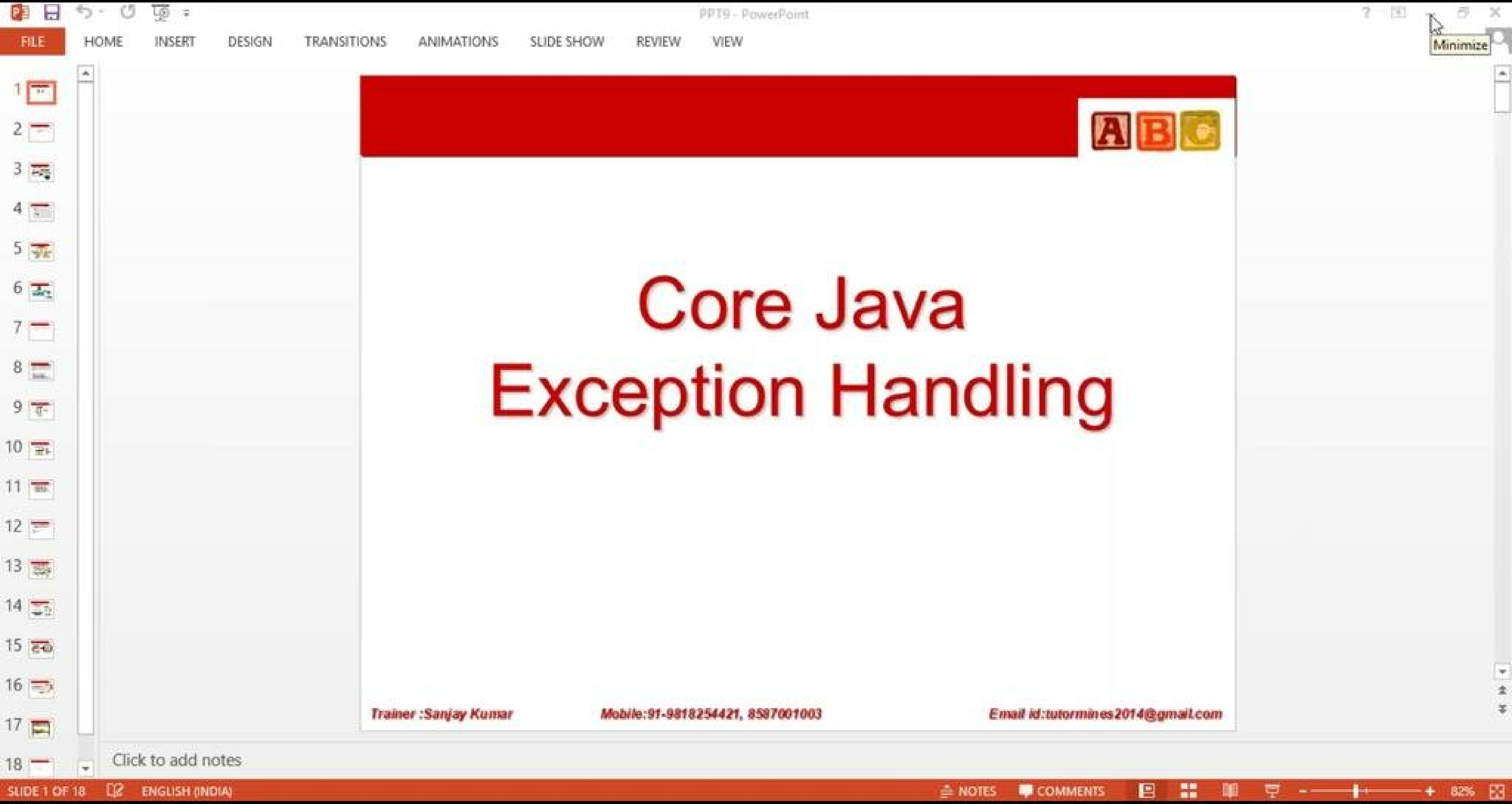


Core Java Exception Handling

Trainer :Sanjay Kumar Mobile:91-9818254421, 8587001003 Email id:tutormines2014@gmail.com

Core Java
Exception Handling

Core Java
Exception Handling

What is an exception?

```
class ExceptionRaised {
     /** Constructor. */
     protected ExceptionRaised() {
     }
    /**
     * This method generates an exception.
     * @param operand1 is numerator in division
     * @param operand2 is denominator in division
     * @return the remainder of the division.
     */
    static int calculate(final int operand1, final int operand2) {
        int result = operand1 / operand2;     // user defined method
        return result;
/**
     * Sole entry point to the class and application.
     * @param args array of String arguments.
     */
public static void main(final String[] args) {
          = new ExceptionRaised();
   try {
        /* variable result is defined to store the result.
        int result = obj.calculate(9, 0);
        System.out.println(result);
      Exception e) {     // Exception object
   System.err.println("Exception occurred  :" + e.toString());
   e.printStackTrace();
  }
}
}
```

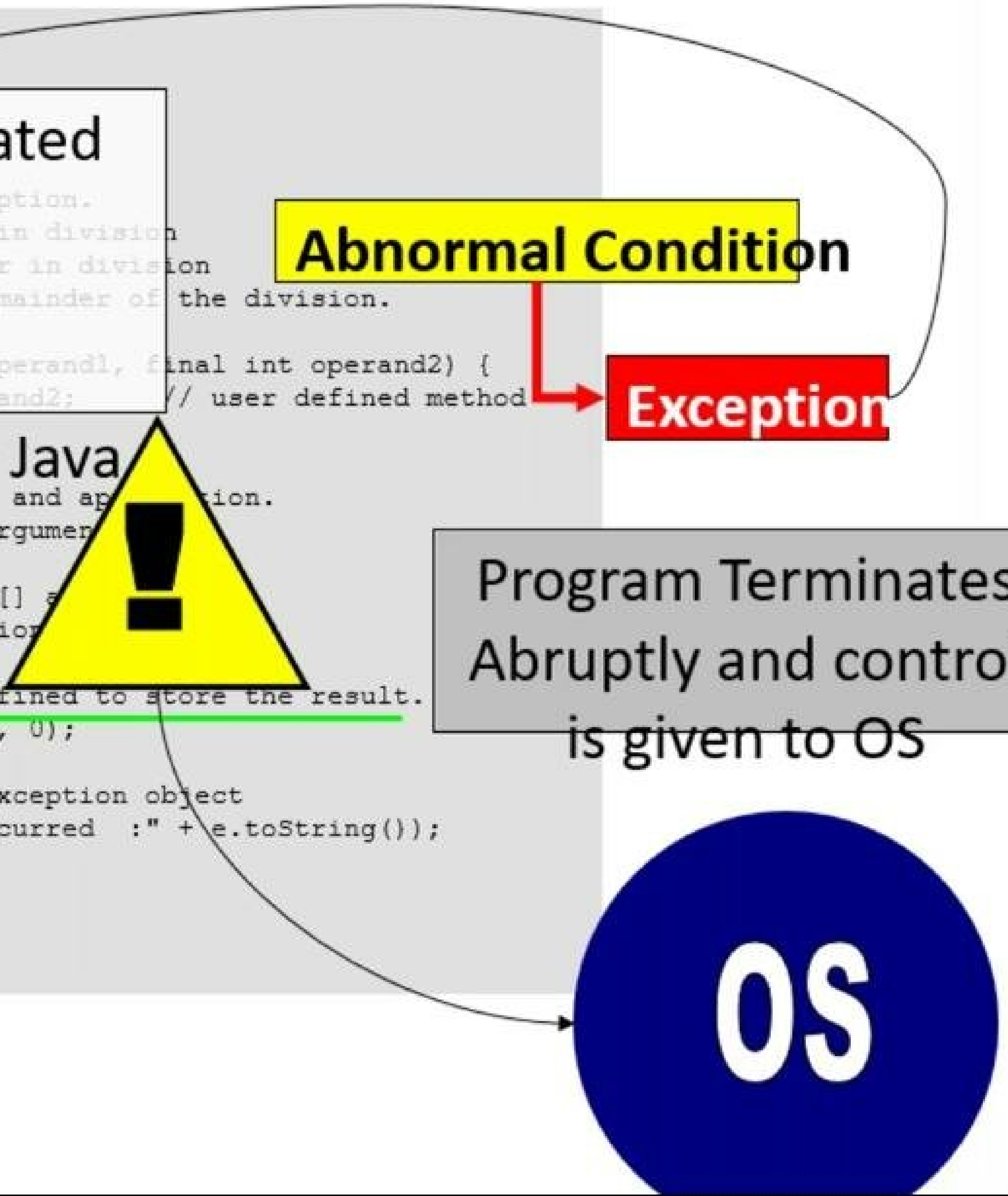
- Can be generated manually in a program

  OR

- Generated by Java Runtime

Error Handling Benefits

- Fixes Error
- Prevents automatic termination



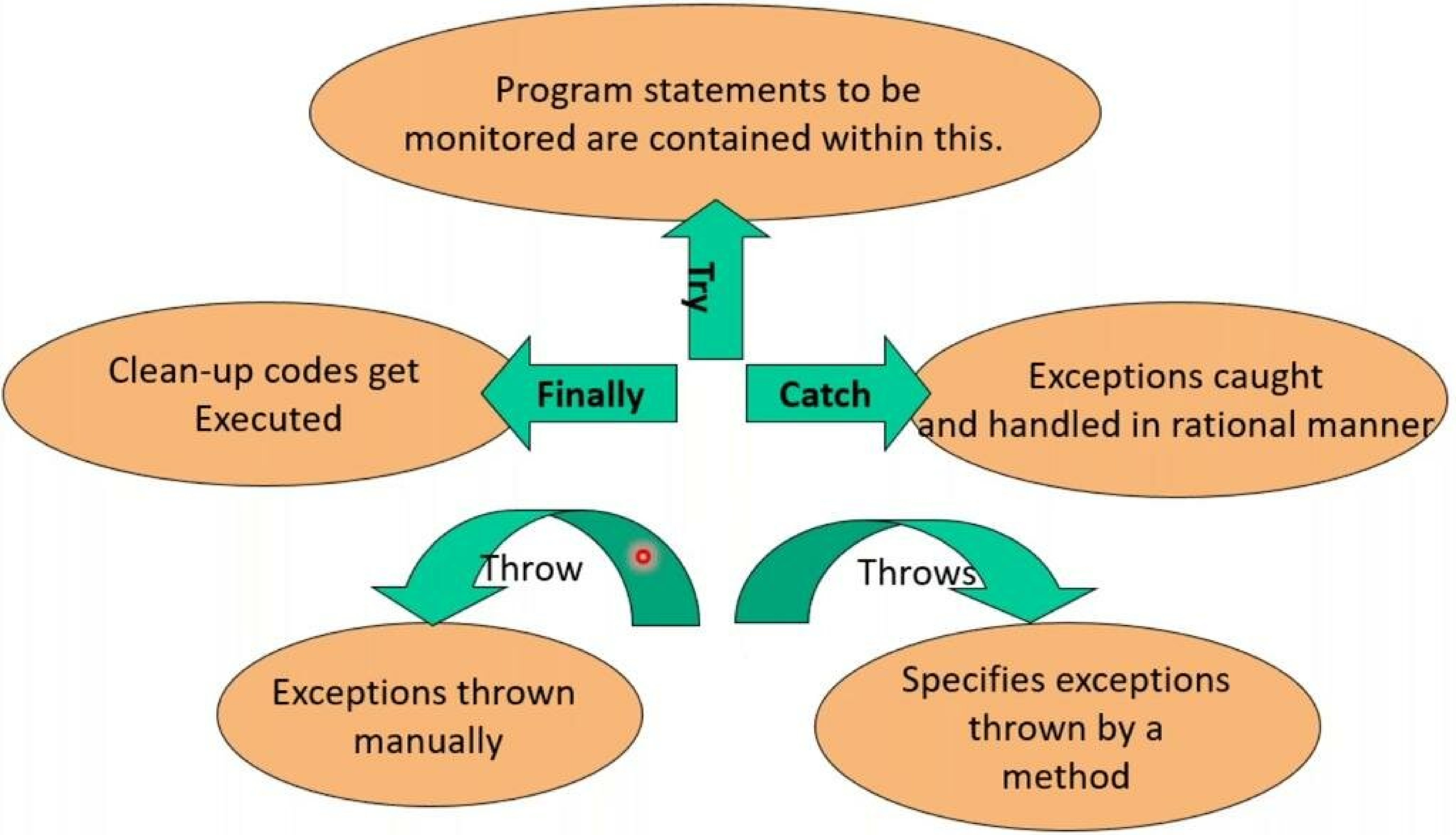
Program Terminates Abruptly and control is given to OS

Handling Exceptions 2-1

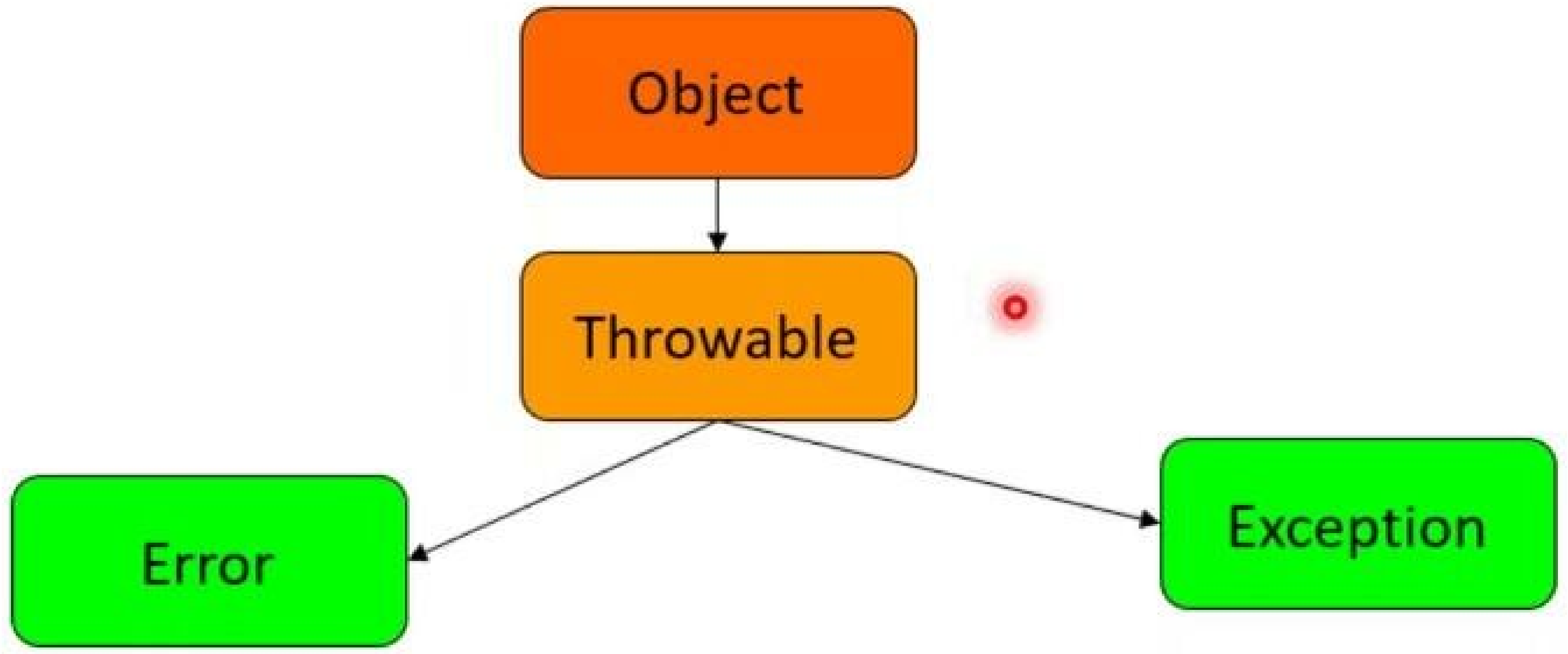
A pseudo code handling a runtime error

```
.........
IF B IS ZERO GO TO ERROR
C = A / B
PRINT C
GO TO EXIT

ERROR:

BLOCK THAT  {
HANDLES THE { "CODE CAUSING ERROR DUE TO DIVISION BY
            {  ZERO"
EXCEPTION


EXIT:
END
```

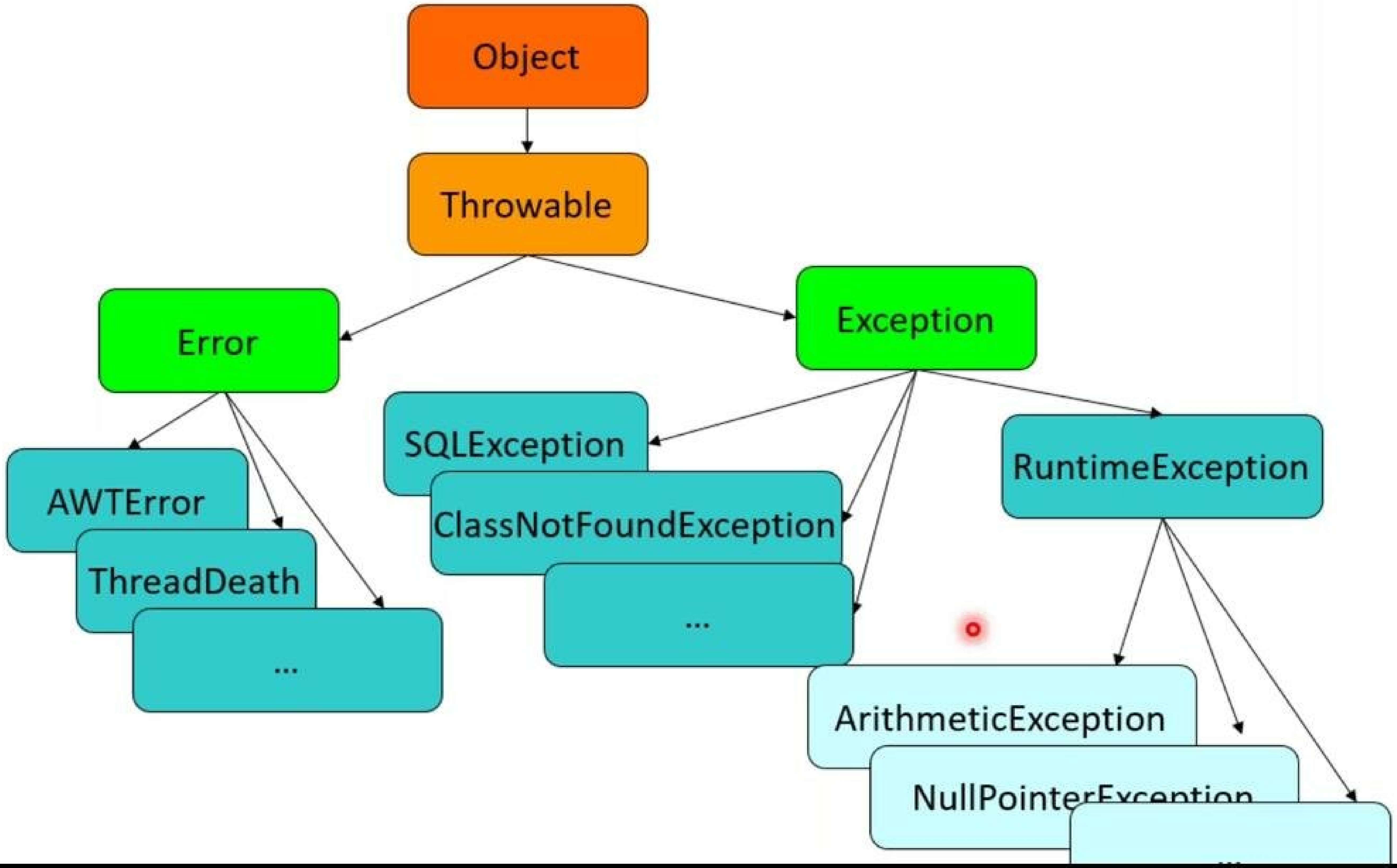
Handling Exceptions 2-2


Program statements to be monitored are contained within this.
Try
Finally
Clean-up codes get Executed
Catch
Exceptions caught and handled in rational manner
Throw
Exceptions thrown manually
Throws
Specifies exceptions thrown by a method

Hierarchy of Exception classes 2-1



Hierarchy of Exception classes 2-1

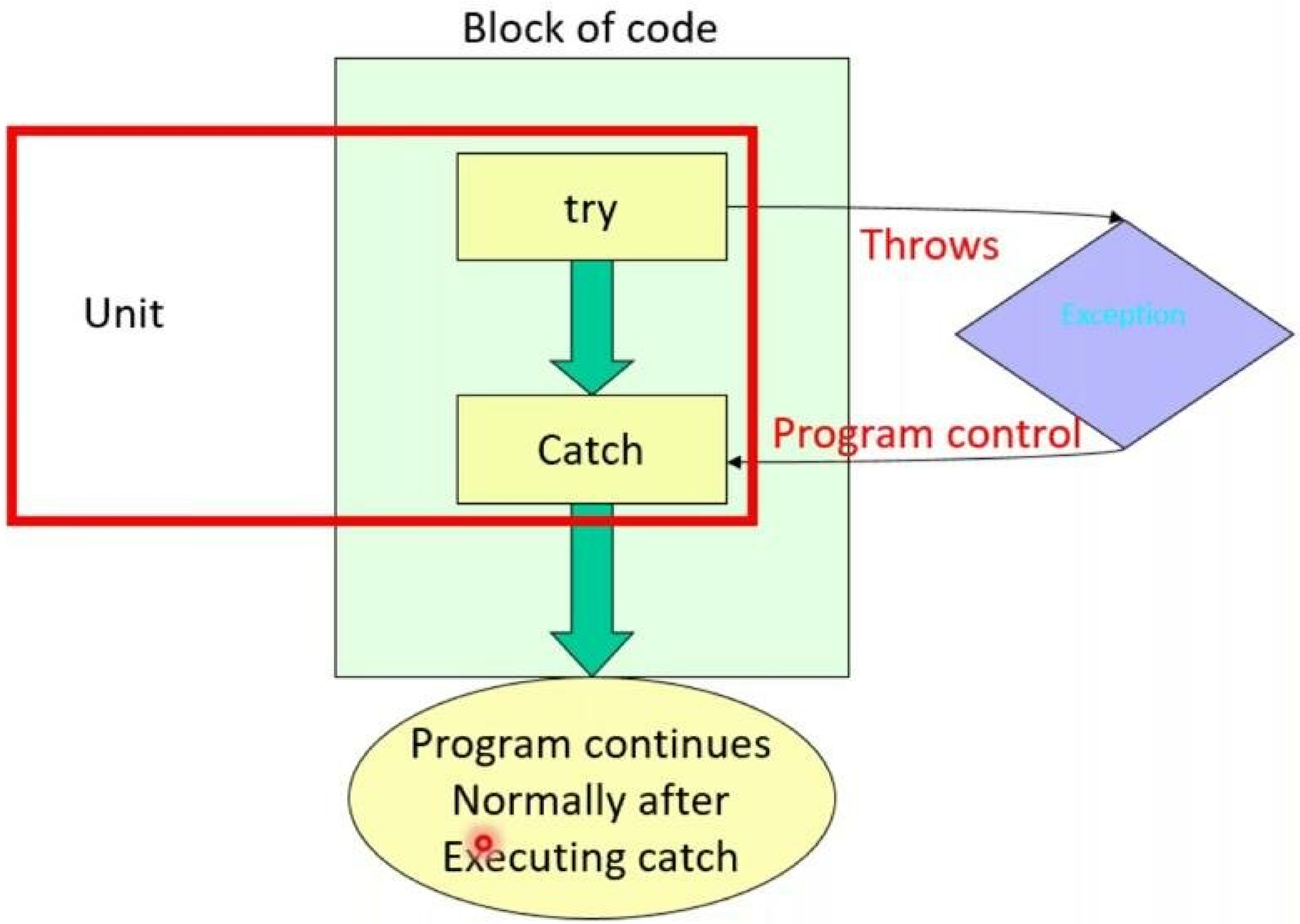


Hierarchy of Exception classes 2-2

- All exception types are subclasses of the built-in class `Throwable`.
- Throwable has two subclasses, they are:
  - Exception: To handle exceptional conditions that user programs should catch.
    - An important subclass of Exception is `RuntimeException`, which includes division by zero and invalid array indexing.
  - Error: To handle exceptional conditions that are not expected to be caught under normal circumstances. i.e. `stack overflow`

Exception	Root class of exception hierarchy
RuntimeException	Base class for many java.lang exceptions
ArithmeticException	Arithmatic error condition, such as divide by zero
IllegalArgumentException	Method received illegal argument
ArrayIndexOutOfBoundsE xception	Array size is less or greater than actual array size
NullPointerException	Attempt to access *null* object member
SecurityException	Security settings do not allow operation
ClassNotFoundException	Unable to load requested class
NumberFormatException	Invalid conversion of a string to a numeric float
IOException	Root class for I/O exceptions
FileNotFoundException	Unable to locate a file
EOFException	End of file
IllegalAccessException	Access to a class denied
NoSuchMethodException	Requested method does not exist
InterruptedException	Thread interrupted

try and catch blocks 2-1



Multiple catch blocks

- Single piece of code can generate more than one error.
- When an exception is thrown, each `catch` statement is inspected in order, and the first one whose type matches that of the exception is executed.
- After one `catch` statement executes, the others are bypassed.

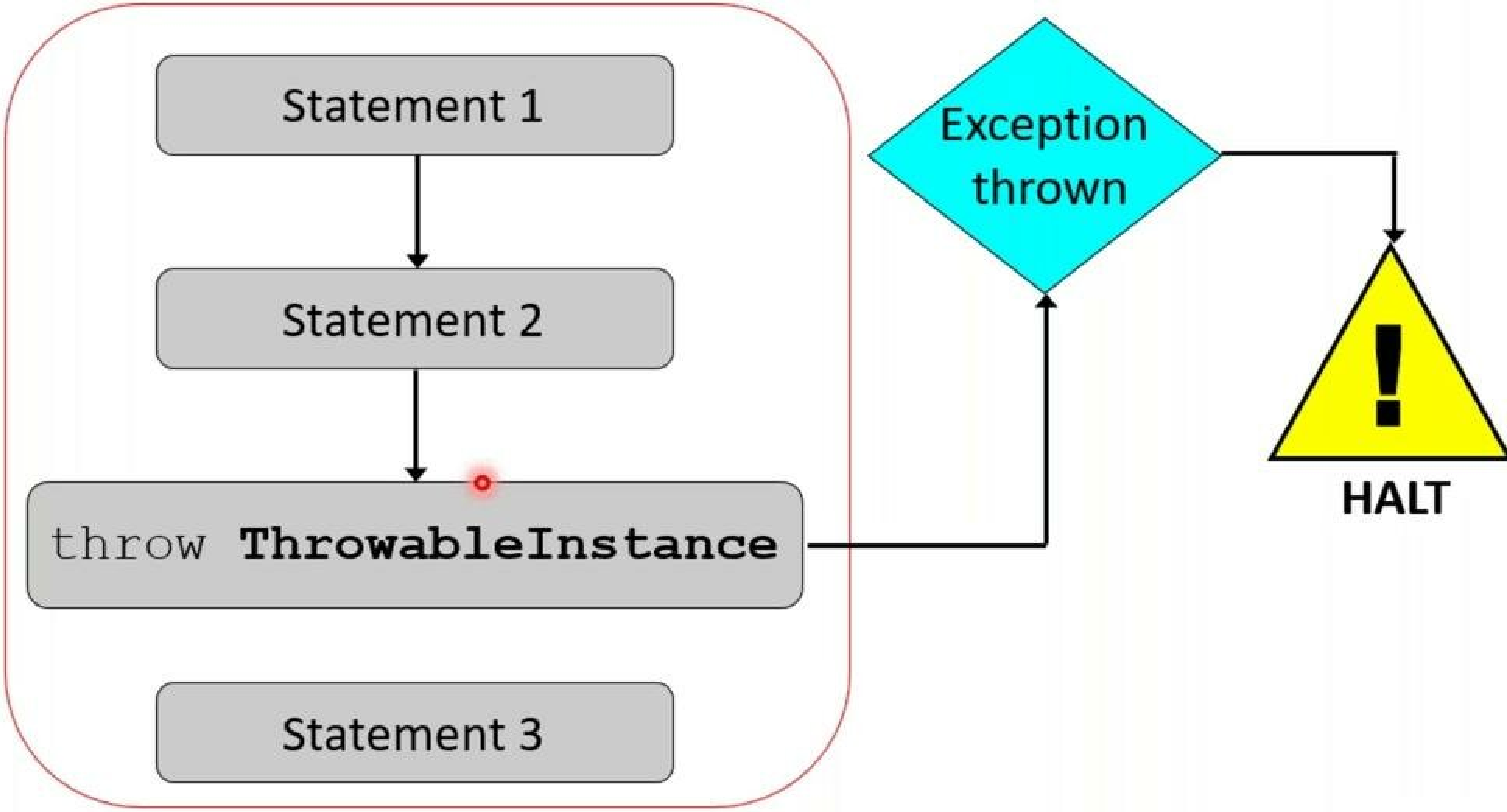
```
 .........
 try{
  }
  catch(ArrayIndexOutOfBoundsException e) {
  }
  catch(Exception e) {
  }
  ..........
```

Nested try - catch blocks

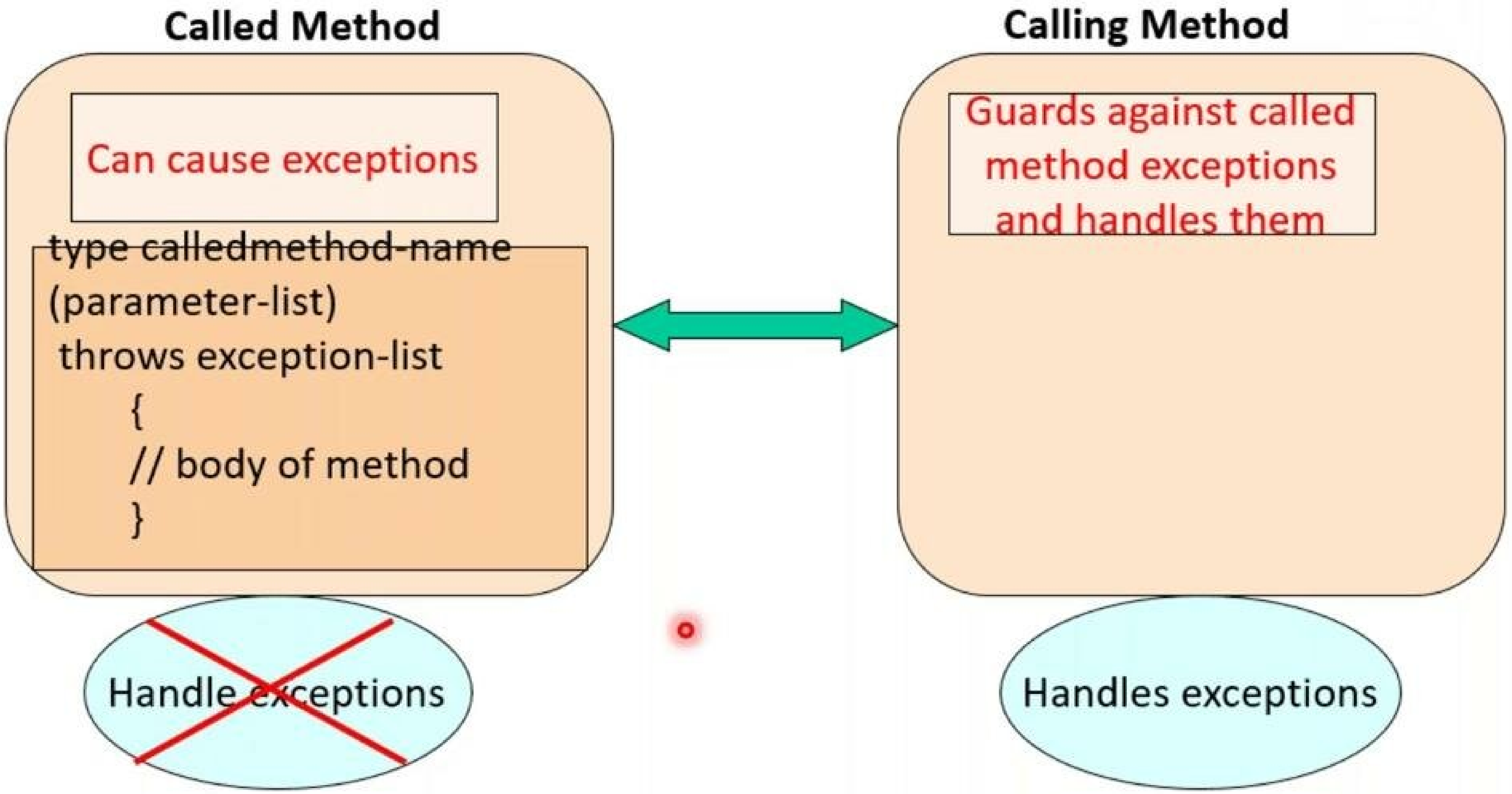
```
* All Rights ReservThis class demonstrate the nested try-catch statements.
  * class NestedException {    /* Constructor. */
    protected NestedException() {
    }  /** This method test the format of the number
     * @param argument is used to store the value of args.
     */
    public test(String argumnet) {
         try {
            int num = args.length;
            /* Nested try block. */
            try {
           int numValue = Integer.parseInt(args[0]);
                System.out.println("The square of " + args[0] + "is "
                +  numValue * numValue);
            } catch (NumberFormatException nb) {
            /** Displaying the appropriate message, if exception
             *  has occurred.
             */
            System.out.println("Not a number! ");
        }
      } catch (ArrayIndexOutOfBoundsException ne) {
           System.out.println("Please enter the number!!!");
    }
public static void main(final String[] args) {
       NestedException obj = new NestedException();
       obj.test(args[0]);
    } }
```

Using throw & throws 2-1

Executable Program Statements



Using throw & throws 2-2



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

User defined exceptions 2-1

Block of code

```
public class MyException extends Exception
 { public MyException() {}
 public MyException(String message) { super(message);
}
 }
public static void main(String args[]) {
try {
throw new MyException
("Arg Length: " + args.length);
 }
catch (MyException e) { e.printStackTrace(); } }
```

Can generate exception which is not a part of built in exceptions.

- Hence, User defined Exceptions Came into use.
- Subclass of exception class.
- Can use all methods of Throwable class.

User defined exceptions 2-2

- *Creating user defined exception.*

Demonstration: Example 6

User defined exceptions 2-2

- *Creating user defined exception.*
- *Sub-classing the Exception class.*

```
class ArraySizeException extends NegativeArraySizeException  {
     /** Constructor. */
     ArraySizeException()  {
         super("You have passed illegal array size");
    }
}
```

Demonstration: Example 6

User defined exceptions 2-2

```
 class ExceptionClass {
ExceptionClass(final int val) {
        size = val;
        try {
            checkSize();
        } catch (ArraySizeException e) {
            System.out.println(e);
        }
    }
   /** Declaring variable to store size and elements of an array. */
    private int size;
    private int[] array;
    /** Method to check the length of an array.
      *  @ throws an ArraySizeException.
      */
    public void checkSize() throws ArraySizeException {
        if (size < 0) {
            throw new ArraySizeException();
    }
        array = new int[3];
        for (int count = 0; count < 3; count++) {
        array[count] = count + 1;
        }
    }
```