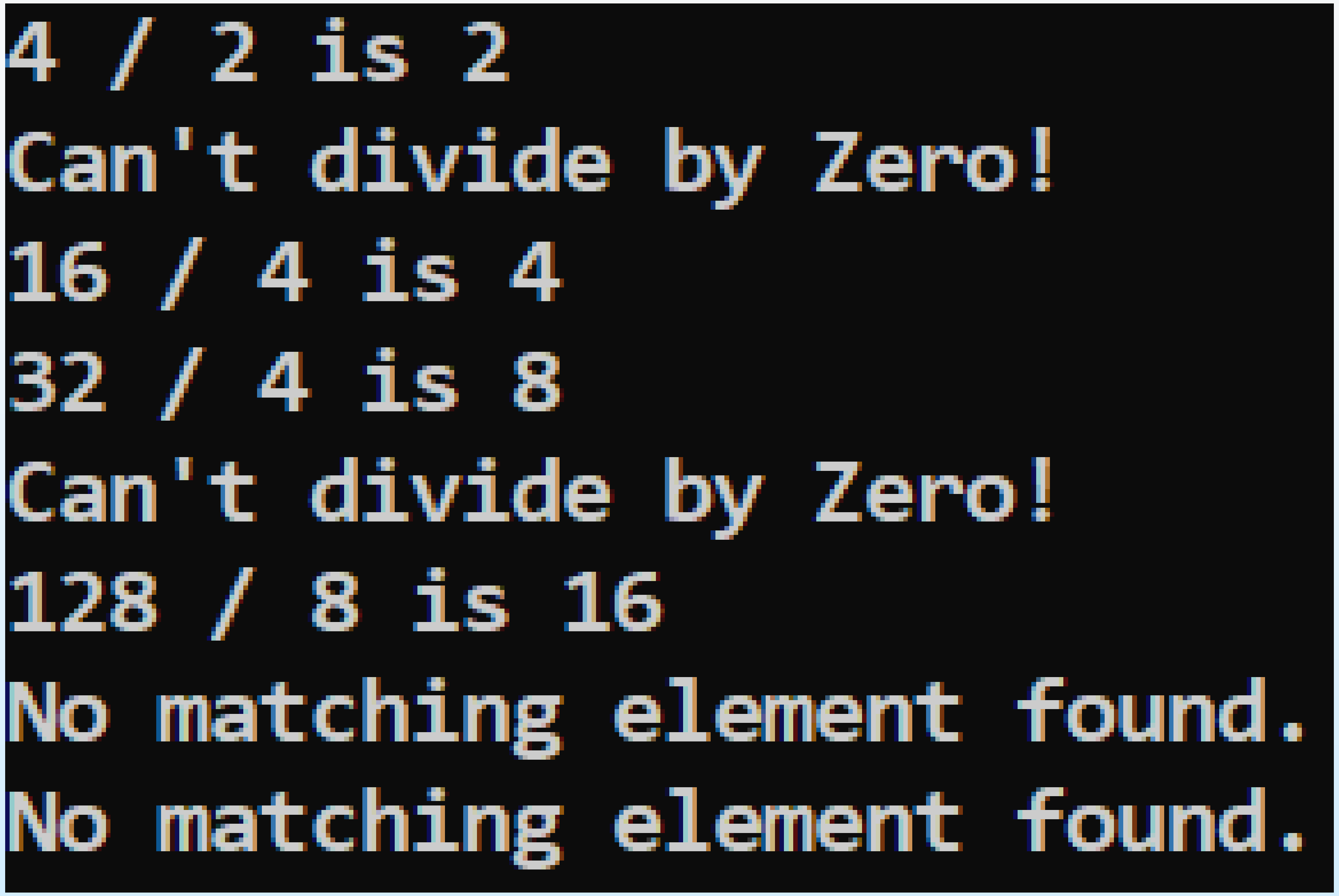
```
try
{
        // .. try block
}
catch (ArithmeticException e)
{
        System.out.println (“Arithmetic exception”);
}
catch (ArrayIndexOutOfBoundsException e)
{
        System.out.println (“Array index out of bounds exception”);
}
```

Question: Define the exception handler for the following code.

```
public  static  void  main ( String [ ] args )
{
        int [ ] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
        int [ ] denom = { 2, 0, 4, 4, 0, 8 };

        for( int i = 0; i < numer.length; i++ )
        {
                System.out.println( numer [ i ] / denom[ i ] );
        }
}
```

```java
class ExcDemo4        // Use multiple catch clauses.
{
  public static void main(String[] args)
  {
    int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int[] denom = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<numer.length; i++)
    {
      try {
        System.out.println(numer[i] + " / " +
                           denom[i] + " is " +
                           numer[i]/denom[i]);
      }
      catch (ArithmeticException exc)
      {  System.out.println("Can't divide by Zero!"); }

      catch (ArrayIndexOutOfBoundsException exc)
      { System.out.println("No matching element found."); }
    }
  }
}
```

OUTPUT:



```
4 / 2 is 2
Can't divide by Zero!
16 / 4 is 4
32 / 4 is 8
Can't divide by Zero!
128 / 8 is 16
No matching element found.
No matching element found.
```

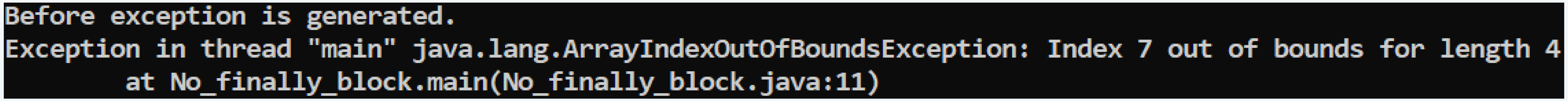
finally block

- The finally block always executes when the try block exits. Java finally block is always executed whether exception is handled or not.
- This ensures that the finally block is executed even if an unexpected exception occurs.

Without *finally* block:

```java
class No_finally_block
{
  public static void main(String[] args)
  {
    int[] nums = new int[4];

    try
    {
      System.out.println("Before exception is generated.");

      nums[7] = 10;
      System.out.println("this won't be displayed");
    }

    catch (ArithmeticException exc)
    {
      System.out.println("Index out-of-bounds!");
    }

    System.out.println("some statement");

    System.out.println("After catch.");
  }
}
```

OUTPUT:

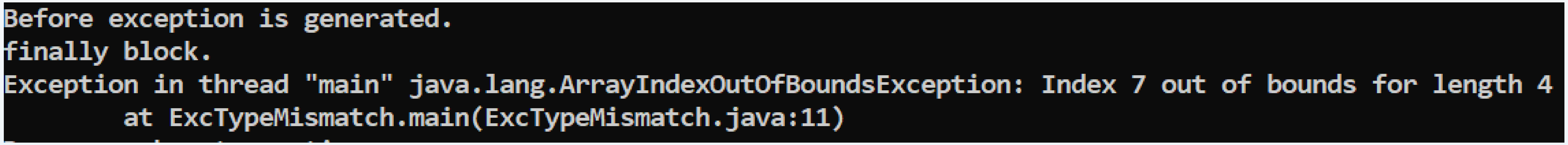


```
Before exception is generated.
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 4
        at No_finally_block.main(No_finally_block.java:11)
```

Finally block demo-1:

```java
class ExcTypeMismatch
{
  public static void main(String[] args)
  {
    int[] nums = new int[4];

    try
    {
      System.out.println("Before exception is generated.");

      nums[7] = 10;
      System.out.println("this won't be displayed");
    }

    catch (ArithmeticException exc)
    {  System.out.println("Index out-of-bounds!"); }

    finally
    {
        System.out.println("finally block.");
    }

    System.out.println("After catch.");
  }
}
```

OUTPUT:

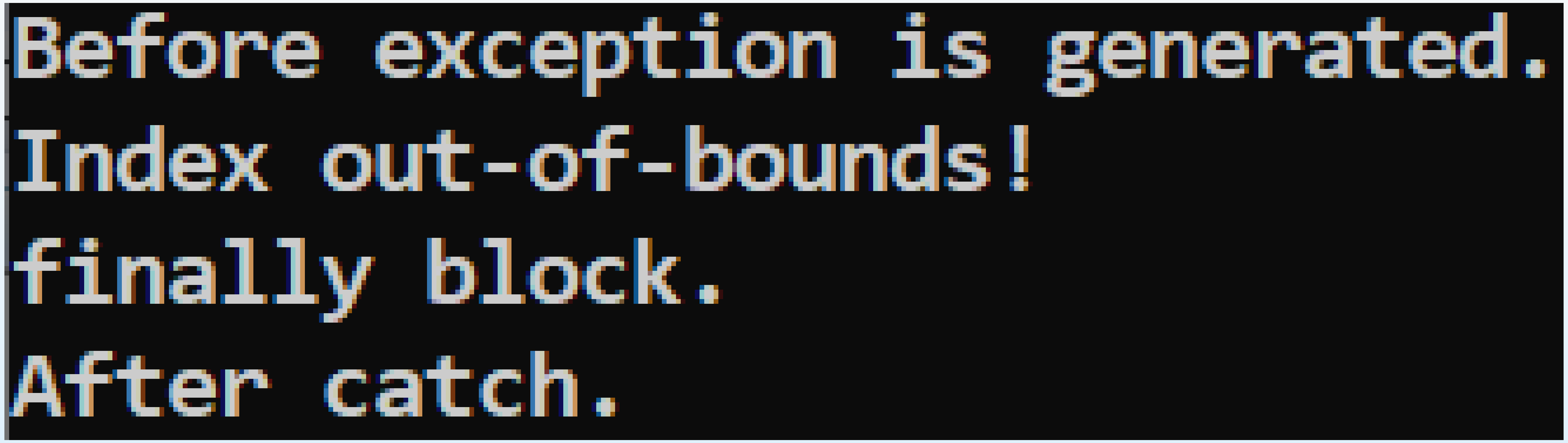


```
Before exception is generated.
finally block.
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 4
        at ExcTypeMismatch.main(ExcTypeMismatch.java:11)
```

Finally block demo-2:

```java
class finally_demo
{
  public static void main(String[] args)
  {
    int[] nums = new int[4];

    try
    {
      System.out.println("Before exception is generated.");

      nums[7] = 10;
      System.out.println("this won't be displayed");
    }

    catch (ArrayIndexOutOfBoundsException exc)
    {  System.out.println("Index out-of-bounds!"); }

    finally
    {
        System.out.println("finally block.");
    }

    System.out.println("After catch.");
  }
}
```

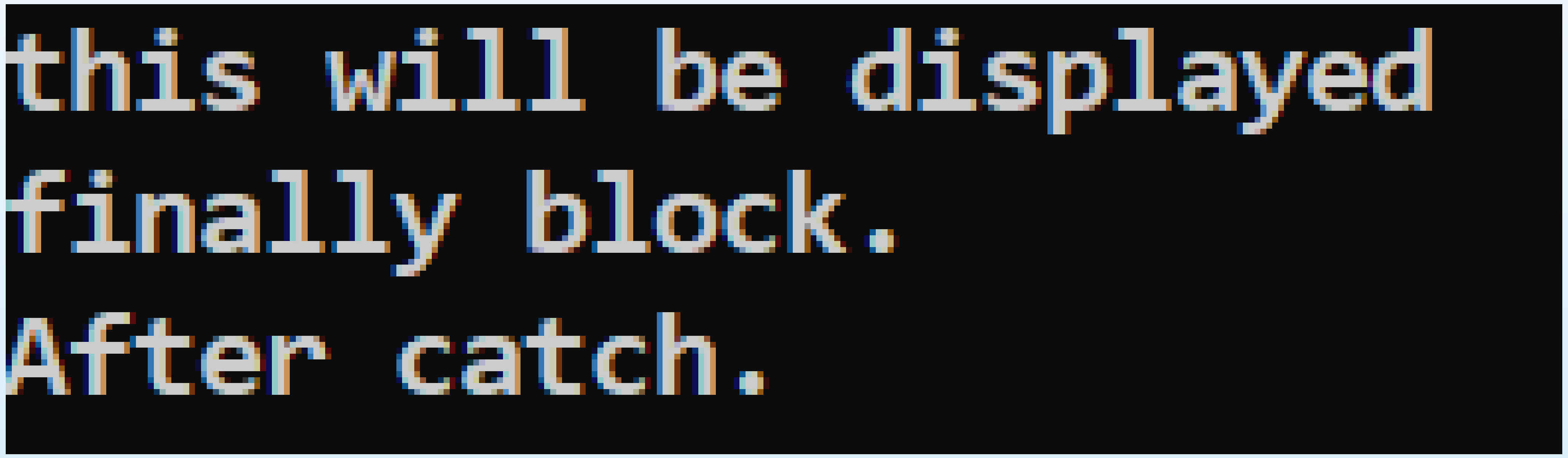
OUTPUT:



```
Before exception is generated.
Index out-of-bounds!
finally block.
After catch.
```

Finally block demo-3:

```
class finally_demo
{
  public static void main(String[] args)
  {
    int[] nums = new int[4];

    try
    {
      //System.out.println("Before exception is generated.");

      //nums[7] = 10;
      System.out.println("this will be displayed");
    }

    catch (ArrayIndexOutOfBoundsException exc)
    {  System.out.println("Index out-of-bounds!"); }

    finally
    {
        System.out.println("finally block.");
    }

    System.out.println("After catch.");
  }
}
```



```
this will be displayed
finally block.
After catch.
```

Exception Handling

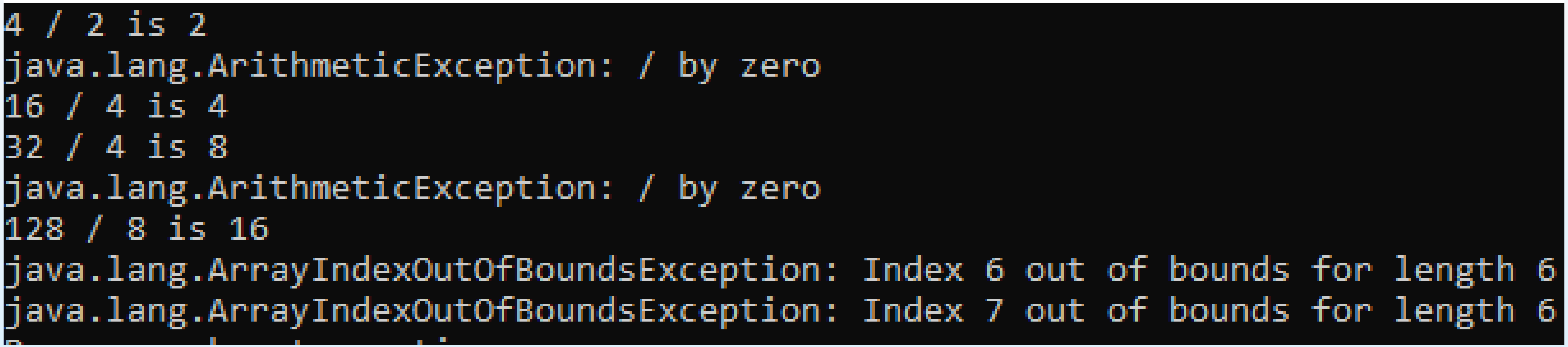
- **Displaying the description of an Exception**
  - *Throwable* returns a string containing the description of the exception
  - Example:

```
try
{
        d = 0;
        a = 42 / d;
        System.out.println ("This will not be printed if d = 0");
}
catch (ArithmeticException e)
{
        System.out.println (“Exception: ” + e);
}
```

Output: **Exception: java.lang.ArithmeticException: / by zero**

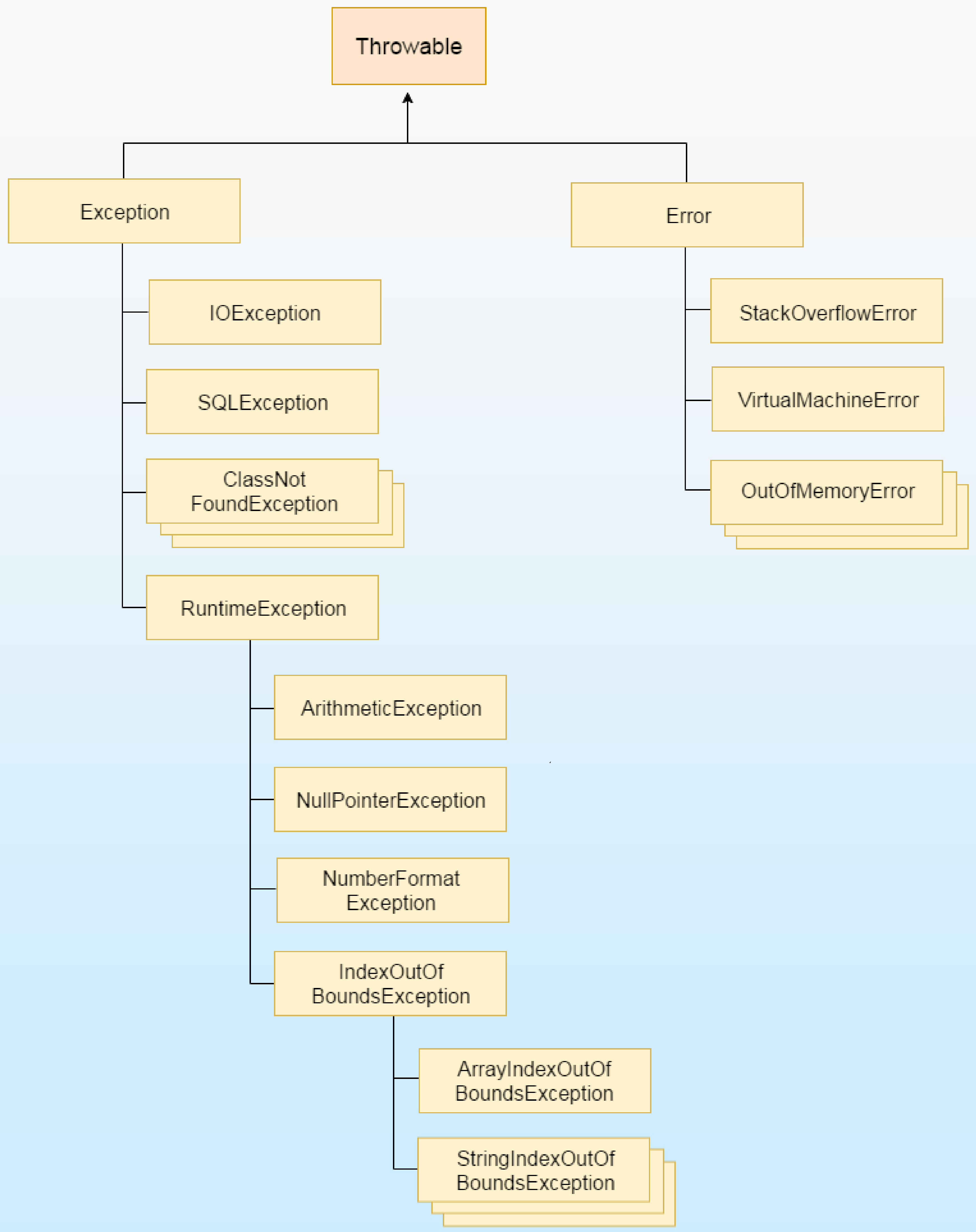
```java
class ExcDemo5
{
  public static void main(String[] args)
  {
    int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int[] denom = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<numer.length; i++)
    {
      try {
        System.out.println(numer[i] + " / " +
                           denom[i] + " is " +
                           numer[i]/denom[i]);
      }
      catch (ArithmeticException exc)
      {  System.out.println(exc); }

      catch (ArrayIndexOutOfBoundsException exc)
      { System.out.println(exc); }
    }
  }
}
```

OUTPUT:



```
4 / 2 is 2
java.lang.ArithmeticException: / by zero
16 / 4 is 4
32 / 4 is 8
java.lang.ArithmeticException: / by zero
128 / 8 is 16
java.lang.ArrayIndexOutOfBoundsException: Index 6 out of bounds for length 6
java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 6
```

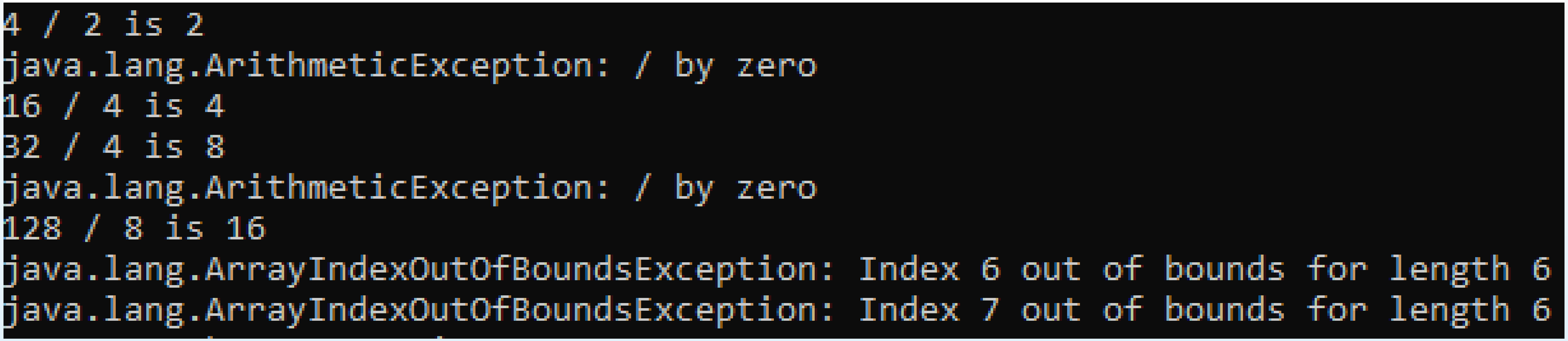
Generic exception type



Exception class demo-1:

```java
class ExcDemo6
{
  public static void main(String[] args)
  {
    int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int[] denom = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<numer.length; i++)
    {
      try
      {
        System.out.println(numer[i] + " / " +
                           denom[i] + " is " +
                           numer[i]/denom[i]);
      }
      catch (Exception exc)
      {  System.out.println(exc); }

    }
  }
}
```

OUTPUT:

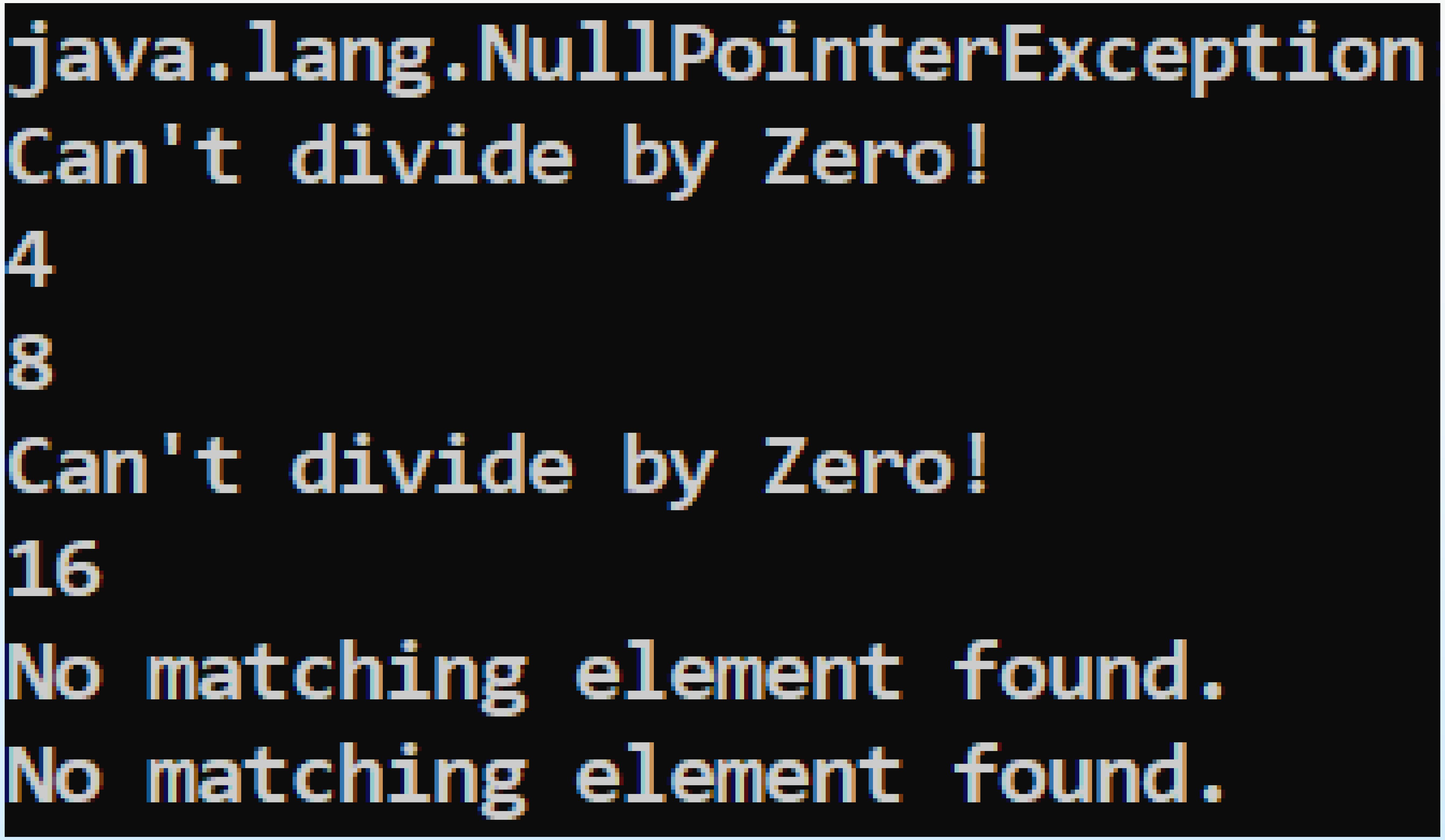


```
4 / 2 is 2
java.lang.ArithmeticException: / by zero
16 / 4 is 4
32 / 4 is 8
java.lang.ArithmeticException: / by zero
128 / 8 is 16
java.lang.ArrayIndexOutOfBoundsException: Index 6 out of bounds for length 6
java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 6
```

Exception class demo-2:

```java
public static void main(String[] args)
{
  int[] num = { 4, 8, 16, 32, 64, 128, 256, 512 };
  int[] denom = { 2, 0, 4, 4, 0, 8 } , temp = null;

  for(int i=0; i<num.length; i++)
  {
      try {

        if( i == 0 ) System.out.println( temp.length );

        System.out.println( num[i]/denom[i]);
      }
     catch (ArithmeticException exc)
     {   System.out.println("Can't divide by Zero!"); }

     catch (ArrayIndexOutOfBoundsException exc)
     { System.out.println("No matching element found."); }

     catch (Exception exc)
     {   System.out.println(exc); }
  }
}
```

OUTPUT:



```
java.lang.NullPointerException
Can't divide by Zero!
4
8
Can't divide by Zero!
16
No matching element found.
No matching element found.
```

Generic Exception handler

```
class Generic_exception_handler
{
  public static void main(String[] args)
  {
    int[] num = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int[] denom = { 2, 0, 4, 4, 0, 8 } , temp = null;

    for(int i=0; i<num.length; i++)
    {
        try
        {

          if( i == 0 )
            System.out.println( temp.length );

          System.out.println( num[i]/denom[i]);
        }

       catch (Exception exc)
       {
           System.out.println(exc);
       }
    }
  }
}
```

Exception Handling

- **Multiple *catch* clauses** (Continued …)
  - While using more than one *catch* statements, exception subclasses must come before any of their superclasses

```
try
{
        int a = 0;
        int b = 42 / a;
}
catch (Exception e)
        { System.out.println ("Generic Exception catch."); }

catch (ArithmeticException e)
        { System.out.println ("This is never reached."); }
```

- Second *catch* statement is unreachable (compiler error)

```
exception java.lang.ArithmeticException has already been caught
catch(ArithmeticException e)
```

  - *ArithmeticException is* a subclass of *Exception* (first *catch* statement will handle all exceptions)
- Reverse the order of *catch* statements

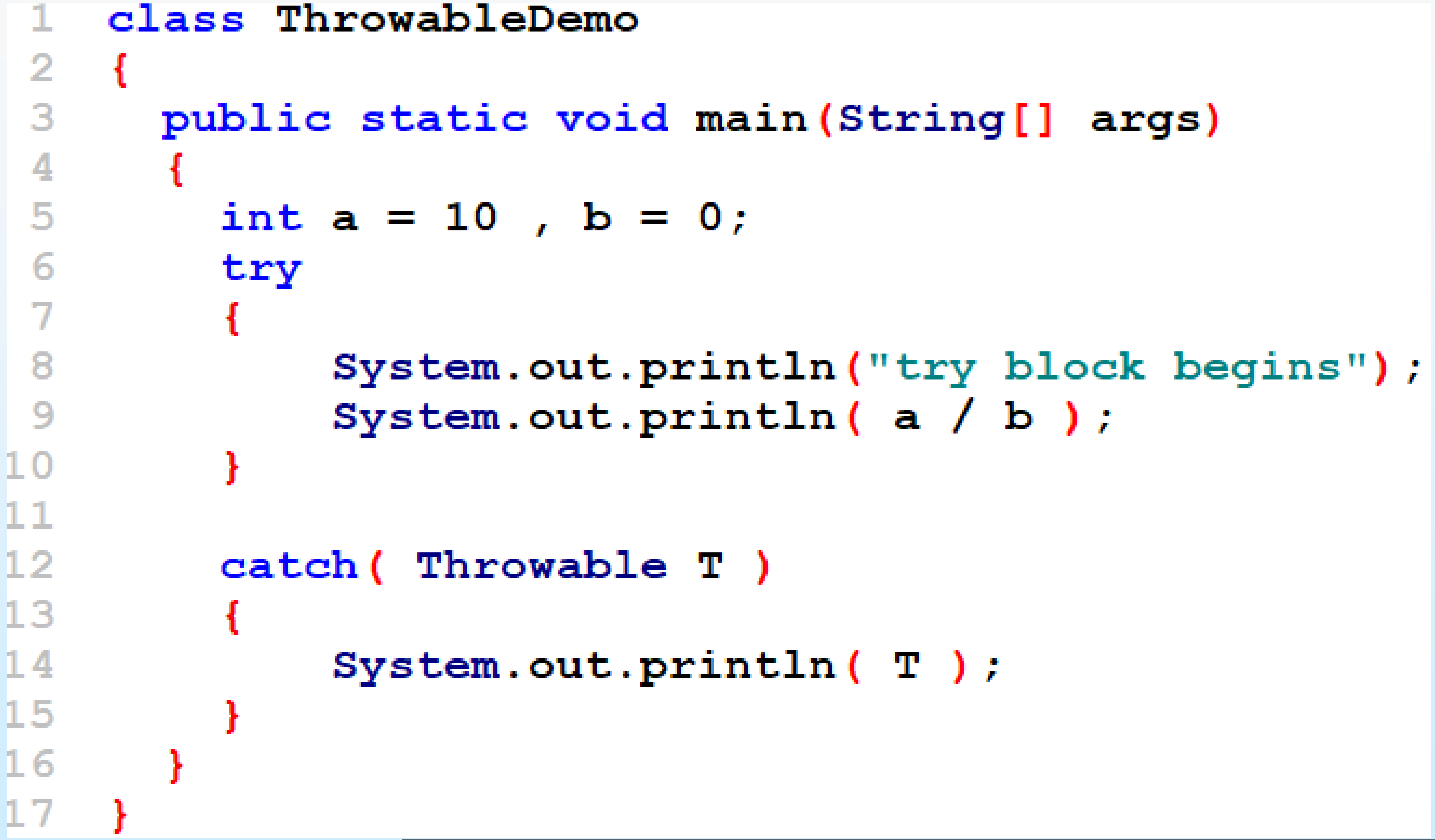
```
    public static void main(String[] args)
    {
      int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
      int[] denom = { 2, 0, 4, 4, 0, 8 };

      for(int i=0; i<numer.length; i++)
      {
        try {
          System.out.println( numer[i]/denom[i]);
        }

        catch (Exception exc)
        {  System.out.println(exc); }

        catch (ArithmeticException exc)
        {  System.out.println("Can't divide by Zero!"); }

        catch (ArrayIndexOutOfBoundsException exc)
        { System.out.println("No matching element found."); }
      }
    }
```

Error !

```java
    public static void main(String[] args)
    {
      int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
      int[] denom = { 2, 0, 4, 4, 0, 8 };

      for(int i=0; i<numer.length; i++)
      {
        try {
          System.out.println( numer[i]/denom[i]);
        }

        catch (ArithmeticException exc)
        {  System.out.println("Can't divide by Zero!"); }

        catch (ArrayIndexOutOfBoundsException exc)
        { System.out.println("No matching element found."); }

        catch (Exception exc)
        {  System.out.println(exc); }
      }
    }
```

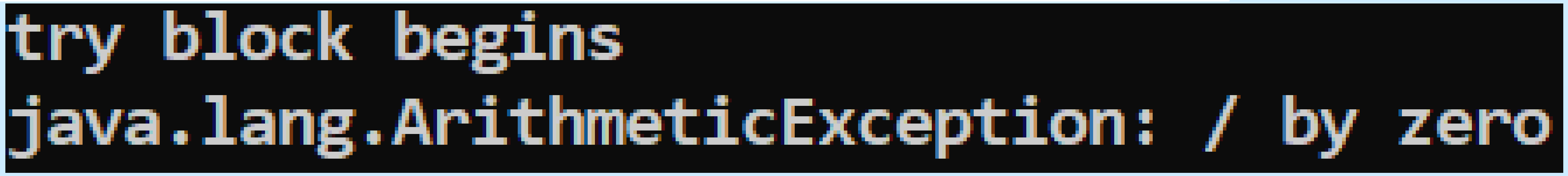
OK

Question-1:

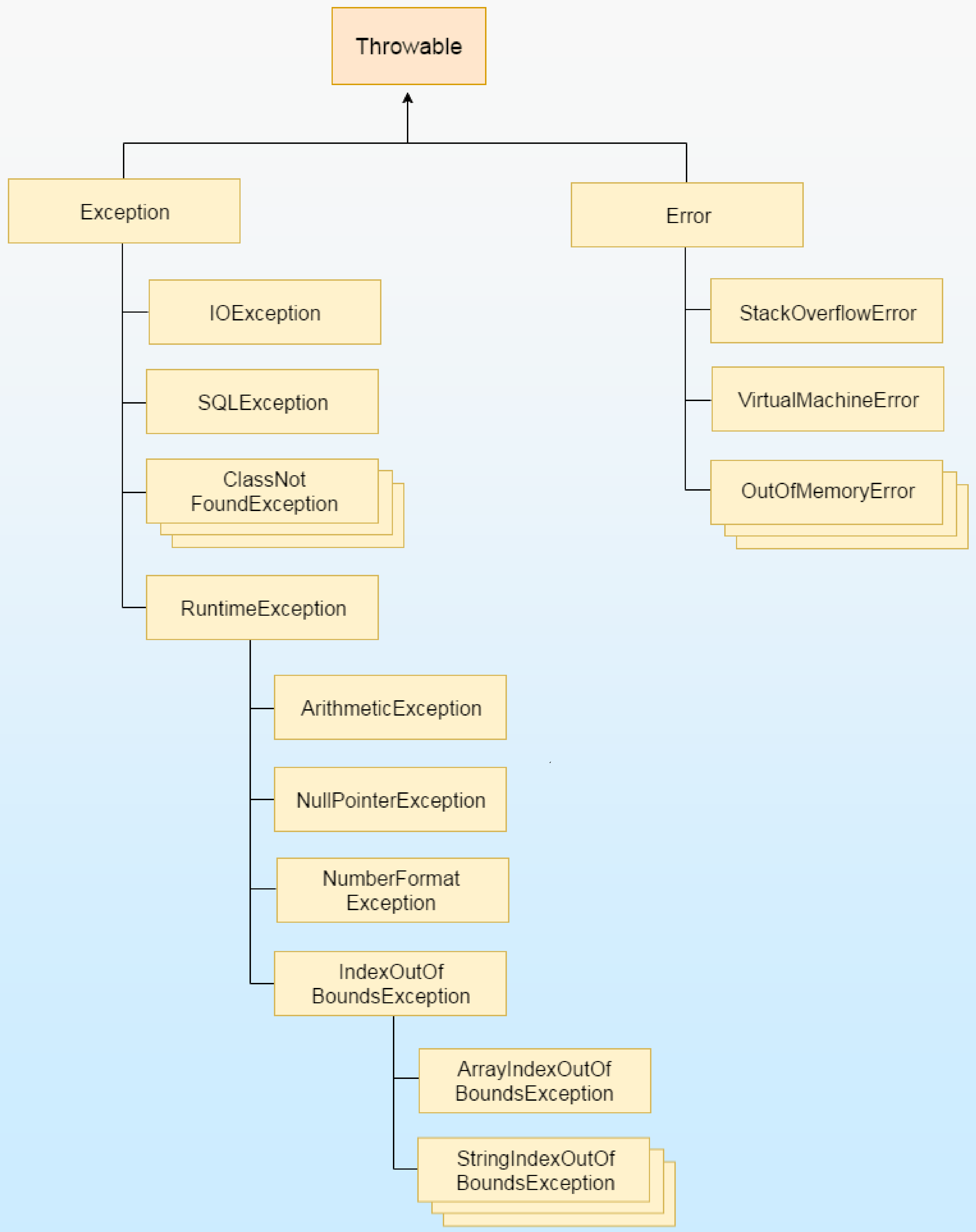


```java
class ThrowableDemo
{
  public static void main(String[] args)
  {
    int a = 10 , b = 0;
    try
    {
        System.out.println("try block begins");
        System.out.println( a / b );
    }

    catch( Throwable T )
    {
        System.out.println( T );
    }
  }
}
```

OUTPUT:



```
try block begins
java.lang.ArithmeticException: / by zero
```


Throwable
Exception
Error
IOException
SQLException
ClassNot FoundException
RuntimeException
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException
StackOverflowError
VirtualMachineError
OutOfMemoryError

Question-2:

```
class ThrowableDemo
{
  public static void main(String[] args)
  {
    int a = 10 , b = 0;
    try
    {
        System.out.println("try block begins");
        System.out.println( a / b );
    }

    catch( Throwable T )
    {
        System.out.println( T );
    }

    catch (ArithmeticException exc)
    {
       System.out.println("Can't divide by Zero!");
    }
  }
}
```

Error !

Solution:

```java
class ThrowableDemo
{
  public static void main(String[] args)
  {
    int a = 10 , b = 0;
    try
    {
        System.out.println("try block begins");
        System.out.println( a / b );
    }

    catch (ArithmeticException exc)
    {
       System.out.println("Can't divide by Zero!");
    }
    catch( Throwable T )
    {
        System.out.println( T );
    }
  }
}
```

Exception Handling

- **Nested *try* statements**
  - A *try* statement can be inside another *try* block
  - An exception generated within the inner **try** block that is not caught by a **catch** associated with that **try** is propagated to the outer **try** block.
  - If no *catch* statement matches, Java run-time system will handle the exception

```
// . . .
try //try block-1
{
    // . . .

    try //try block-2
    {
        // . . .

        try //try block-3
        {
            // exception generated.
        }
        catch // catch block-1
        { }
    }
    catch // catch block-2
    { }
}
catch // catch block-3
{ }
// . . .
```

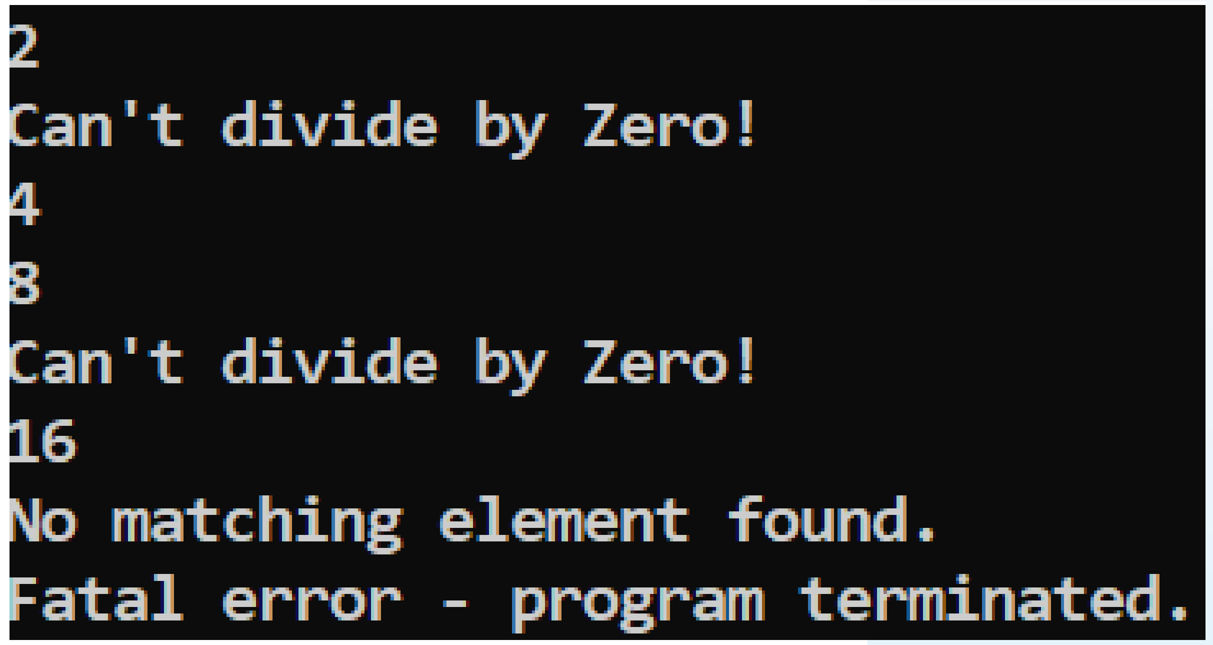
Exception Handling

- **Nested *try* statements**
  - Each time a *try* statement is entered, context of that exception is pushed on to the stack
  - If an inner *try* does not have a *catch* handler for a particular exception, stack is unwound and next *try* statement’s *catch* handlers are inspected for a match
  - Continues until one of the *catch* statements matches or until all nested *try* statements are exhausted
  - If no *catch* statement matches, Java run-time system will handle the exception

Nested try example

```java
    public static void main(String[] args)
    {
      int[] num = { 4, 8, 16, 32, 64, 128, 256, 512 };
      int[] denom = { 2, 0, 4, 4, 0, 8 };

      try
      { // outer try
        for(int i=0; i<num.length; i++)
        {
          try
          { // nested try
            System.out.println( num[i] / denom[i] );
          }
          catch (ArithmeticException exc)
          {
            System.out.println("Can't divide by Zero!");
          }
        }
      }
      catch (ArrayIndexOutOfBoundsException exc)
      {
        System.out.println("No matching element found.");
        System.out.println("Fatal error - program terminated.");
      }
    }
```

OUTPUT:



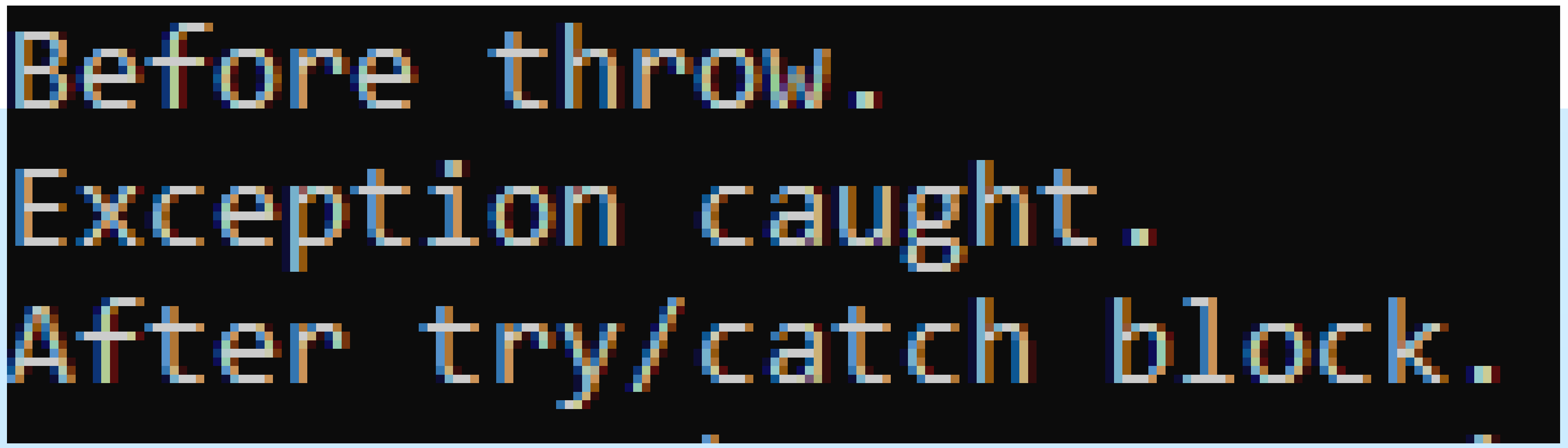
```
2
Can't divide by Zero!
4
8
Can't divide by Zero!
16
No matching element found.
Fatal error - program terminated.
```

Exception Handling

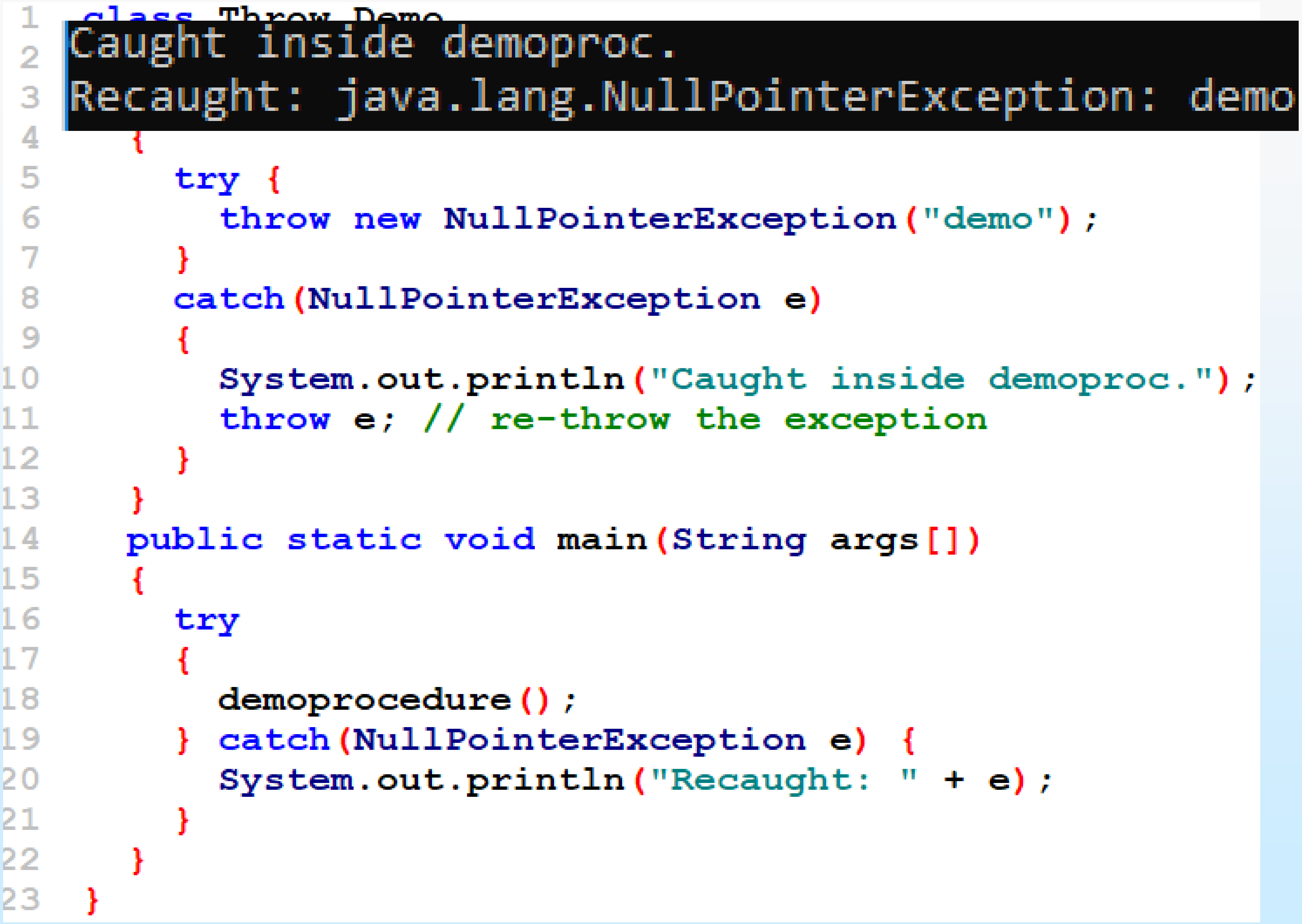
The *throw* statement

- Explicitly throwing an exception in a program
- Syntax: throw *ThrowableInstance*
  - *ThrowableInstance* must be an object of class *Throwable* or its subclass
- Action:
  - Flow of execution stops immediately after the *throw* statement
  - Subsequent statements are not executed
  - Searches for a *catch* block in *try* blocks from innermost to outermost level
  - If there is a match, control is transferred to that *catch* block; otherwise default exception handler terminates the program

```java
// Manually throw an exception.
class Throw_Demo
{
  public static void main(String[] args)
  {
    try
    {
      System.out.println("Before throw.");
      throw new ArithmeticException();
    }
    catch (ArithmeticException exc)
    {
      System.out.println("Exception caught.");
    }
    System.out.println("After try/catch block.");
  }
}
```



```
Before throw.
Exception caught.
After try/catch block.
```



```
Caught inside demoproc.
Recaught: java.lang.NullPointerException: demo
```

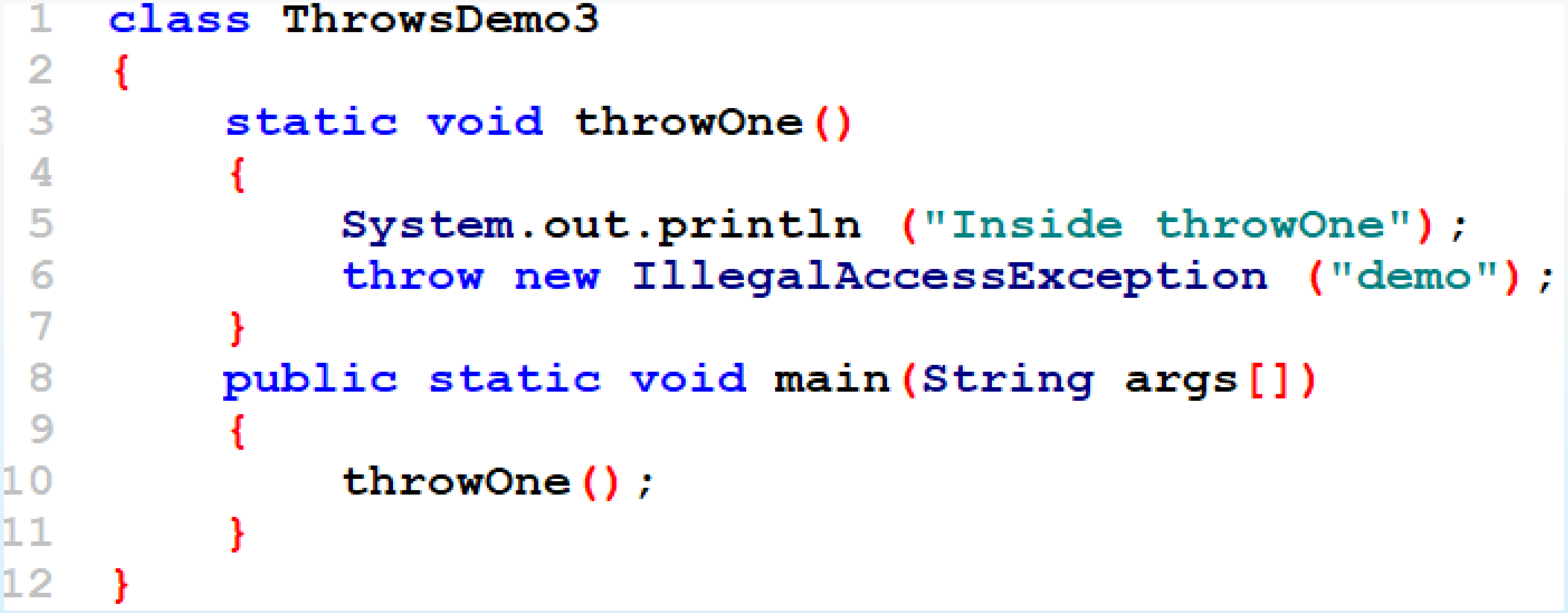
```java
class ThrowDemo
  {
    try {
      throw new NullPointerException("demo");
    }
    catch(NullPointerException e)
    {
      System.out.println("Caught inside demoproc.");
      throw e; // re-throw the exception
    }
  }
  public static void main(String args[])
  {
    try
    {
      demoprocedure();
    } catch(NullPointerException e) {
      System.out.println("Recaught: " + e);
    }
  }
}
```

Exception Handling

- **The *throws* clause**
  - Used when a method is capable of causing an exception that it does not handle
    - To indicate this to callers of the method, so that they can take necessary actions
  - Necessary for all exceptions, except those of type *Error* and *RuntimeException*, or any of their subclasses
  - All exceptions that a method can throw must be declared in the *throws* clause; otherwise, compile-time error results
  - Syntax:

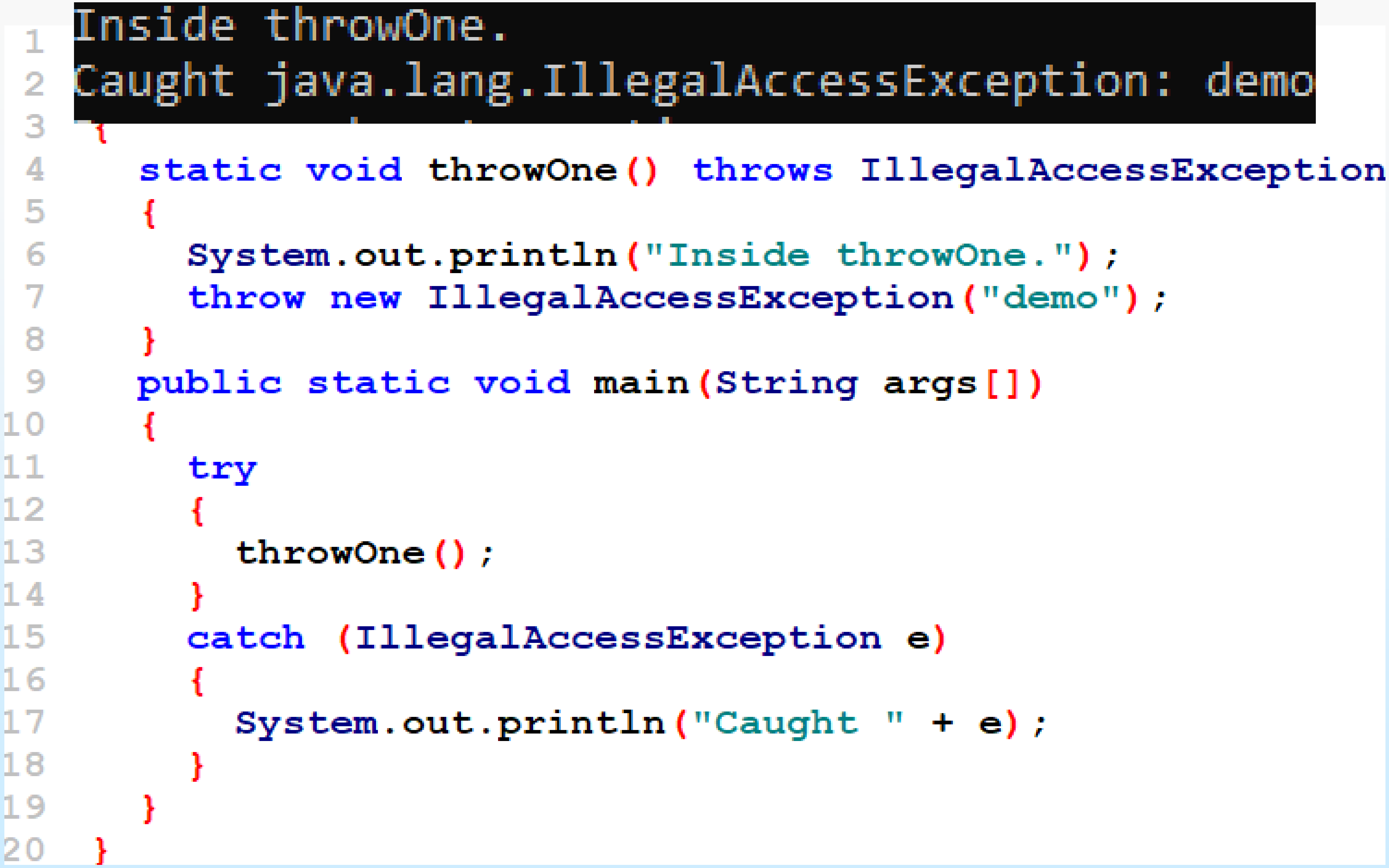
```
type method-name (parameter-list) throws exception-list
{
        // body of method
}
```

    - *exception-list* is a comma-separated list of exceptions



```java
class ThrowsDemo3
{
    static void throwOne()
    {
        System.out.println ("Inside throwOne");
        throw new IllegalAccessException ("demo");
    }
    public static void main(String args[])
    {
        throwOne();
    }
}
```

Error: unreported exception IllegalAccessException; must be caught or declared to be thrown



```
Inside throwOne.
Caught java.lang.IllegalAccessException: demo
```

```java
{
  static void throwOne() throws IllegalAccessException
  {
    System.out.println("Inside throwOne.");
    throw new IllegalAccessException("demo");
  }
  public static void main(String args[])
  {
    try
    {
      throwOne();
    }
    catch (IllegalAccessException e)
    {
      System.out.println("Caught " + e);
    }
  }
}
```

Exception Handling

- **The *finally* keyword**(Continued …)

  - Creates a block of code that will be executed after a try/catch block has completed and before the code following the try/catch block
  - When a method is about to return to the caller from inside a try/catch block, via an uncaught exception or an explicit return statement, the *finally* block is executed just before the method returns

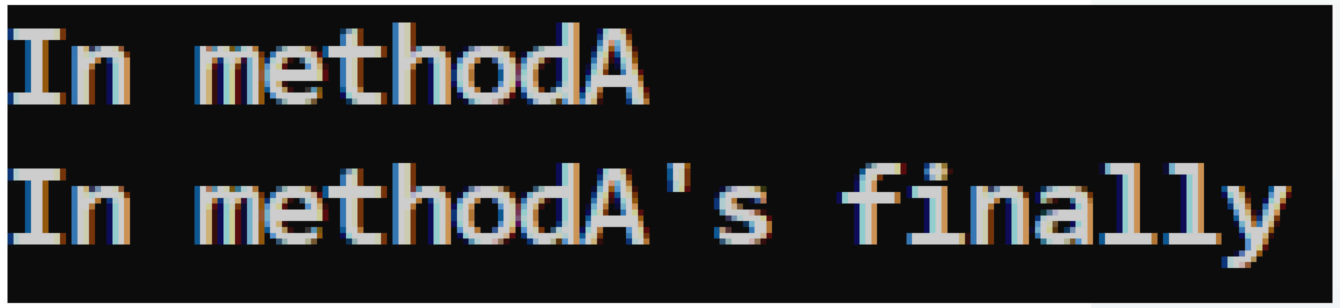
The *finally* keyword (Continued …)

```java
class FinallyDemo1
{
    static void methodA()
    {
        try
        {
            System.out.println("In methodA");
            throw new RuntimeException();
        }

        finally
        { System.out.println ("In methodA's finally");  }
    }

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

```
In methodA
In methodA's finally
Exception caught: java.lang.RuntimeException
```

The *finally* keyword – Example 2

```
class FinallyDemo2
{
    static void methodA()
    {
        try
        {
            System.out.println("In methodA");
            return;
        }

        finally
        { System.out.println ("In methodA's finally");   }
    }

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

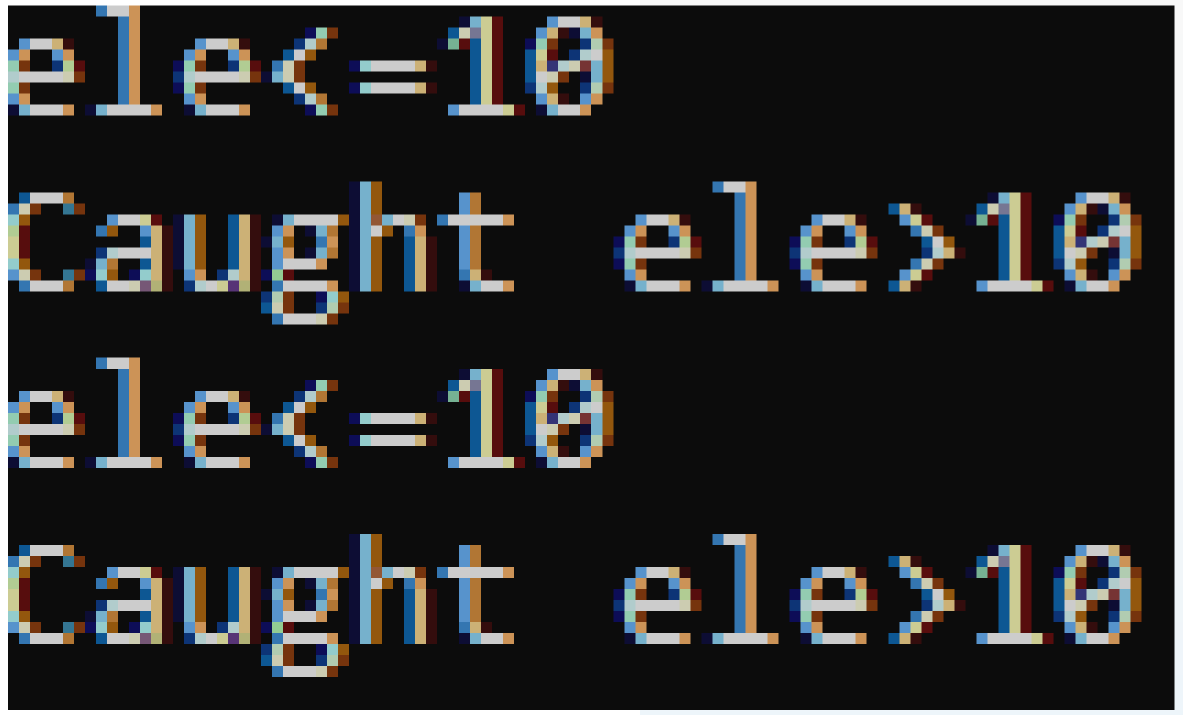


```
In methodA
In methodA's finally
```

Exception Handling

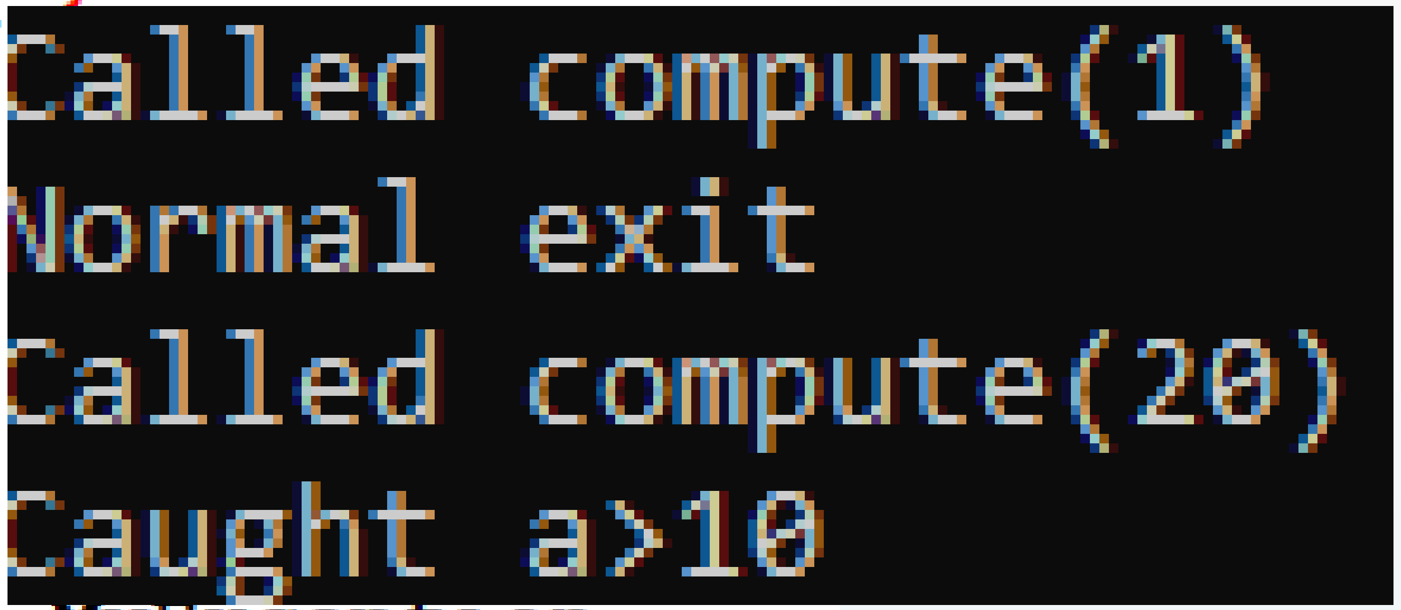
- **User-Defined exceptions**
  - Define a subclass of Exception (which is a subclass of Throwable)
    - Exception class does not define any methods of its own; inherits all methods of Throwable
    - Override these methods
  - Constructors
    - Exception()
    - Exception(String)

```
class MyException extends Exception
{
  String des;
  MyException( String ex )
  {  des = ex;  }
}

class CustomExceptionDemo
{
  public static void main(String args[])
  {
    int a[] = { 5 , 15 , 10 , 20 };
    for( int ele : a )
    {
        try
        {
            if( ele > 10)
                throw new MyException("ele>10");
            System.out.println("ele<=10");
        }
        catch( MyException e )
        { System.out.println("Caught " + e.des ); }
    }
  }
}
```



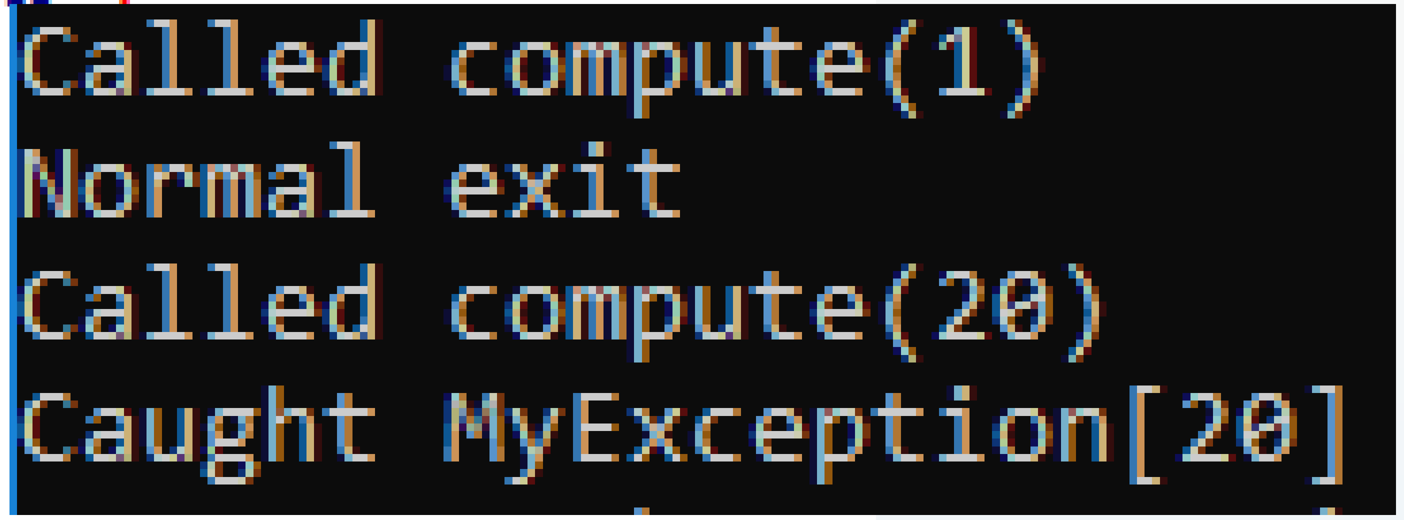
```
ele<=10
Caught ele>10
ele<=10
Caught ele>10
```

```java
class MyException extends Exception {
  String des;
  MyException( String ex )
  {   des = ex;   }
}

class CustomExceptionDemo2 {
  static void compute(int a) throws MyException
  {
    System.out.println("Called compute(" + a + ")");
    if(a > 10)
      throw new MyException("a>10");
    System.out.println("Normal exit");
  }

  public static void main(String args[])
  {
    try {
      compute(1);
      compute(20);
    }
    catch( MyException e )
    { System.out.println("Caught " + e.des ); }
  }
}
```



```
Called compute(1)
Normal exit
Called compute(20)
Caught a>10
```

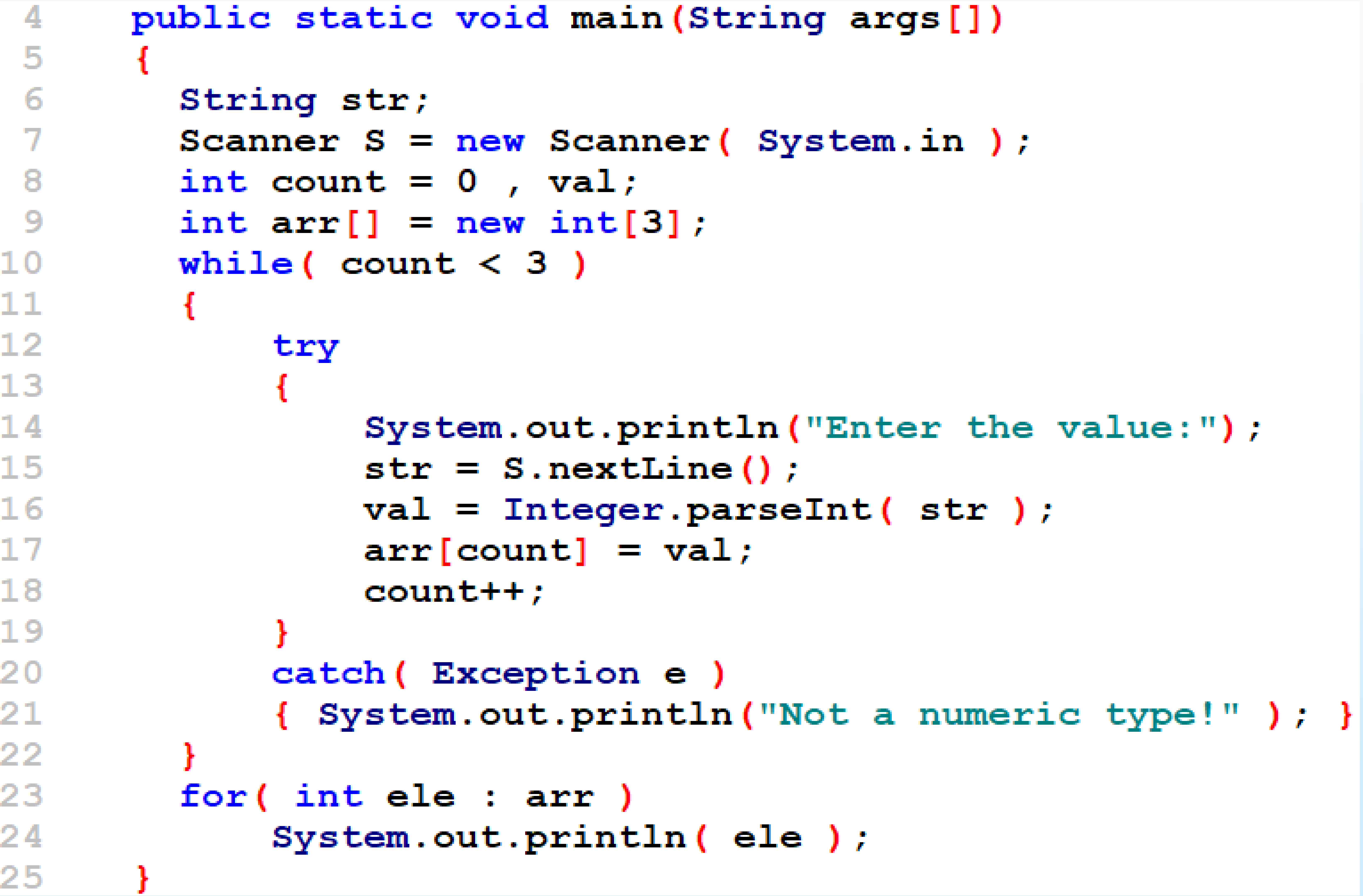
```java
class MyException extends Exception {
  private int detail;

  MyException(int a)
  {  detail = a; }

  public String toString()
  {  return "MyException[" + detail + "]"; }
}
class ExceptionDemo {
  static void compute(int a) throws MyException {
    System.out.println("Called compute(" + a + ")");
    if(a > 10)
      throw new MyException(a);
    System.out.println("Normal exit");
  }

  public static void main(String args[]) {
    try {
      compute(1);
      compute(20);
    } catch (MyException e)
      { System.out.println("Caught " + e); }
  }
}
```



```
Called compute(1)
Normal exit
Called compute(20)
Caught MyException[20]
```

Question

Given an integer array of size 3. Read the input from the keyboard and store in the array if it is of numeric type, otherwise display the exception message as "*Enter a numeric value*". Repeat this step until the user enters 3 numeric values.



```java
public static void main(String args[])
{
  String str;
  Scanner S = new Scanner( System.in );
  int count = 0 , val;
  int arr[] = new int[3];
  while( count < 3 )
  {
      try
      {
          System.out.println("Enter the value:");
          str = S.nextLine();
          val = Integer.parseInt( str );
          arr[count] = val;
          count++;
      }
      catch( Exception e )
      { System.out.println("Not a numeric type!" ); }
  }
  for( int ele : arr )
      System.out.println( ele );
}
```

The End