

Exceptions

Syntax Errors, Runtime Errors, and Logic Errors

- *Syntax errors* arise because the rules of the language have not been followed. They are detected by the compiler.
- *Runtime errors* occur while the program is running if the environment detects an operation that is impossible to carry out.
- *Logic errors* occur when a program doesn't perform the way it was intended to.

Runtime Errors

```
1      import java.util.Scanner;
2
3      public class ExceptionDemo {
4          public static void main(String[] args) {
5              Scanner scanner = new Scanner(System.in);
6              System.out.print("Enter an integer: ");
7              int number = scanner.nextInt();
8
9              // Display the result
10             System.out.println(
11                 "The number entered is " + number);
12         }
13     }
```

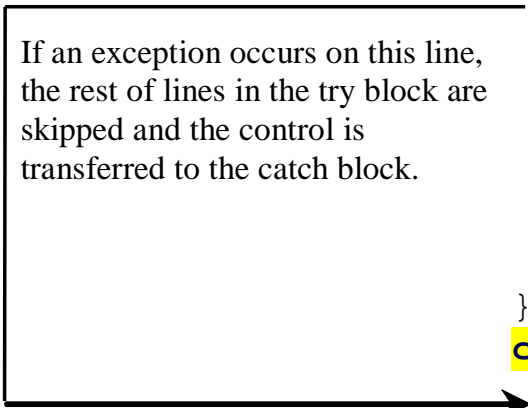
8 If an exception occurs on this line, the rest of the lines in the method are skipped and the program is terminated.

13 ↓ Terminated.

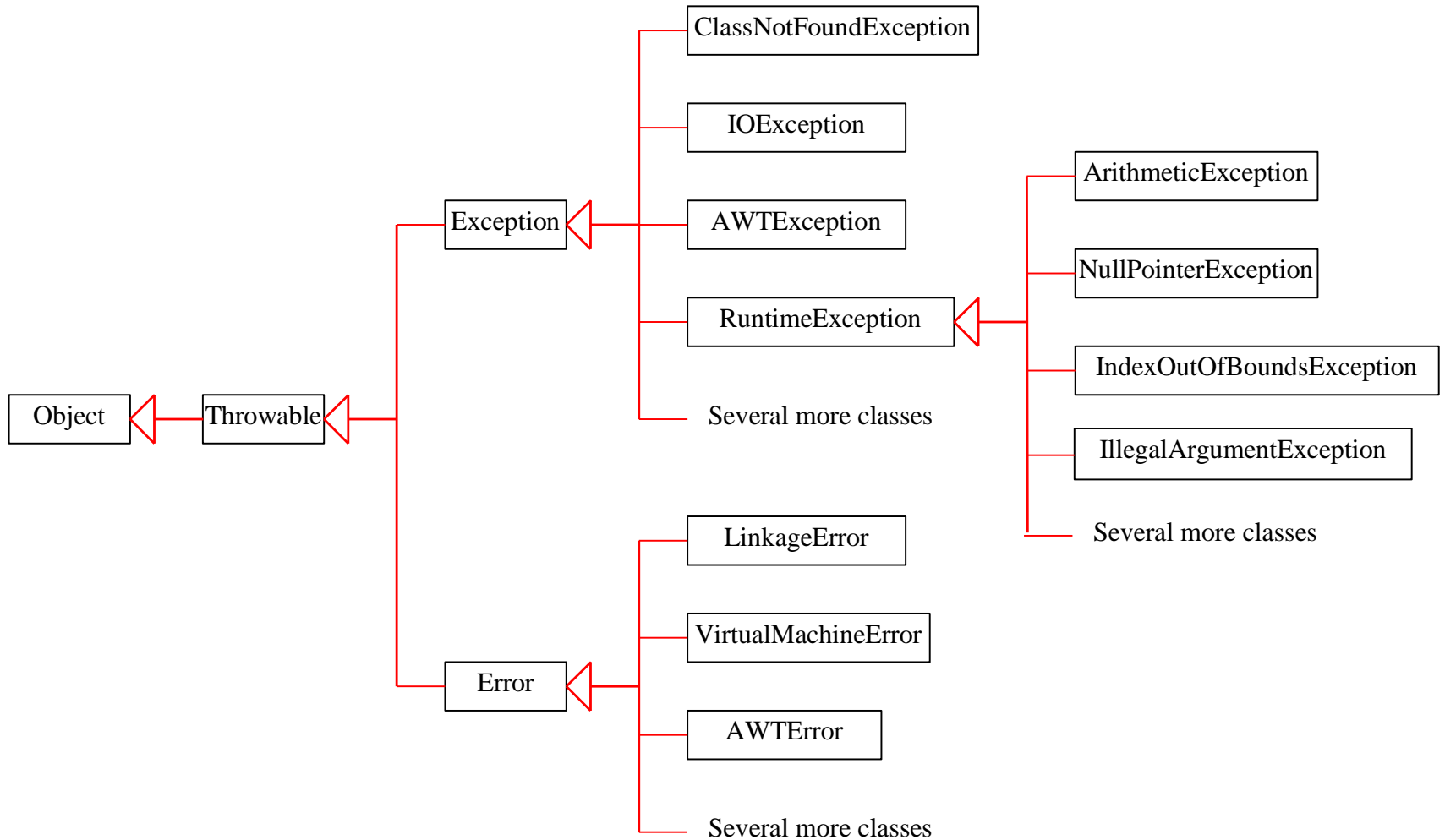
Catch Runtime Errors

```
1      import java.util.*;
2
3      public class HandleExceptionDemo {
4          public static void main(String[] args) {
5              Scanner scanner = new Scanner(System.in);
6              boolean continueInput = true;
7
8              do {
9                  try {
10                     System.out.print("Enter an integer: ");
11                     int number = scanner.nextInt();
12
13                     // Display the result
14                     System.out.println(
15                         "The number entered is " + number);
16
17                     continueInput = false;
18                 }
19                 catch (InputMismatchException ex) {
20                     System.out.println("Try again. (" +
21                         "Incorrect input: an integer is required)");
22                     scanner.nextLine(); // discard input
23                 }
24             } while (continueInput);
25     }
```

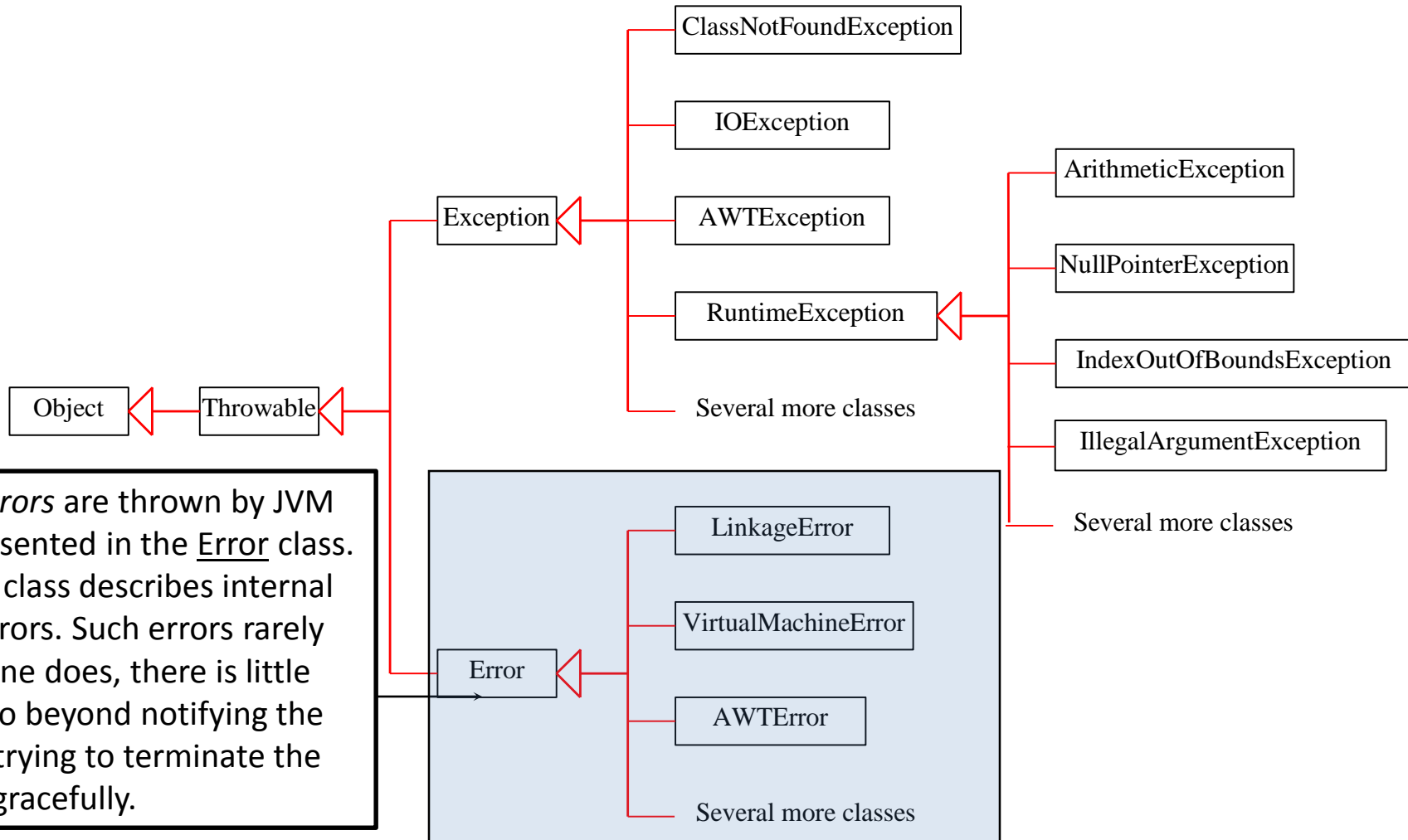
If an exception occurs on this line, the rest of lines in the try block are skipped and the control is transferred to the catch block.



Exception Classes

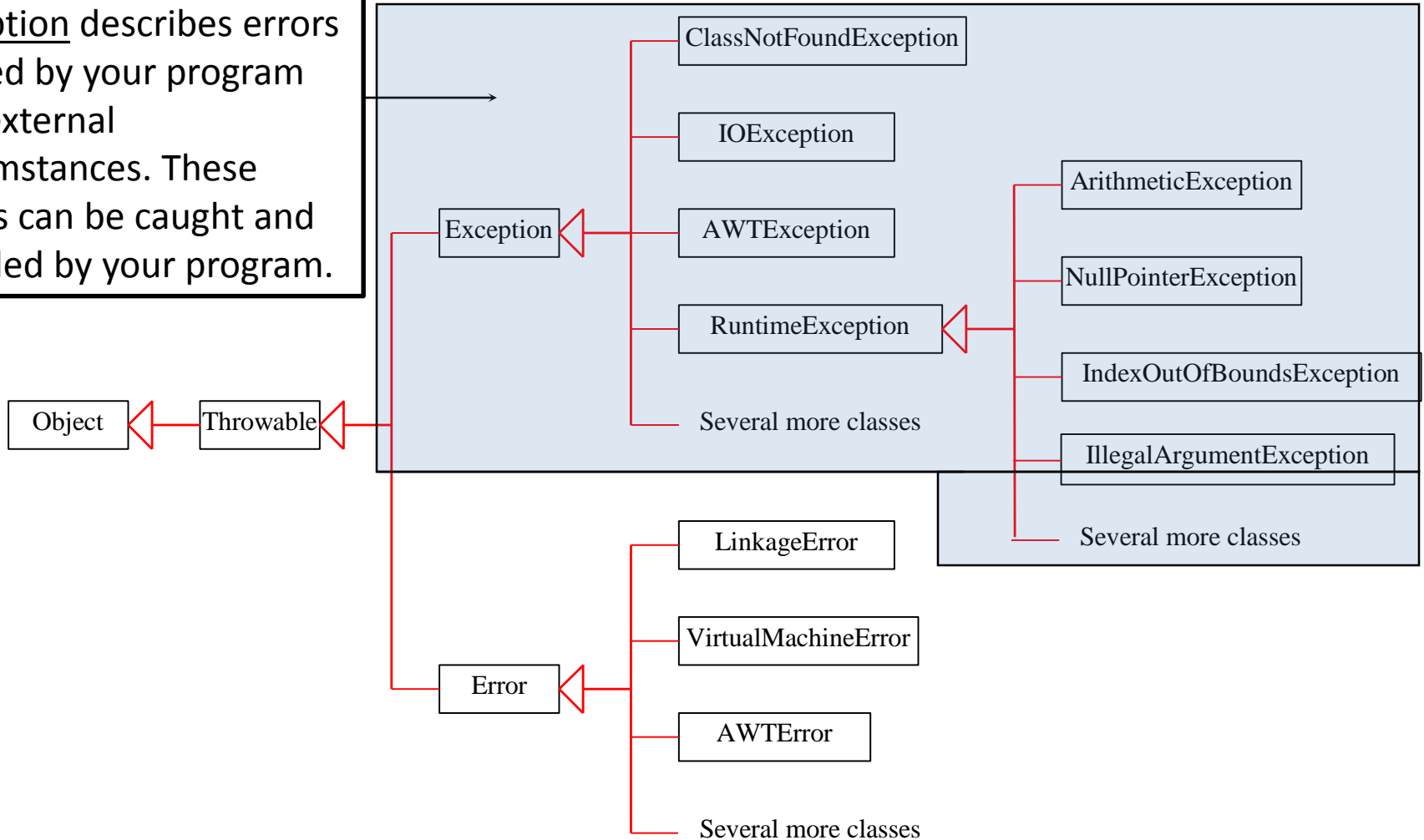


System Errors

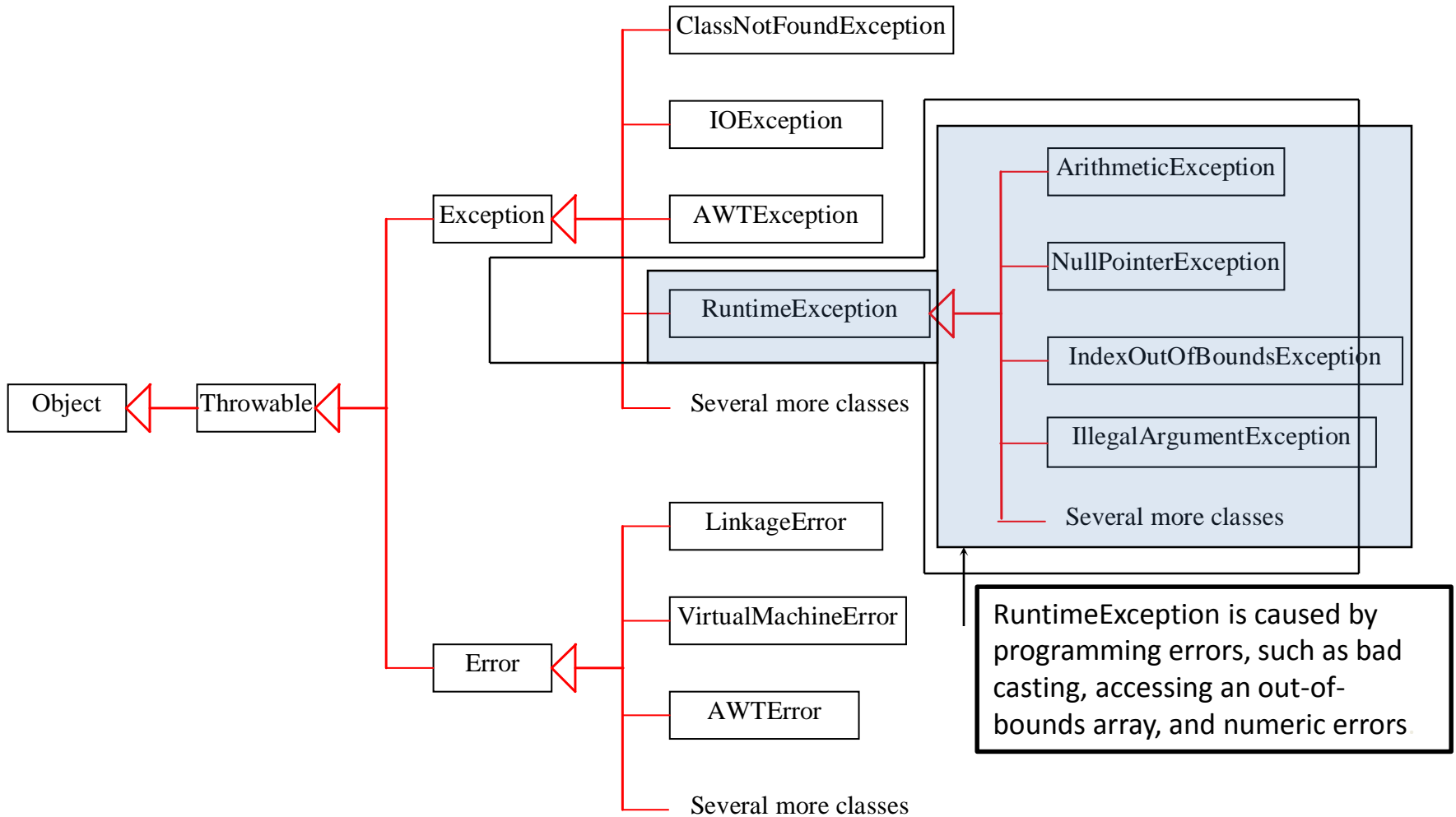


Exceptions

Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.



Runtime Exceptions



Checked Exceptions vs. Unchecked Exceptions

RuntimeException, Error and their subclasses are known as *unchecked exceptions*. All other exceptions are known as *checked exceptions*, meaning that the compiler forces the programmer to check and deal with the exceptions.

Categories Of Exceptions

- Unchecked exceptions
- Checked exception

Characteristics Of Unchecked Exceptions

- The compiler doesn't require you to catch them if they are thrown.
 - *No try-catch block required by the compiler*
- They can occur at any time in the program (not just for a specific method)
- Typically they are fatal runtime errors that are beyond the programmer's control
 - Use conditional statements rather than the exception handling model.
- Examples:
 - `NullPointerException`, `IndexOutOfBoundsException`, `ArithmeticException`...

Common Unchecked Exceptions: NullPointerException

- `int [] arr = null;`
- `arr[0] = 1;` ← **NullPointerException**
- `arr = new int [4];`
- `int i;`
- `for (i = 0; i <= 4; i++)`
- `arr[i] = i;`
- `arr[i-1] = arr[i-1] / 0;`

Common Unchecked Exceptions: ArrayIndexOutOfBoundsException

- `int [] arr = null;`
- `arr[0] = 1;`

- `arr = new int [4];`
- `int i;`
- `for (i = 0; i <= 4; i++)`
- `arr[i] = i;`

- `arr[i-1] = arr[i-1] / 0;`

ArrayIndexOutOfBoundsException
(when i = 4)



Common Unchecked Exceptions: ArithmeticExceptions

1. `int [] arr = null;`
2. `arr[0] = 1;`
3. `arr = new int [4];`
4. `int i;`
5. `for (i = 0; i <= 4; i++)`
6. `arr[i] = i;`
7. `arr[i-1] = arr[i-1] / 0;`



ArithmeticException
(Division by zero)

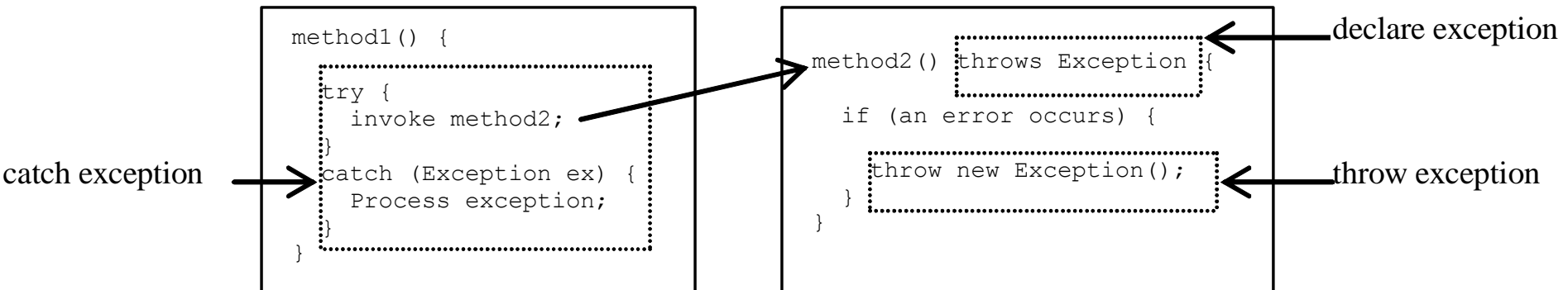
Checked Exceptions

- Must be handled if the potential for an error exists
 - You must use a try-catch block
- Deal with problems that occur in a specific place
 - When a particular method is invoked you must enclose it within a try-catch block
- Example:
 - `InterruptedException` in the case of `join()`

Checked Exceptions

```
try {  
t1.join();  
} catch (InterruptedException e) {  
e.printStackTrace();  
}
```


Declaring, Throwing, and Catching Exceptions



Declaring Exceptions

Every method must state the types of checked exceptions it might throw. This is known as *declaring exceptions*.

```
public void myMethod()  
    throws IOException
```

```
public void myMethod()  
    throws IOException, OtherException
```

Throwing Exceptions

When the program detects an error, the program can create an instance of an appropriate exception type and throw it. This is known as *throwing an exception*. Here is an example,

```
throw new TheException();
```

```
TheException ex = new TheException();  
throw ex;
```

Throwing Exceptions Example

```
/** Set a new radius */  
public void setRadius(double newRadius)  
    throws IllegalArgumentException {  
    if (newRadius >= 0)  
        radius = newRadius;  
    else  
        throw new IllegalArgumentException(  
            "Radius cannot be negative");  
}
```

Catching Exceptions

```
try {  
    statements; // Statements that may throw exceptions  
}  
catch (Exception1 exVar1) {  
    handler for exception1;  
}  
catch (Exception2 exVar2) {  
    handler for exception2;  
}  
...  
catch (ExceptionN exVar3) {  
    handler for exceptionN;  
}
```

The Finally Clause

- An additional part of Java's exception handling model (try-catch-*finally*).
- Used to enclose statements that must always be executed whether or not an exception occurs.

The Finally Clause: Exception Thrown

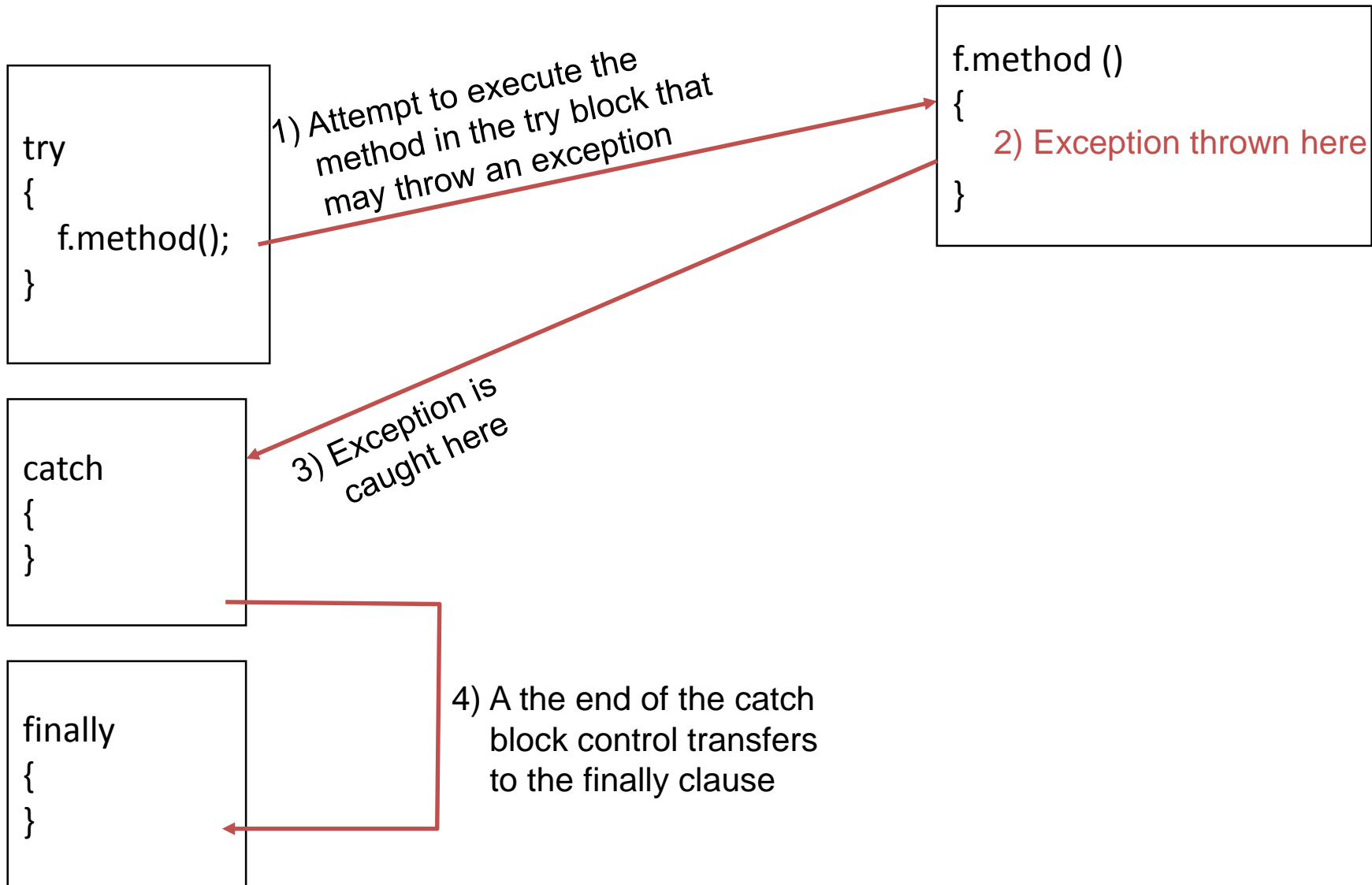
```
try  
{  
    f.method();  
}
```

```
catch  
{  
}
```

```
finally  
{  
}
```

```
f.method ()  
{  
  
}
```

The Finally Clause: Exception Thrown



The Finally Clause: No Exception Thrown

```
try
{
  f.method();
}
```

1) Attempt to execute the method in the try block that may throw an exception

```
f.method ()
{
  2) Code runs okay here
}
```

```
catch
{
}
```

```
finally
{
}
```

3) At the end of f.method () control transfers to the finally clause