JAVA

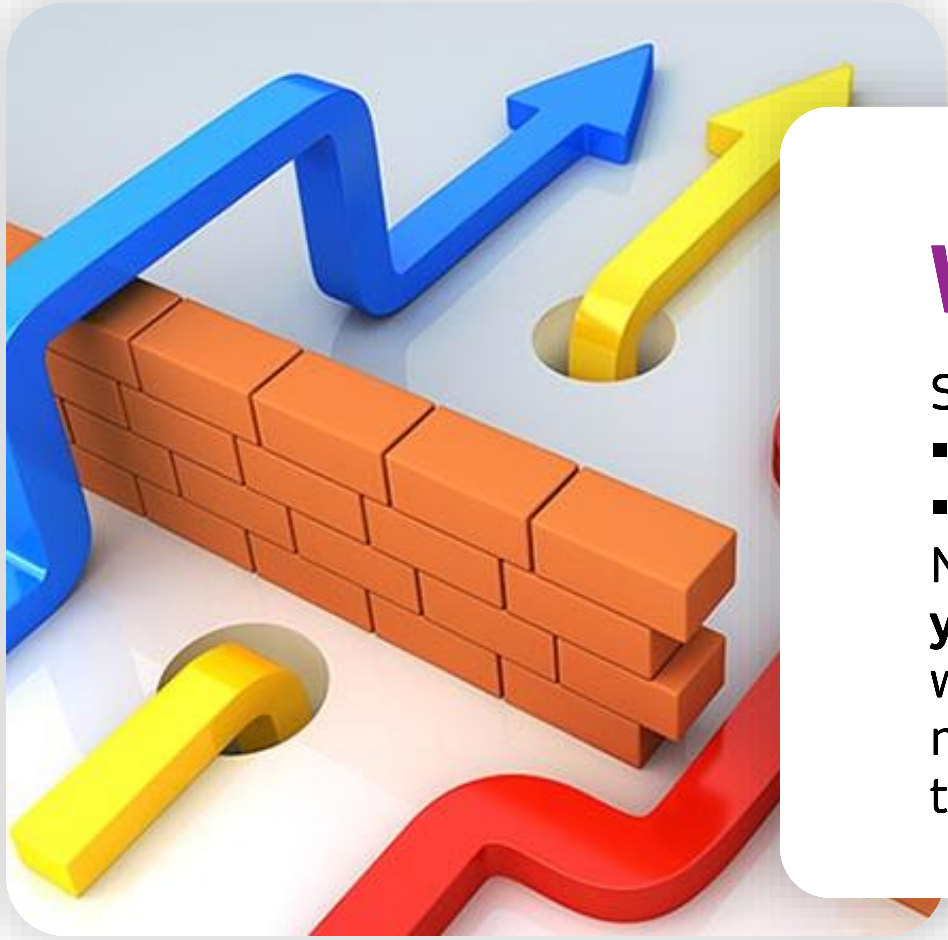
Exception Handling

Jayaprakash M
CSE 3rd Year

What is Exception?

a person or thing that is excluded from a general statement or **does not follow a rule**.

Exception: an abnormal condition that arises in a code sequence at run time/run time error



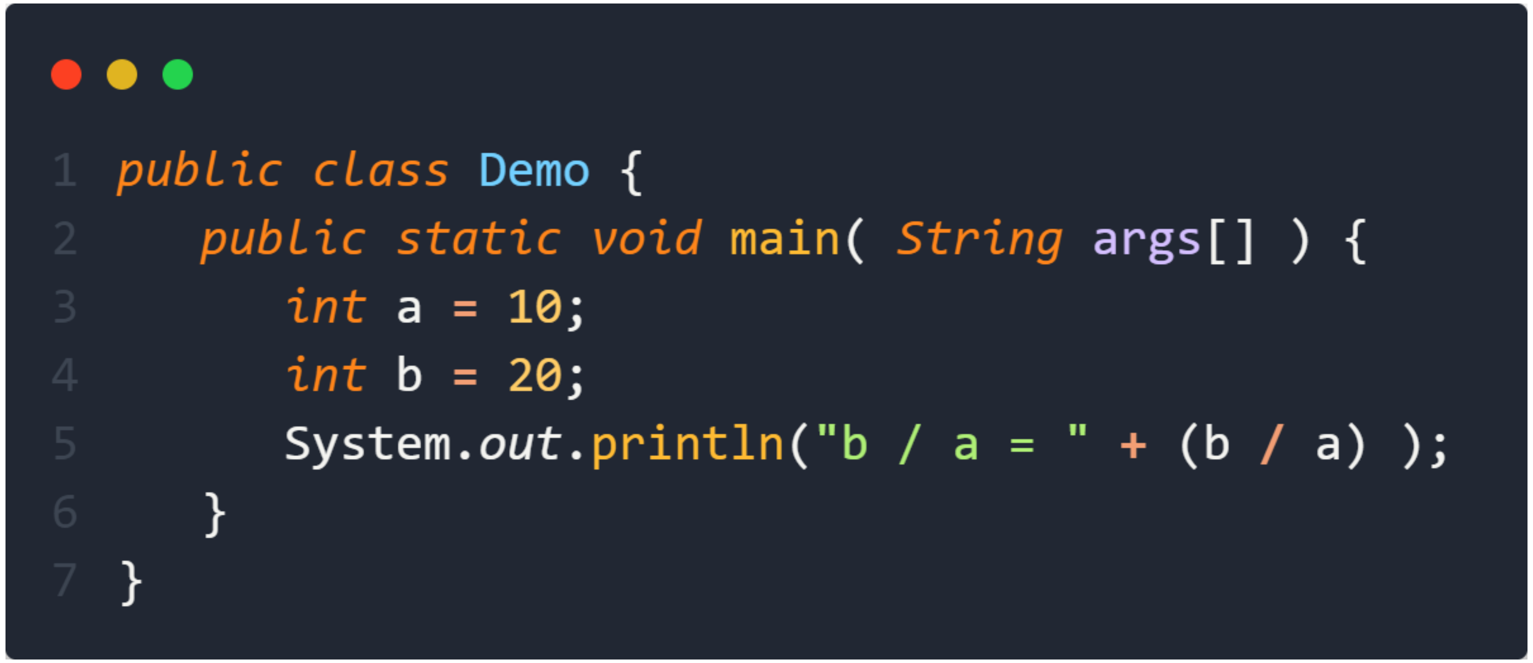
Why Handling?

Stuff happens.

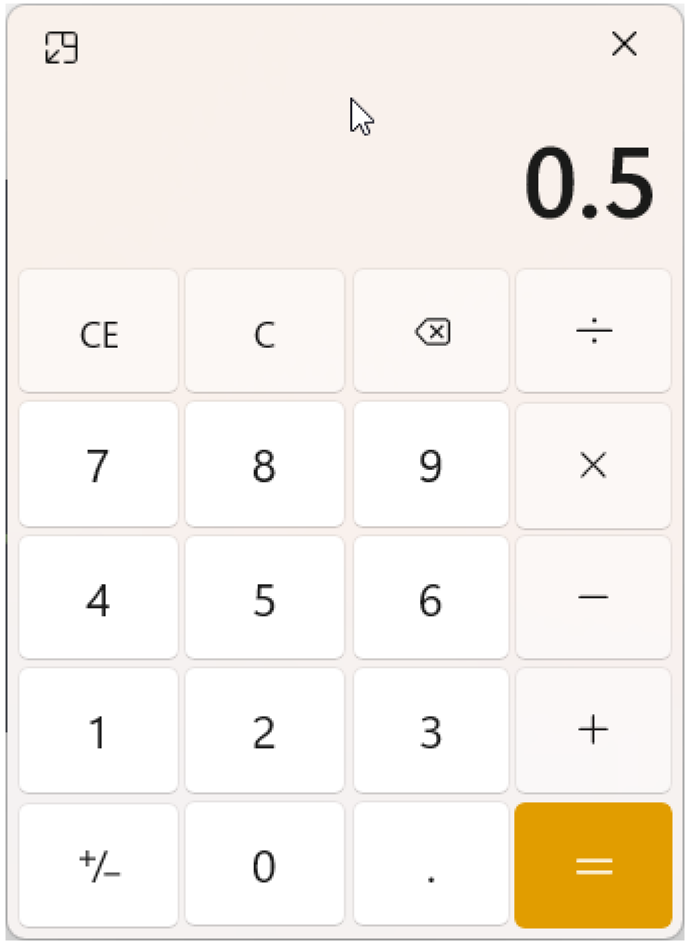
- **The file isn't there.**
- **The server is down.**

No matter how good a programmer you are, **you can't control everything**. Things can go wrong. Very wrong. When you write a risky method, you need code to handle the bad things that might happen.

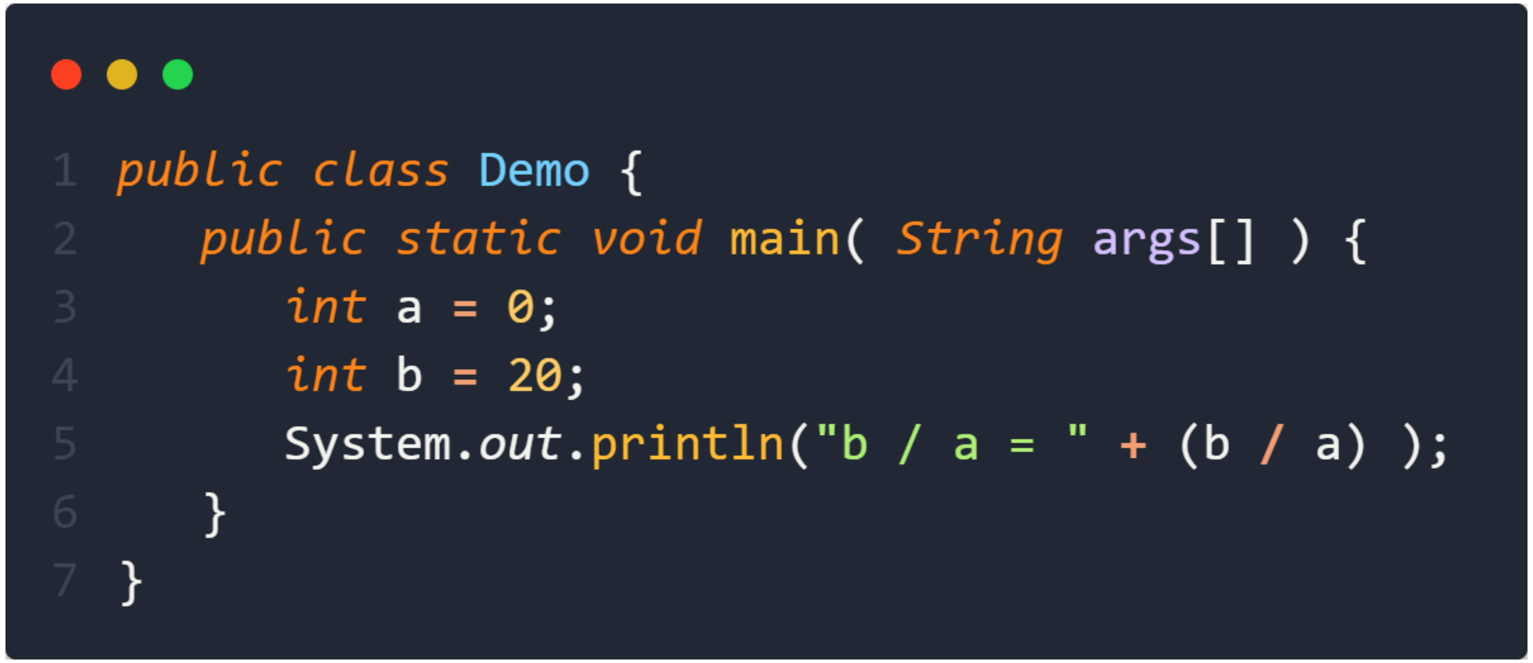
Applications Exception Handling



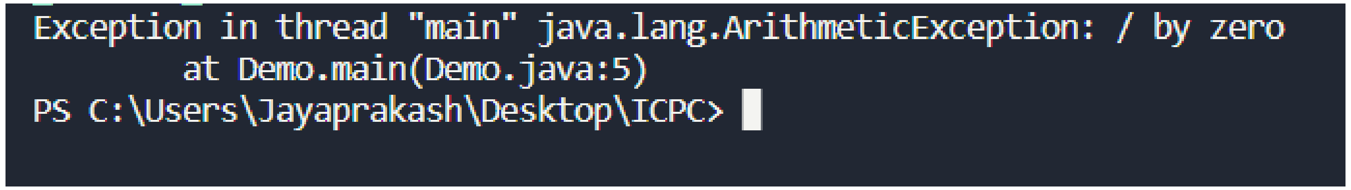
```java
public class Demo {
    public static void main( String args[] ) {
        int a = 10;
        int b = 20;
        System.out.println("b / a = " + (b / a) );
    }
}
```



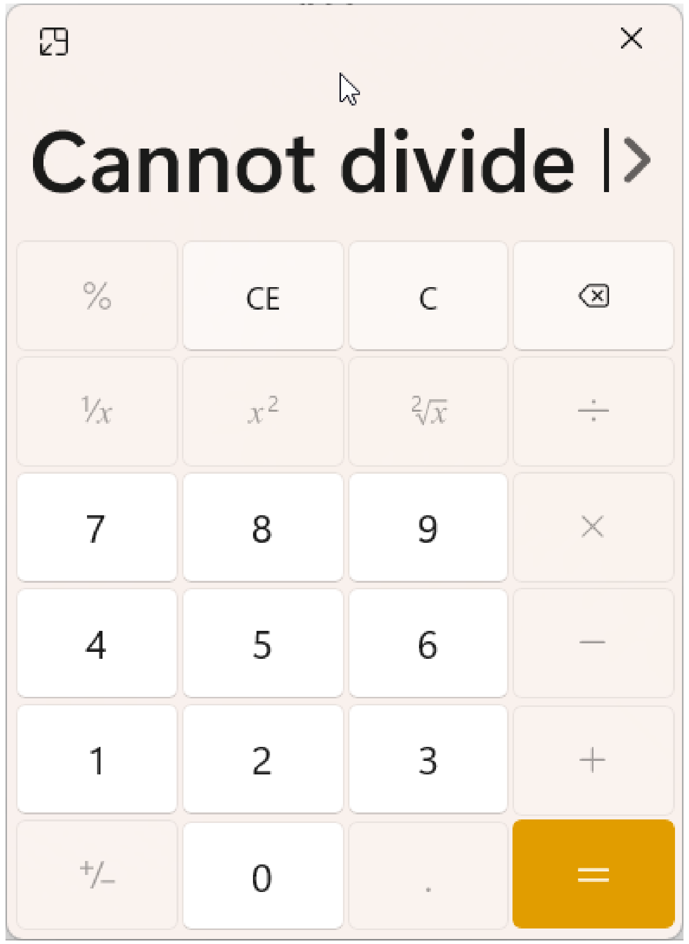
Applications Exception Handling



```java
public class Demo {
    public static void main( String args[] ) {
        int a = 0;
        int b = 20;
        System.out.println("b / a = " + (b / a) );
    }
}
```

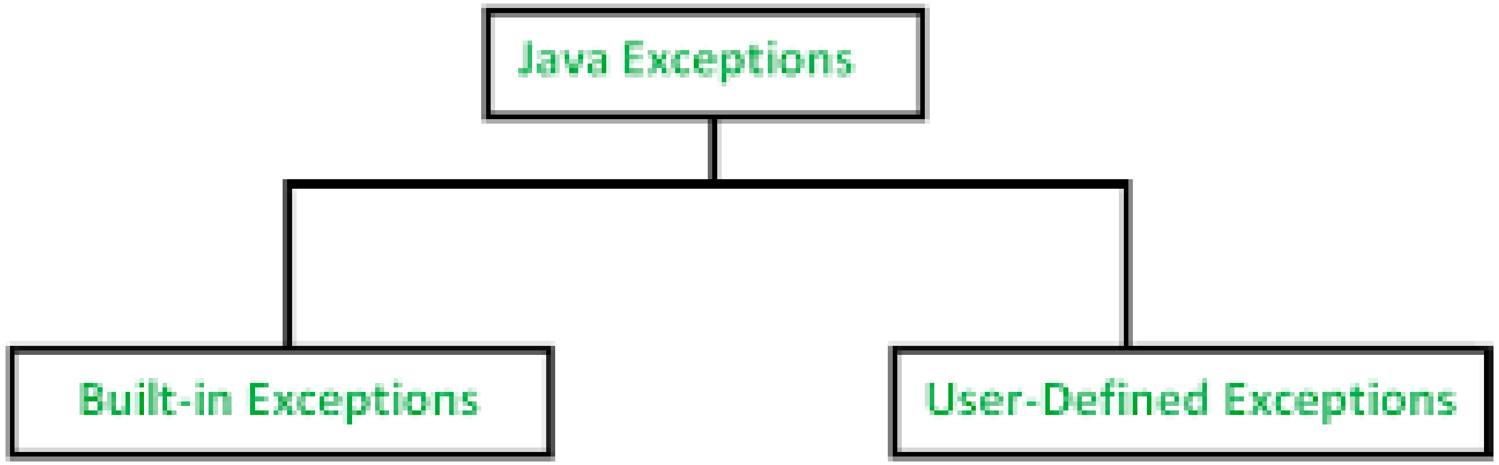


```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Demo.main(Demo.java:5)
PS C:\Users\Jayaprakash\Desktop\ICPC> 
```



Cannot divide

Types of Exceptions



generated by the Java run-time system (relate to fundamental errors that violate the rules of the Java language)

Manually generated (typically used to report some error condition to the caller of a method)

Terms of Exceptions

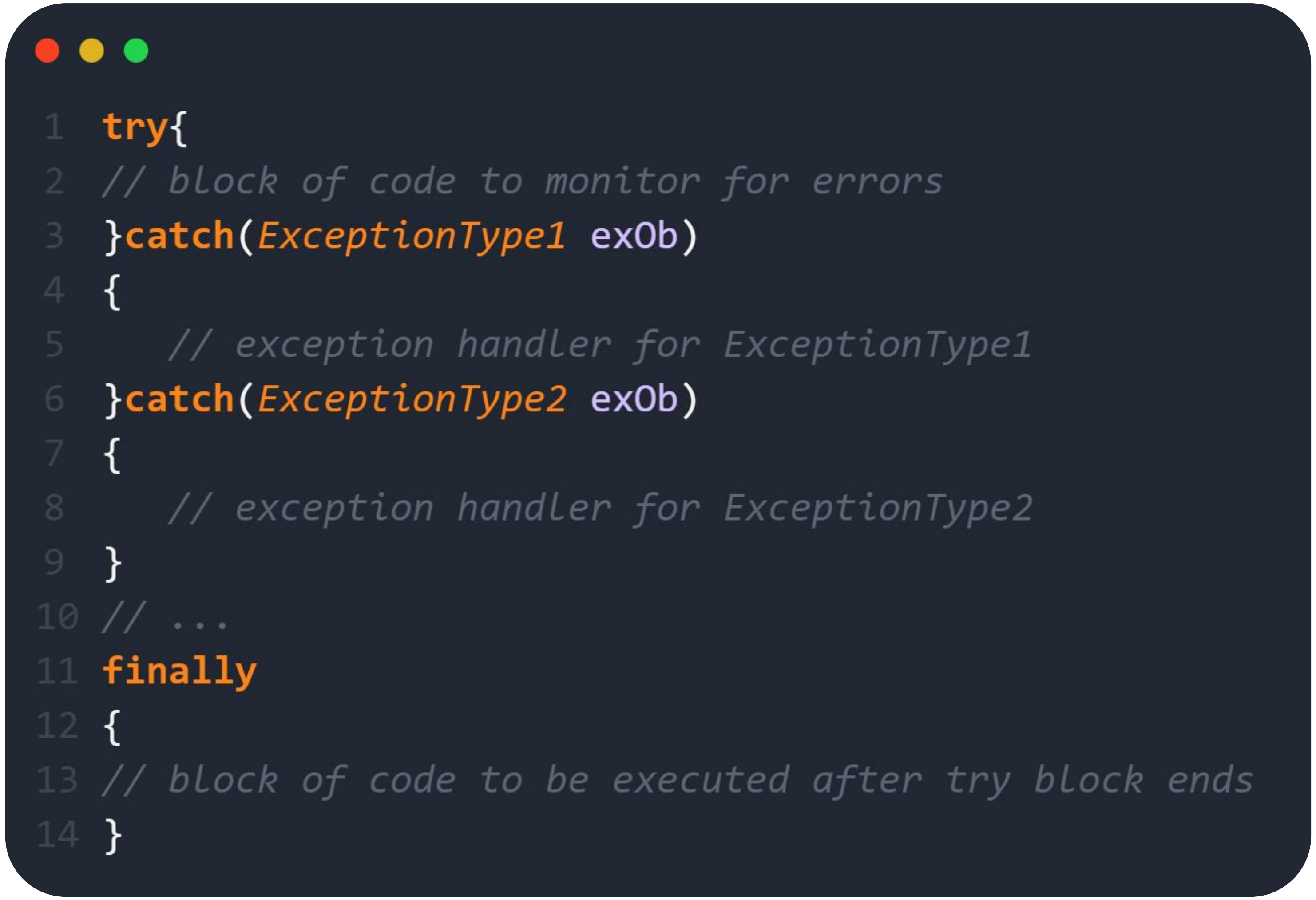
```
try {
   // do risky thing


} catch(Exception e) {
   // try to recover
}
```

It's just like declaring a method argument.

This code runs only if an Exception is thrown.

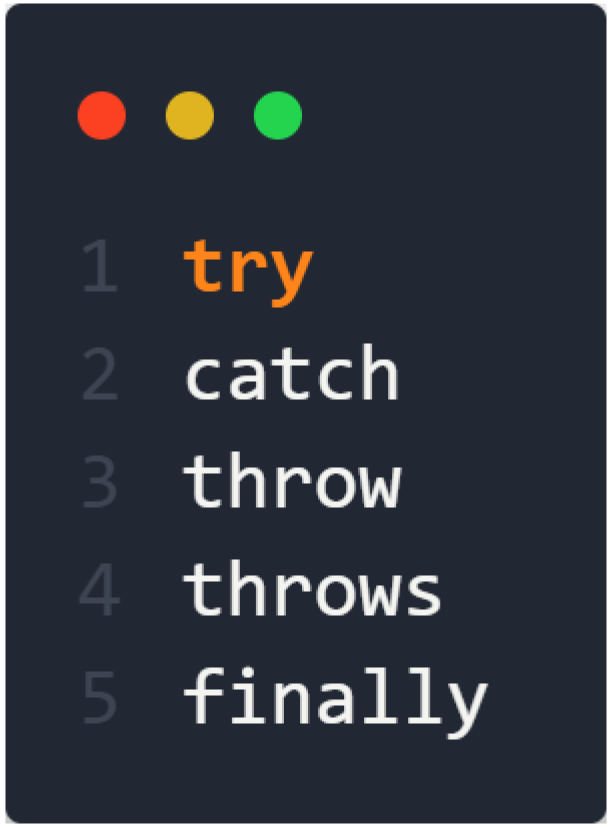
I'm gonna TRY this risky thing, and I'm gonna CATCH myself if I fall.

Basic Structure of Exception Handling



```
try{
// block of code to monitor for 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
}
```

Terms of Exceptions


```
try
catch
throw
throws
finally
```

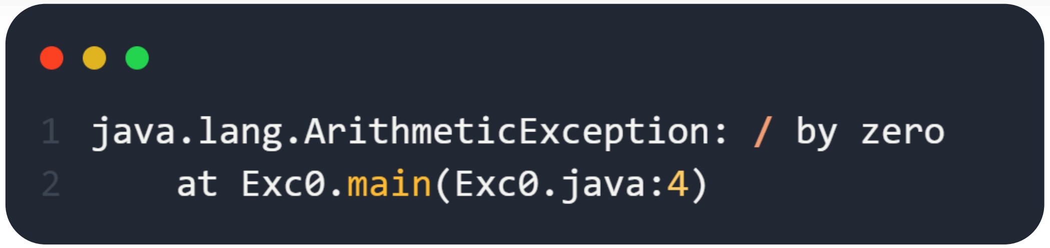
Uncaught Exceptions

What will Happen if I not capture any Exceptions?

No Problem, Java has its **default handler** from (java-runtime-system)

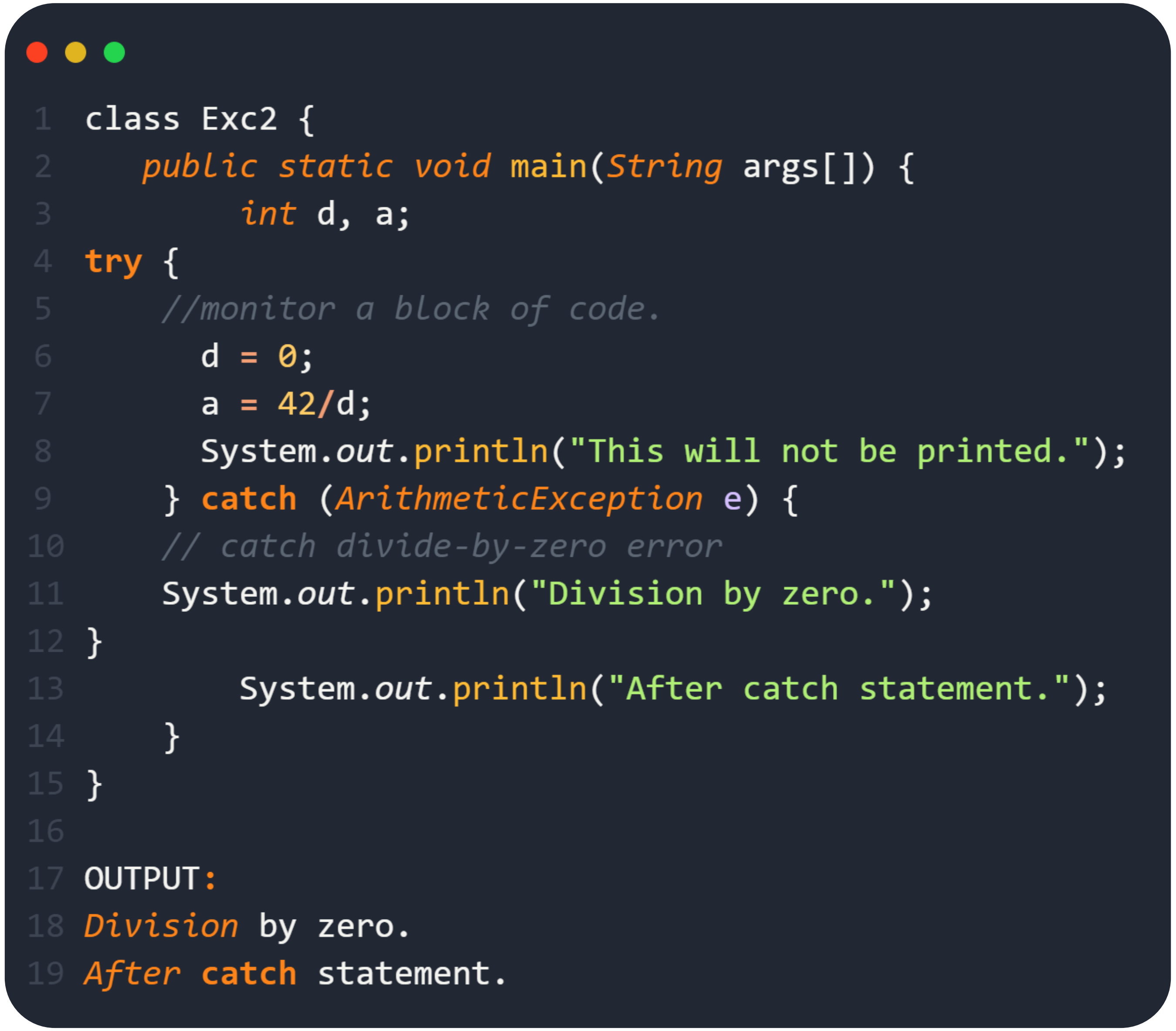
It will cause 3 things

1. Displays a String Describing Exception
2. Prints the Stack trace from error source
3. Terminates the Program



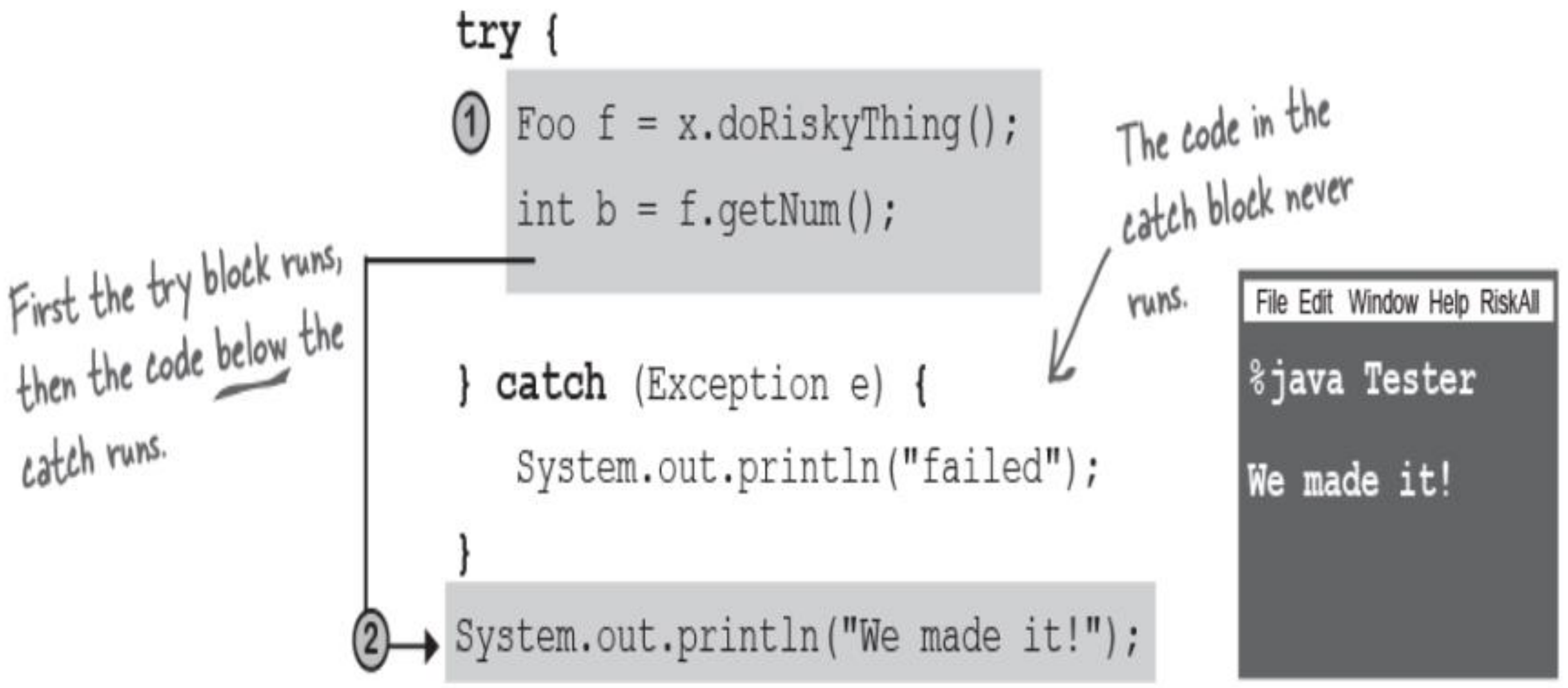
```
java.lang.ArithmeticException: / by zero
    at Exc0.main(Exc0.java:4)
```

TRY-CATCH



```
class Exc2 {
   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.");
    } 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.
```

TRY-CATCH



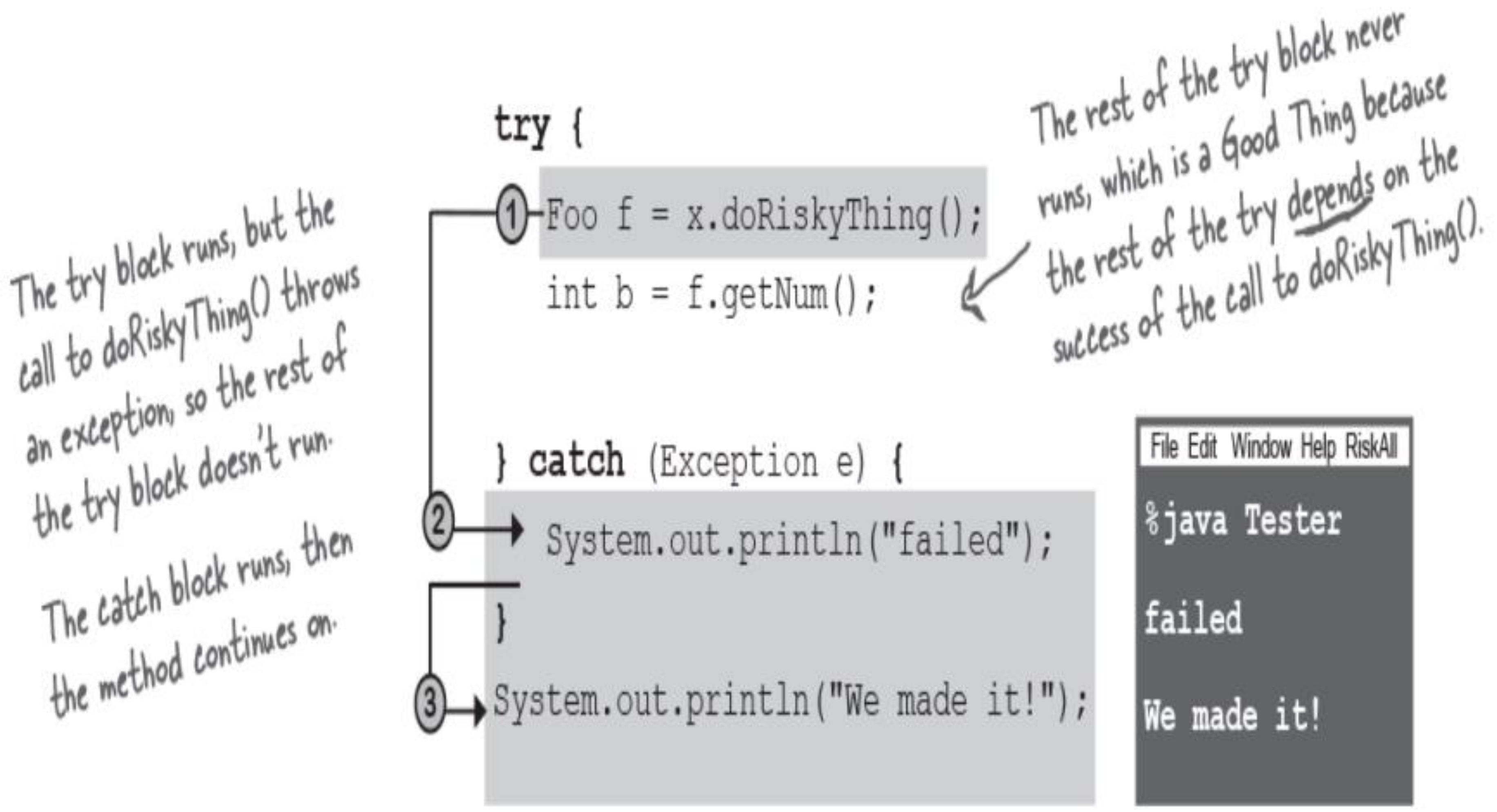
```
try {
① Foo f = x.doRiskyThing();
  int b = f.getNum();

} catch (Exception e) {
  System.out.println("failed");
}
② System.out.println("We made it!");
```

First the try block runs, then the code below the catch runs.

The code in the catch block never runs.

```
%java Tester

We made it!
```

TRY-CATCH



```
try {
① Foo f = x.doRiskyThing();
  int b = f.getNum();

} catch (Exception e) {
② System.out.println("failed");
}
③ System.out.println("We made it!");
```

The try block runs, but the call to doRiskyThing() throws an exception, so the rest of the try block doesn't run.

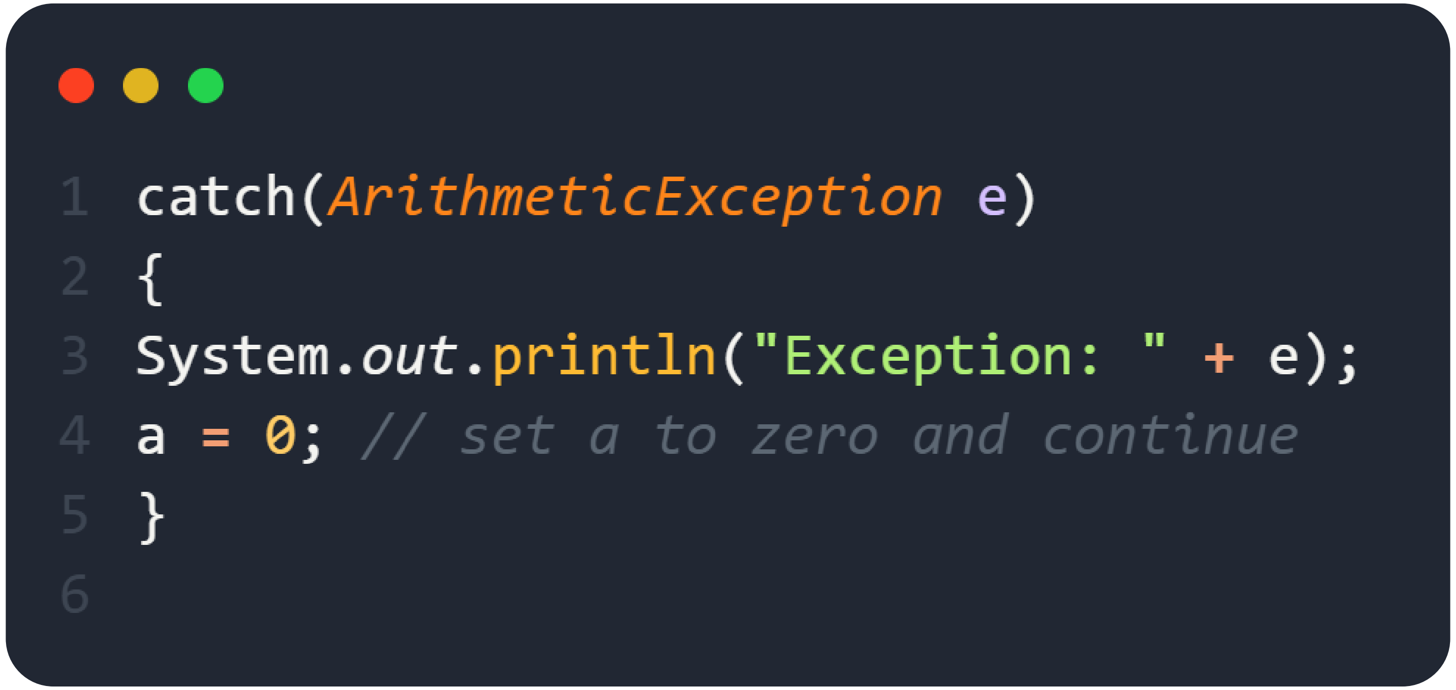
The catch block runs, then the method continues on.

The rest of the try block never runs, which is a Good Thing because the rest of the try depends on the success of the call to doRiskyThing().

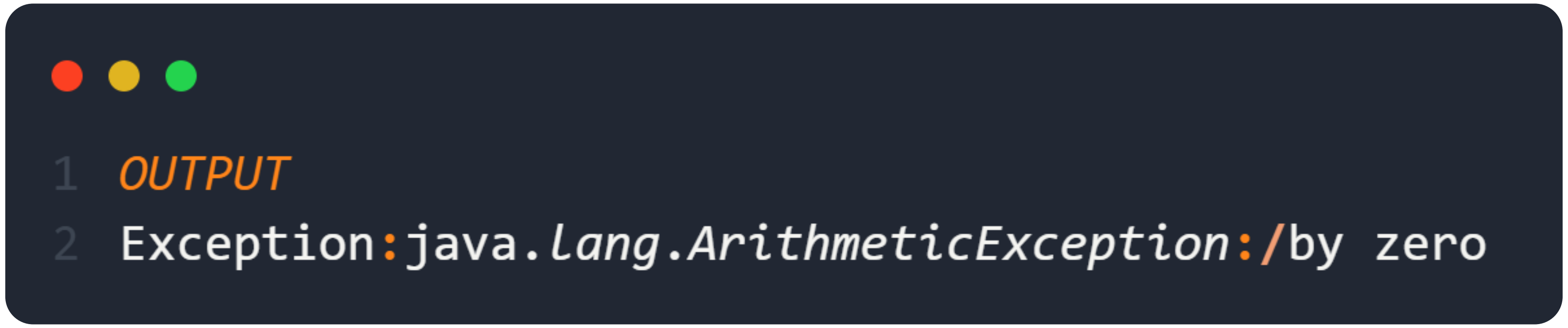
```
%java Tester
failed
We made it!
```

TRY-CATCH-IN-ONE-PIC

SEE-EXCEPTION Description

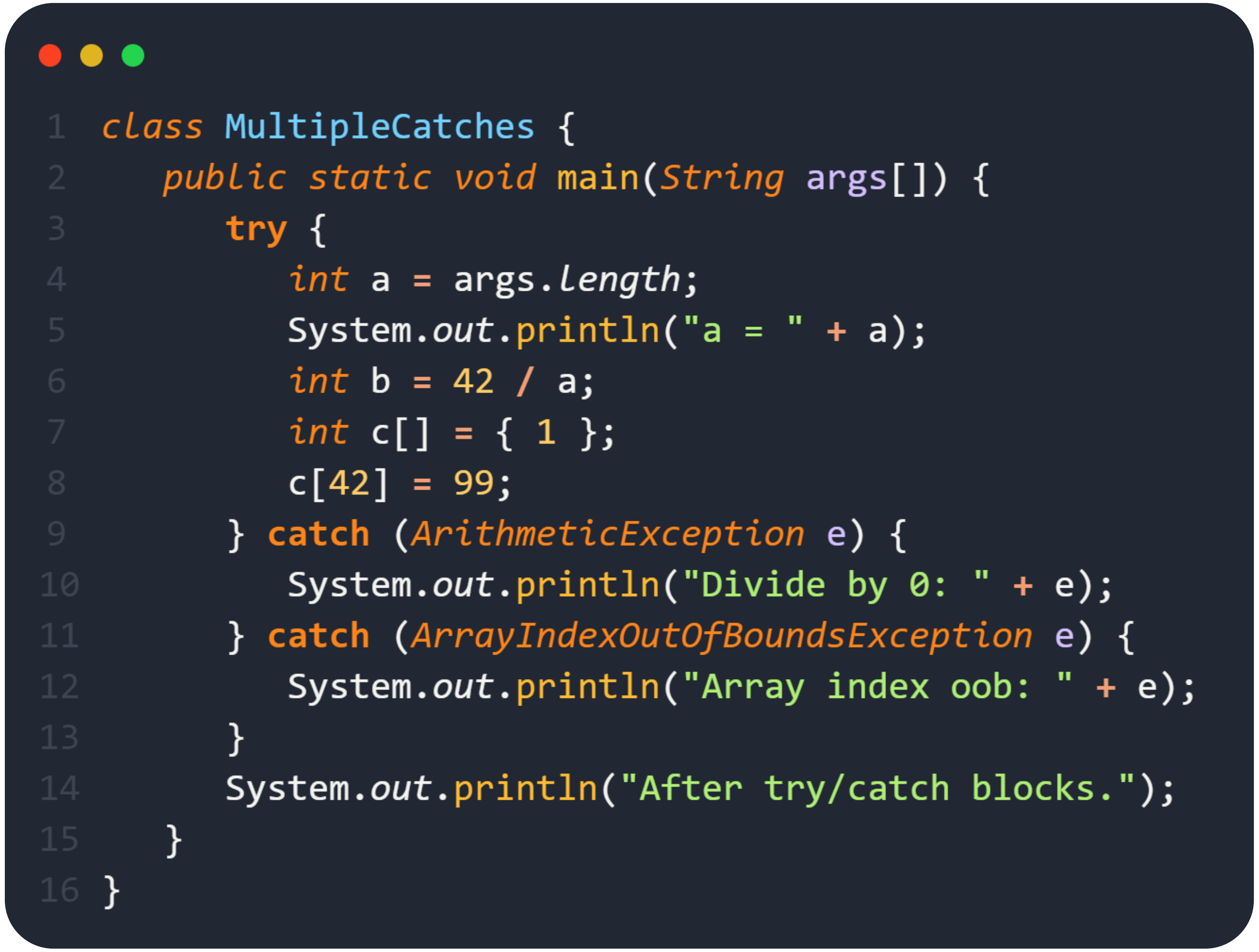


```java
catch(ArithmeticException e)
{
System.out.println("Exception: " + e);
a = 0; // set a to zero and continue
}

```

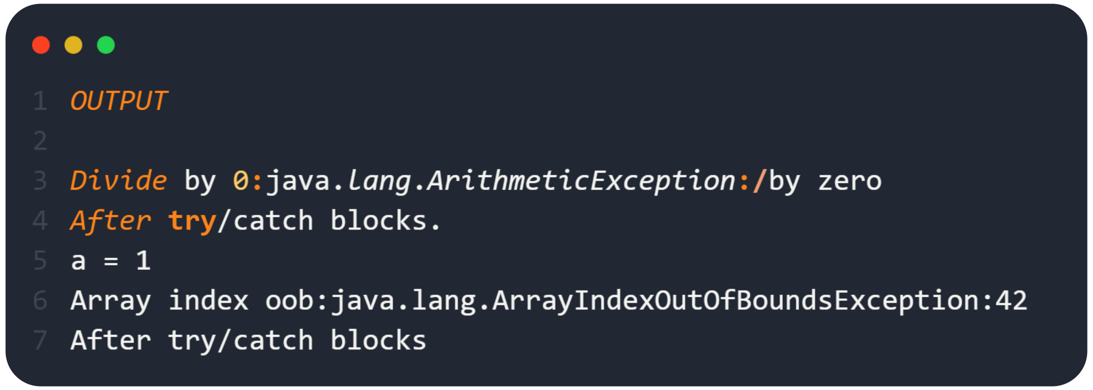


```
OUTPUT
Exception:java.lang.ArithmeticException:/by zero
```

MULTIPLE-CATCHES

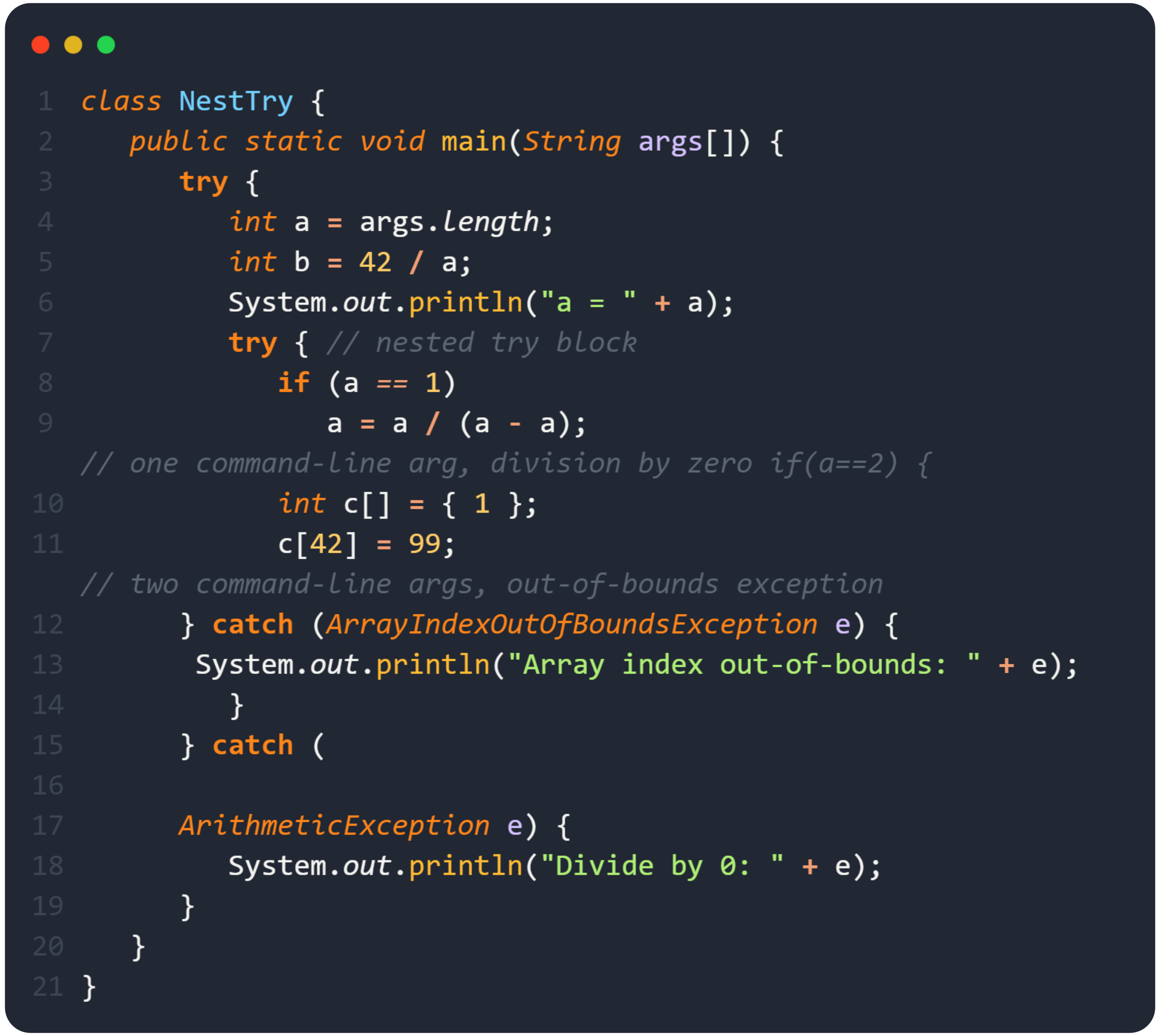


```java
class MultipleCatches {
   public static void main(String args[]) {
      try {
         int a = args.length;
         System.out.println("a = " + a);
         int b = 42 / a;
         int c[] = { 1 };
         c[42] = 99;
      } catch (ArithmeticException e) {
         System.out.println("Divide by 0: " + e);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Array index oob: " + e);
      }
      System.out.println("After try/catch blocks.");
   }
}
```

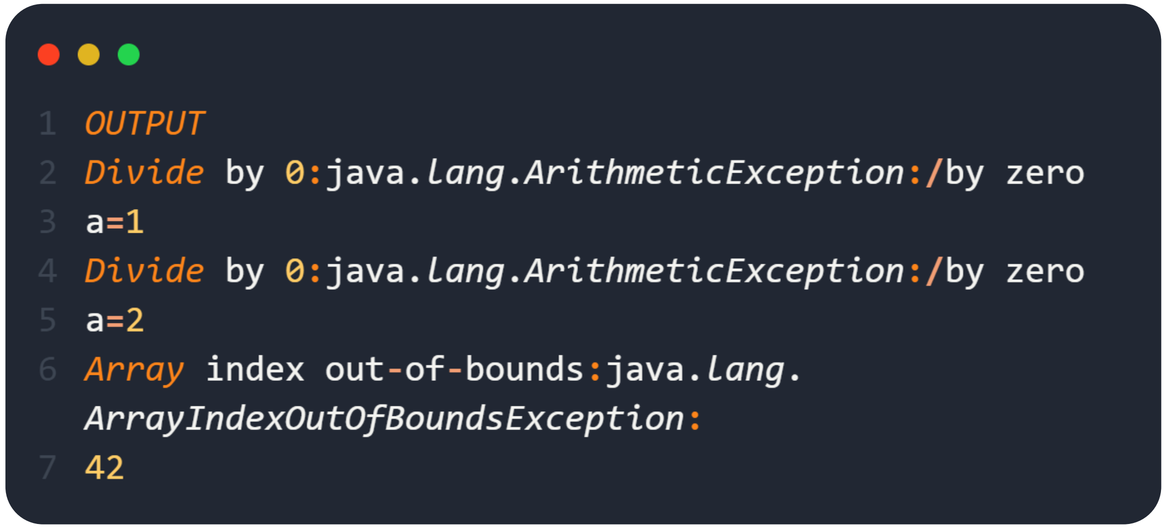


```
OUTPUT

Divide by 0:java.lang.ArithmeticException:/by zero
After try/catch blocks.
a = 1
Array index oob:java.lang.ArrayIndexOutOfBoundsException:42
After try/catch blocks
```

NESTED-CATCHES



```java
class NestTry {
   public static void main(String args[]) {
      try {
         int a = args.length;
         int b = 42 / a;
         System.out.println("a = " + a);
         try { // nested try block
            if (a == 1)
               a = a / (a - a);
// one command-line arg, division by zero if(a==2) {
            int c[] = { 1 };
            c[42] = 99;
// two command-line args, out-of-bounds exception
      } catch (ArrayIndexOutOfBoundsException e) {
       System.out.println("Array index out-of-bounds: " + e);
         }
      } catch (

      ArithmeticException e) {
         System.out.println("Divide by 0: " + e);
      }
   }
}
```



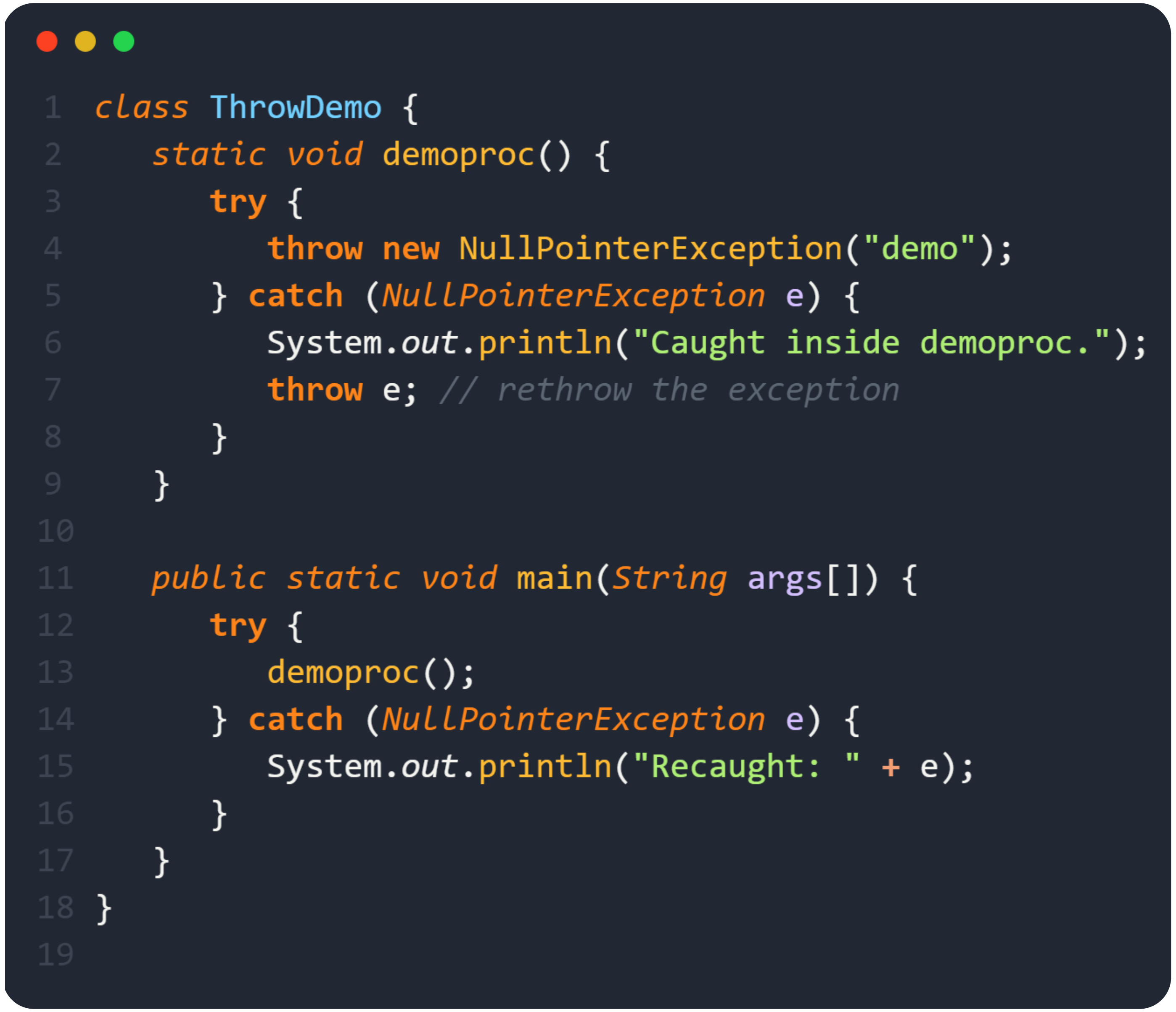
```
OUTPUT
Divide by 0:java.lang.ArithmeticException:/by zero
a=1
Divide by 0:java.lang.ArithmeticException:/by zero
a=2
Array index out-of-bounds:java.lang.
ArrayIndexOutOfBoundsException:
42
```

Throw Exceptions

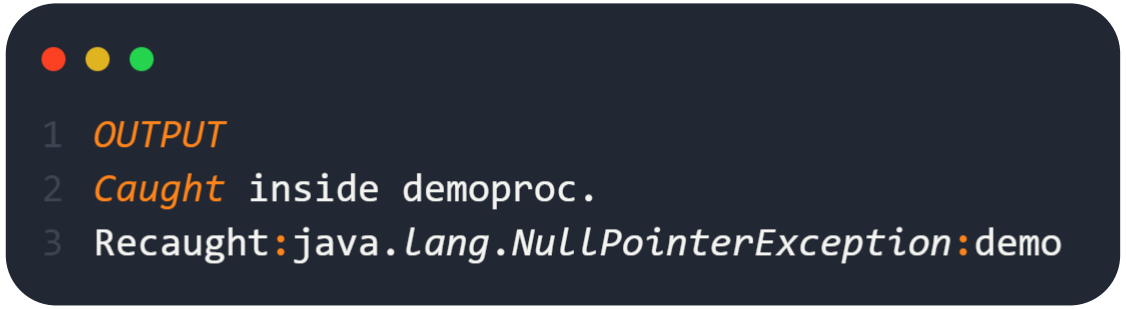
To throw an **exception** explicitly, use the throw statement

- ✓ flow of execution stops immediately after the throw statement
- ✓ nearest enclosing try block is inspected to see if it has a catch statement that matches the type of exception and control is transferred to that statement
- ✓ No match: next enclosing try statement is inspected, and so on
- ✓ no matching catch found in any block: default exception handler halts the program and prints the stack trace

Throw-Exceptions

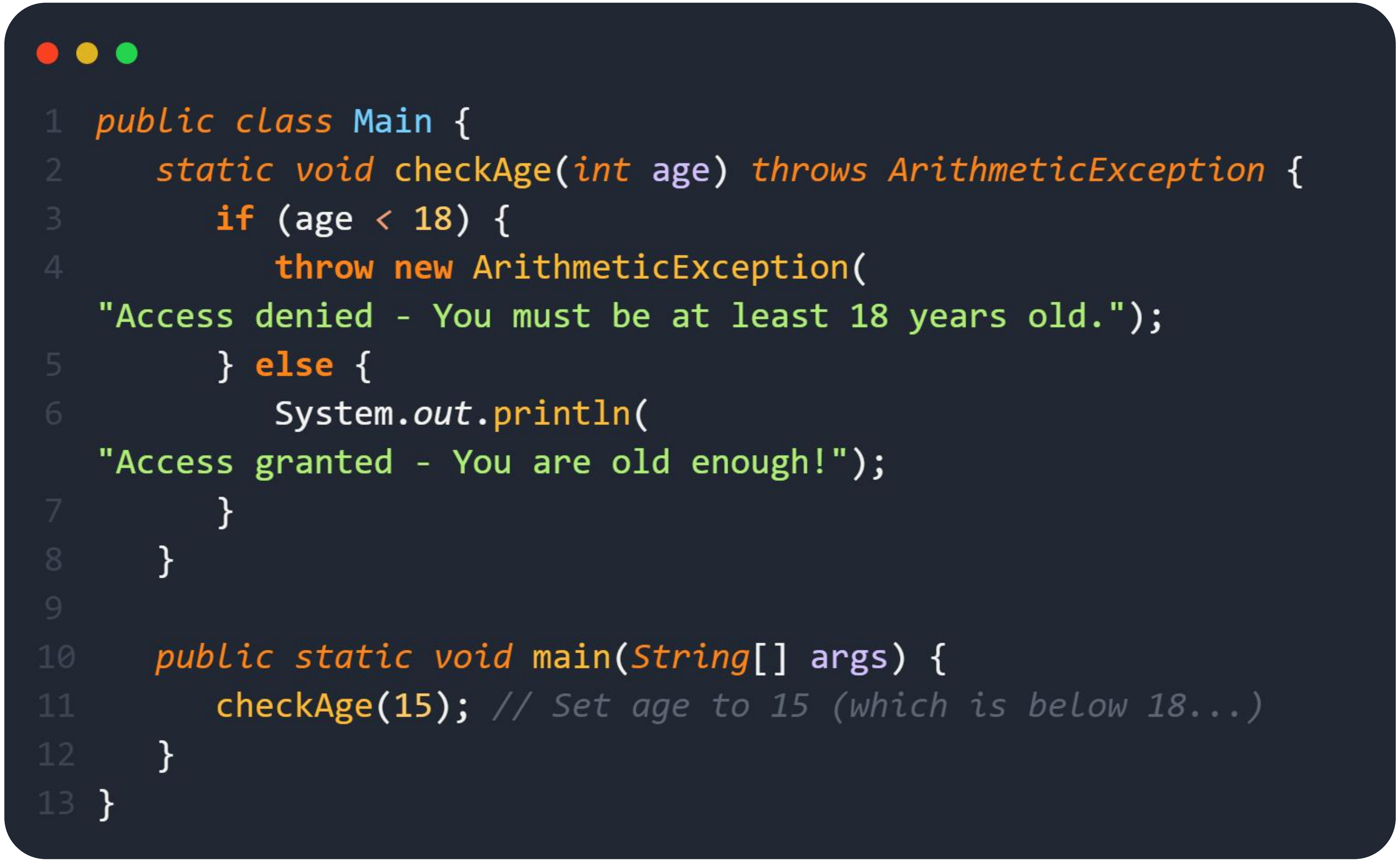


```java
class ThrowDemo {
    static void demoproc() {
        try {
            throw new NullPointerException("demo");
        } catch (NullPointerException e) {
            System.out.println("Caught inside demoproc.");
            throw e; // rethrow the exception
        }
    }

    public static void main(String args[]) {
        try {
            demoproc();
        } catch (NullPointerException e) {
            System.out.println("Recaught: " + e);
        }
    }
}

```



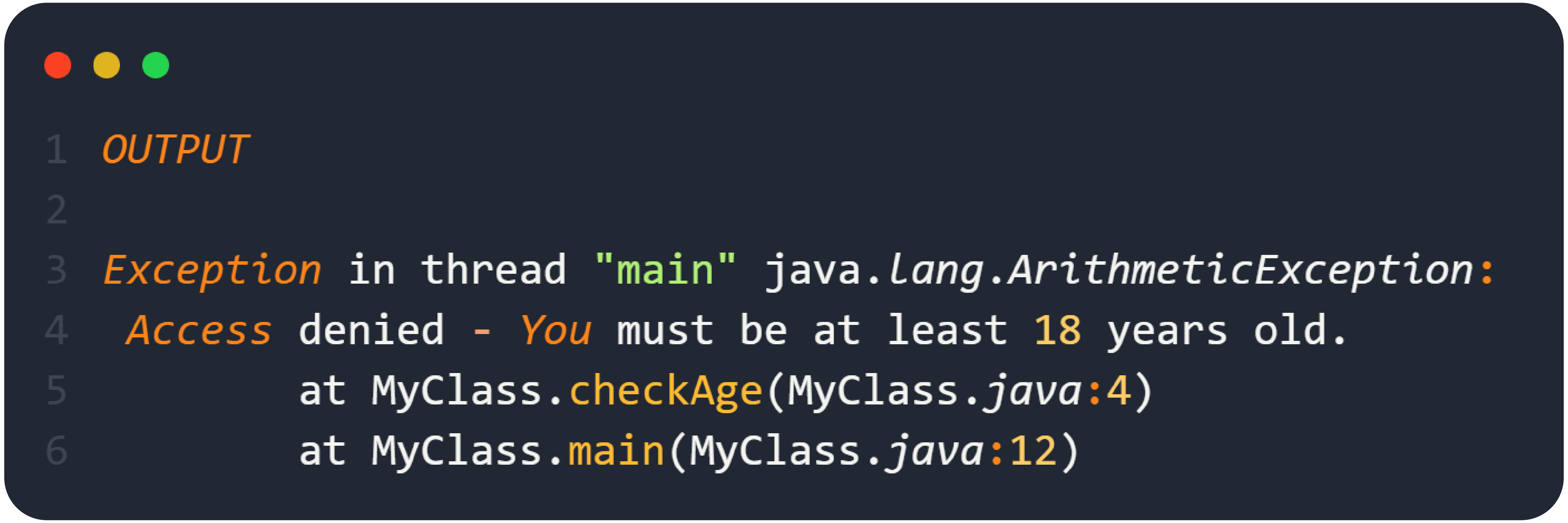
```
OUTPUT
Caught inside demoproc.
Recaught:java.lang.NullPointerException:demo
```

Throws-Exception



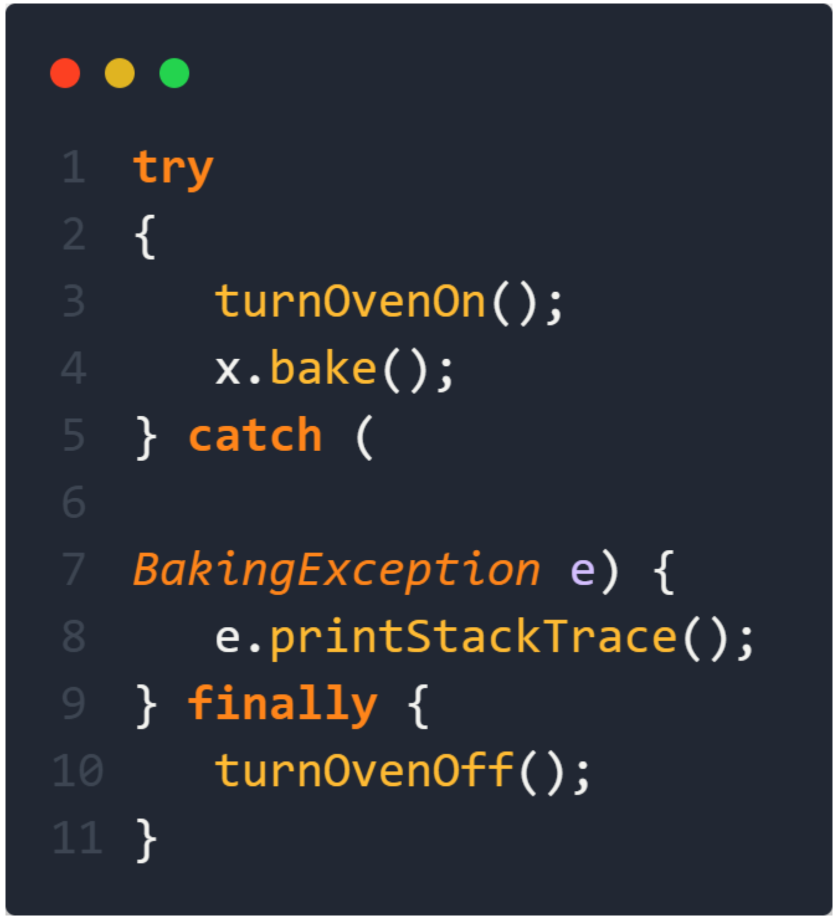
```java
public class Main {
    static void checkAge(int age) throws ArithmeticException {
        if (age < 18) {
            throw new ArithmeticException(
"Access denied - You must be at least 18 years old.");
        } else {
            System.out.println(
"Access granted - You are old enough!");
        }
    }

    public static void main(String[] args) {
        checkAge(15); // Set age to 15 (which is below 18...)
    }
}
```

Throws-Exception



```
OUTPUT

Exception in thread "main" java.lang.ArithmeticException:
 Access denied - You must be at least 18 years old.
        at MyClass.checkAge(MyClass.java:4)
        at MyClass.main(MyClass.java:12)
```

Final-Exception



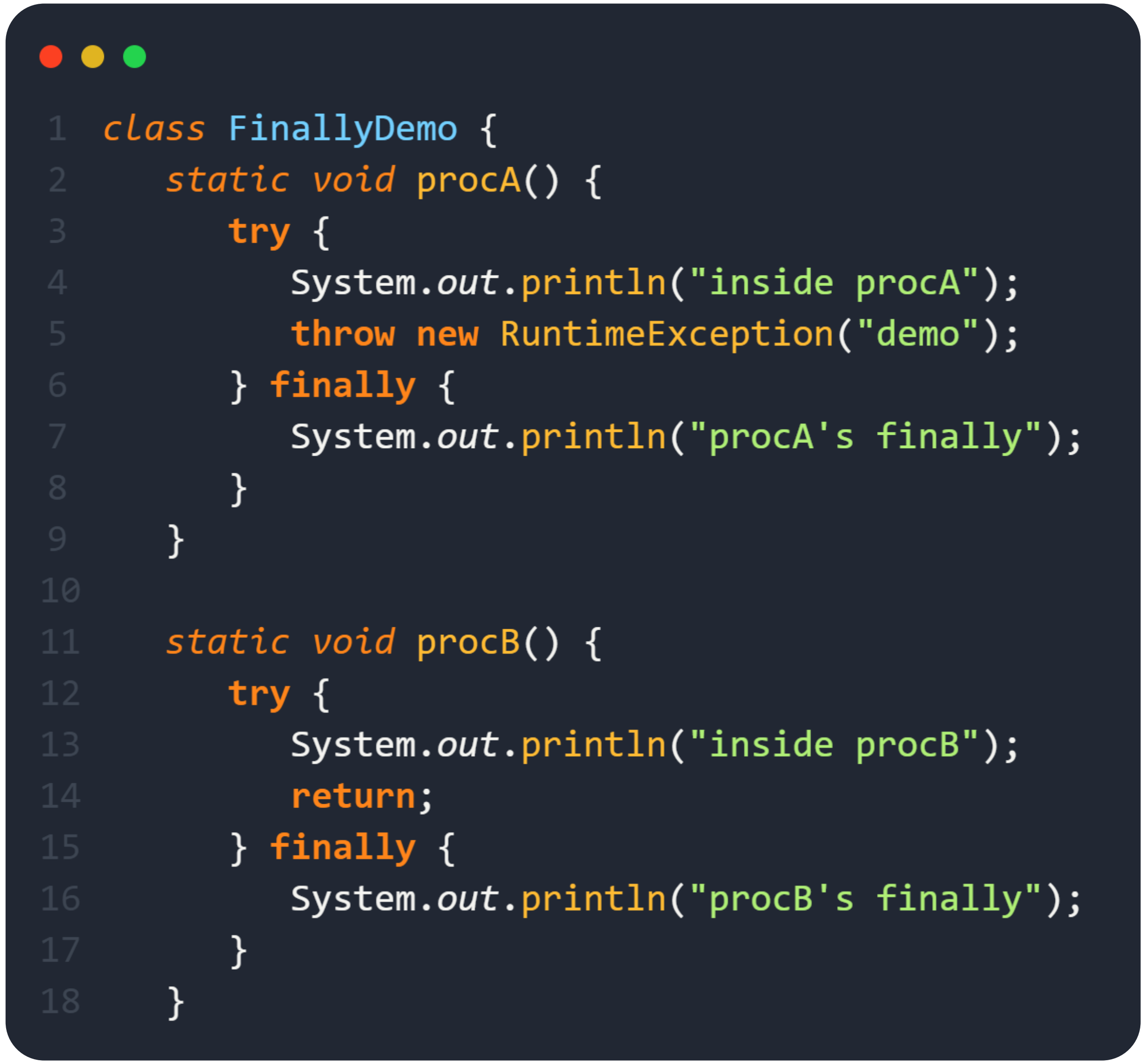
```
try
{
    turnOvenOn();
    x.bake();
} catch (

BakingException e) {
    e.printStackTrace();
} finally {
    turnOvenOff();
}
```

If you try to cook something,
you start by turning on the oven.

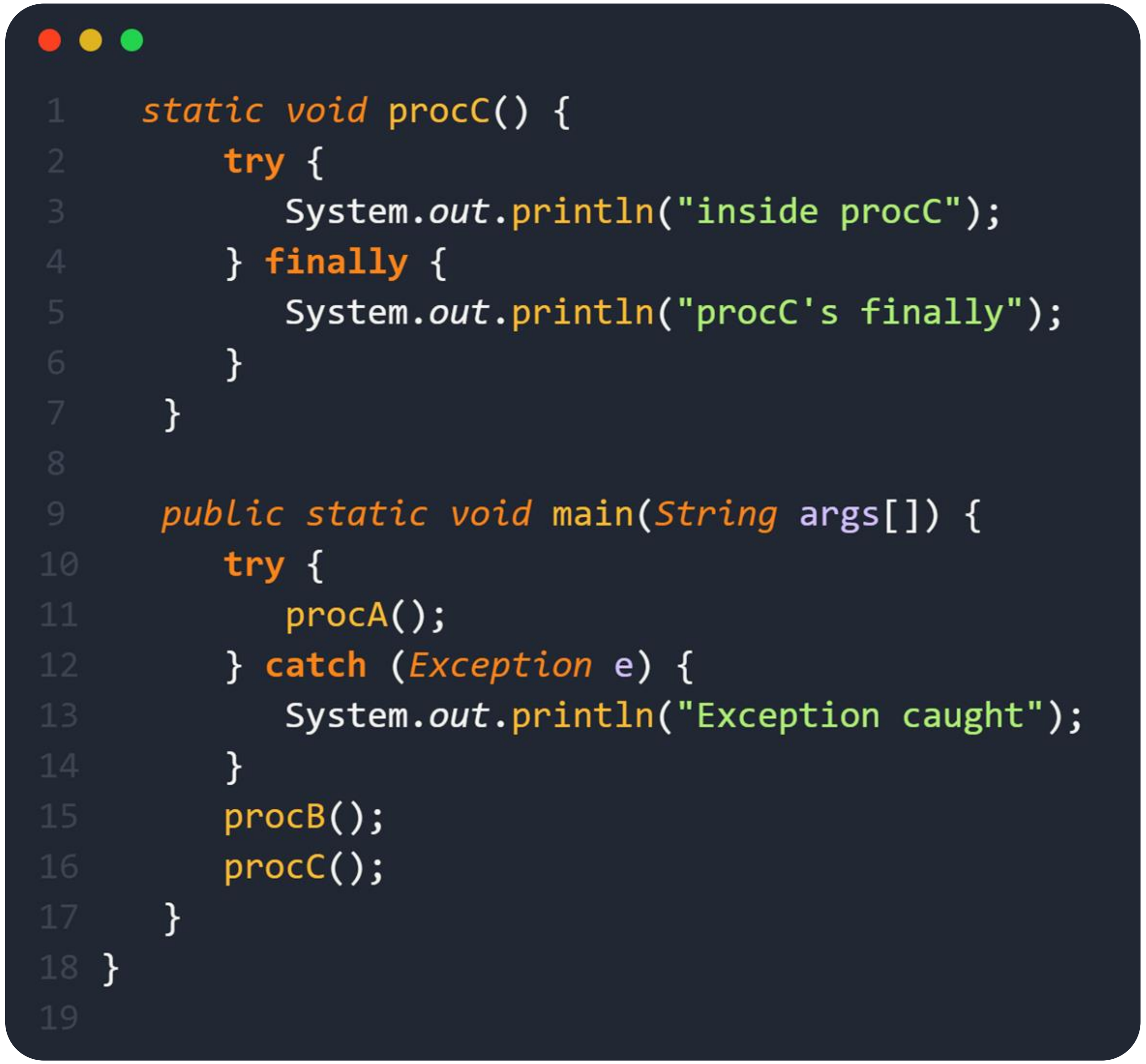
1. If the thing you try is a complete failure,
   a. you must turn off the oven.
2. If the thing you try succeeds,
   b. you must turn off the oven.

You must turn off the oven no matter what! A finally block is where you put code that must run regardless of an exception.

Final-Exception

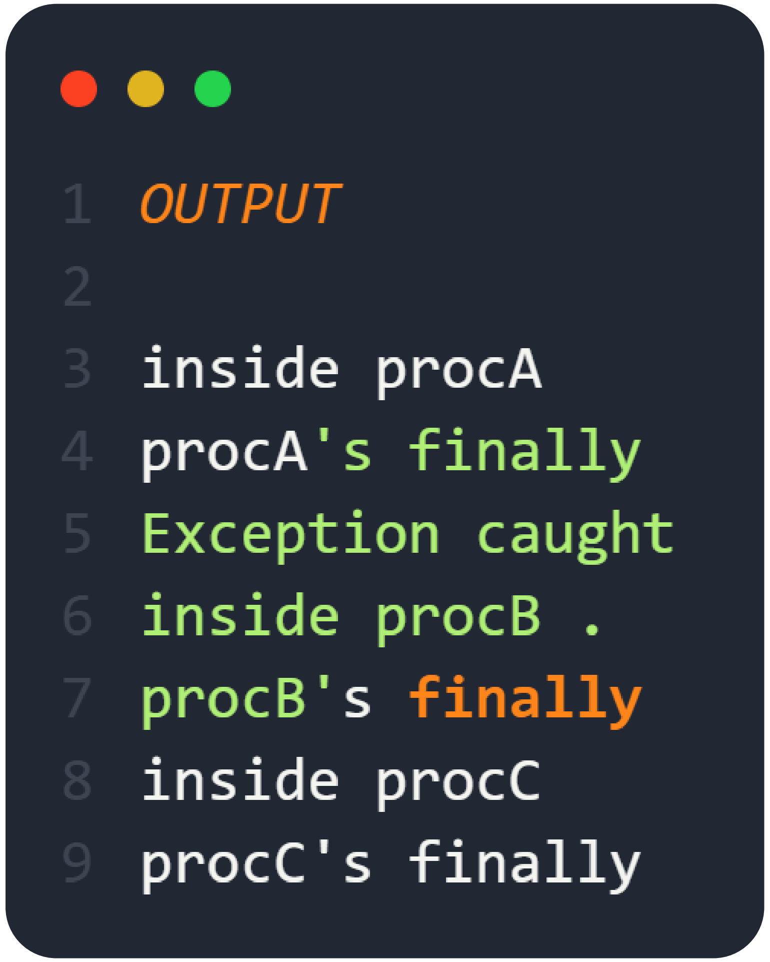


```java
class FinallyDemo {
    static void procA() {
        try {
            System.out.println("inside procA");
            throw new RuntimeException("demo");
        } finally {
            System.out.println("procA's finally");
        }
    }

    static void procB() {
        try {
            System.out.println("inside procB");
            return;
        } finally {
            System.out.println("procB's finally");
        }
    }
```



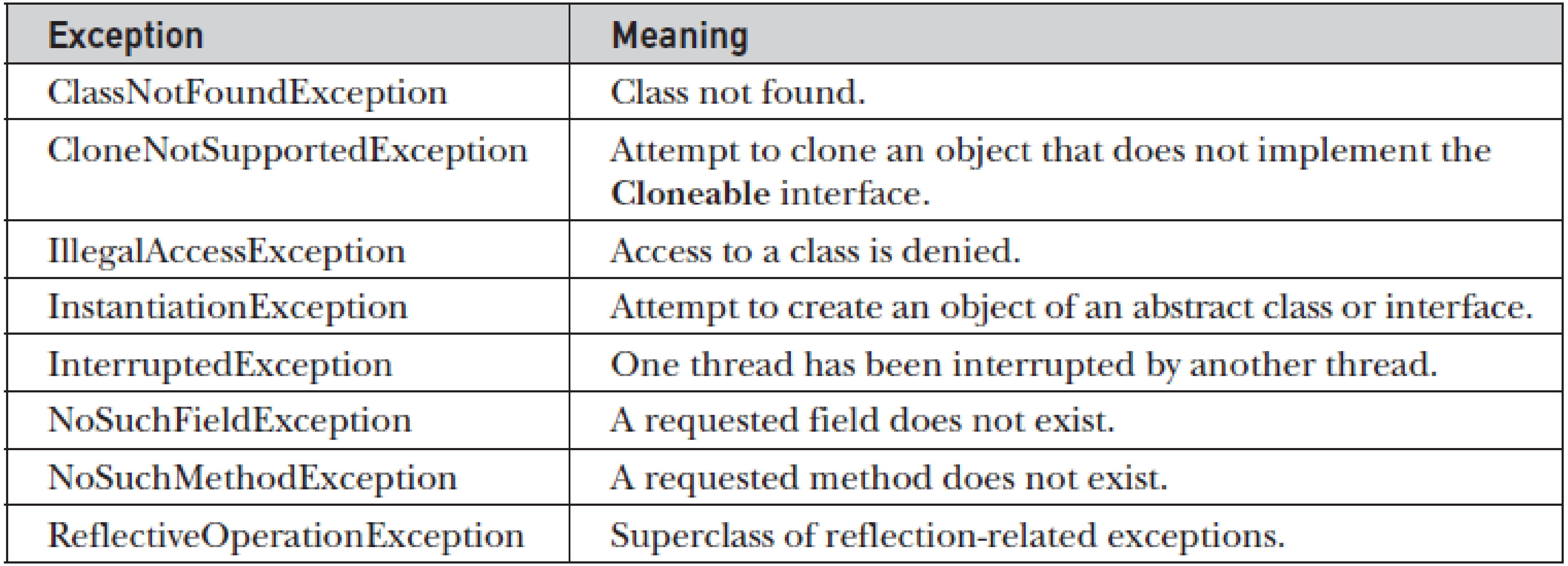
```java
    static void procC() {
        try {
            System.out.println("inside procC");
        } finally {
            System.out.println("procC's finally");
        }
    }

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

Final-Exception



```
OUTPUT

inside procA
procA's finally
Exception caught
inside procB .
procB's finally
inside procC
procC's finally
```

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.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.



Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.
ReflectiveOperationException	Superclass of reflection-related exceptions.

THANK YOU !

THE END