Chapter 12
Exception Handling

Motivations

Goal: Robust code.

When a program runs into a runtime error, the program terminates abnormally. How can you handle the runtime error so that the program can continue to run or terminate gracefully?

Answer: Exception Handling.

Exceptions are objects (from built-in classes).

Exception Handling

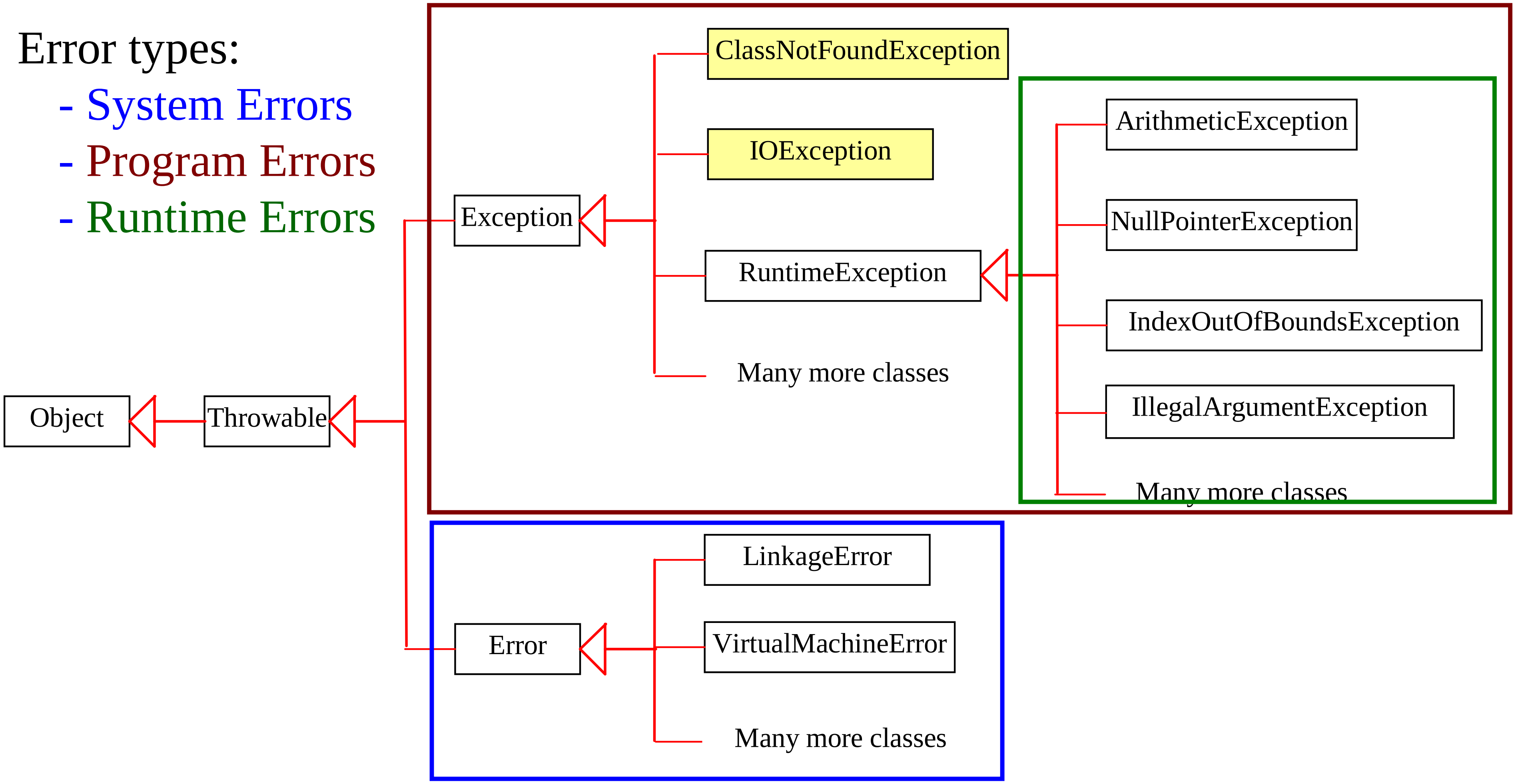
Exception handling enables a method to throw an exception to its caller. Without this capability, a method must handle the exception or terminate the program.

Exceptions are similar to methods, they return values (exceptions).

Exception Types/Classes

Error types:

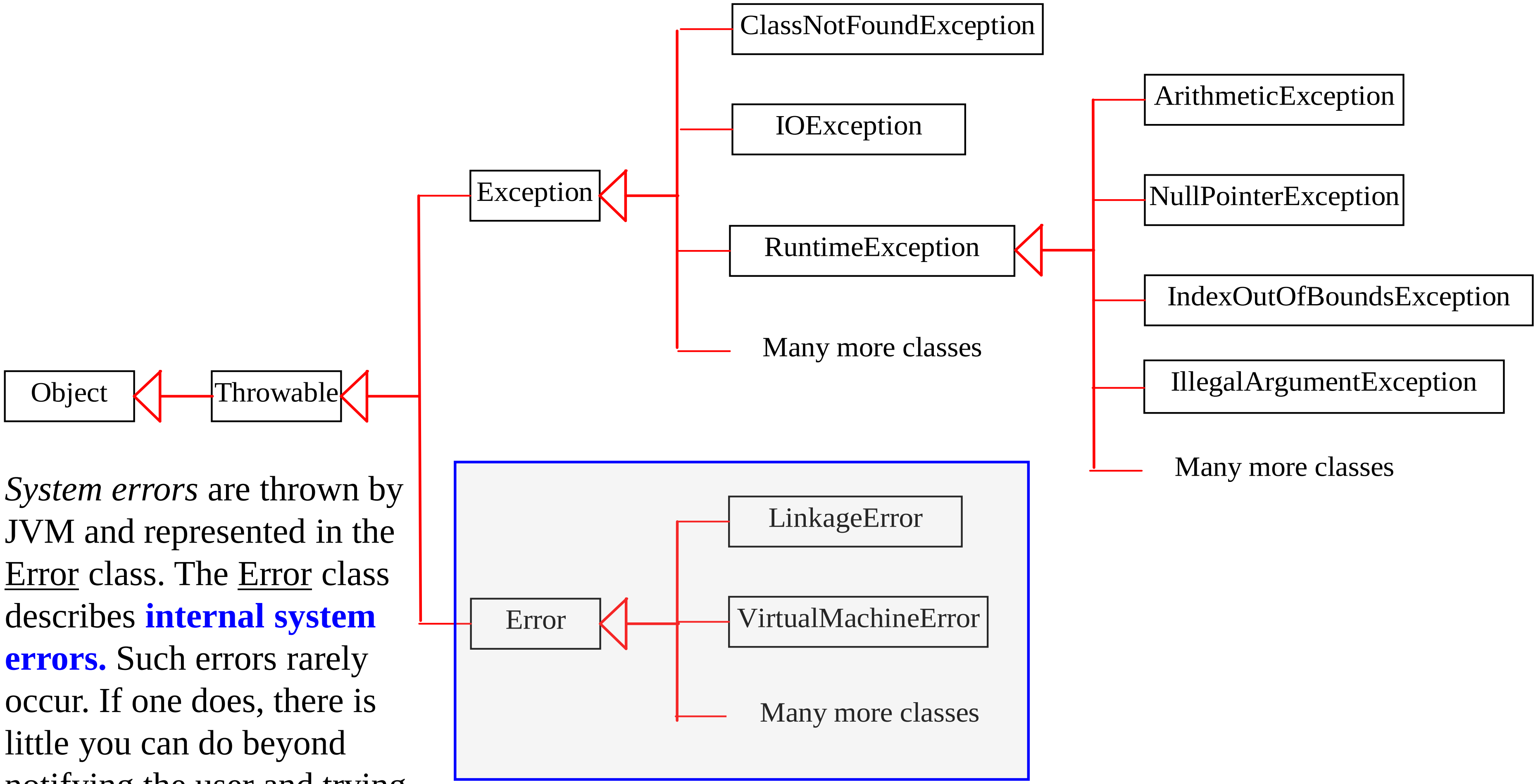
- System Errors
- Program Errors
- Runtime Errors



Unchecked Exceptions: System Errors and Runtime Errors

Checked Exception: IO errors and Class Errors, must check them!

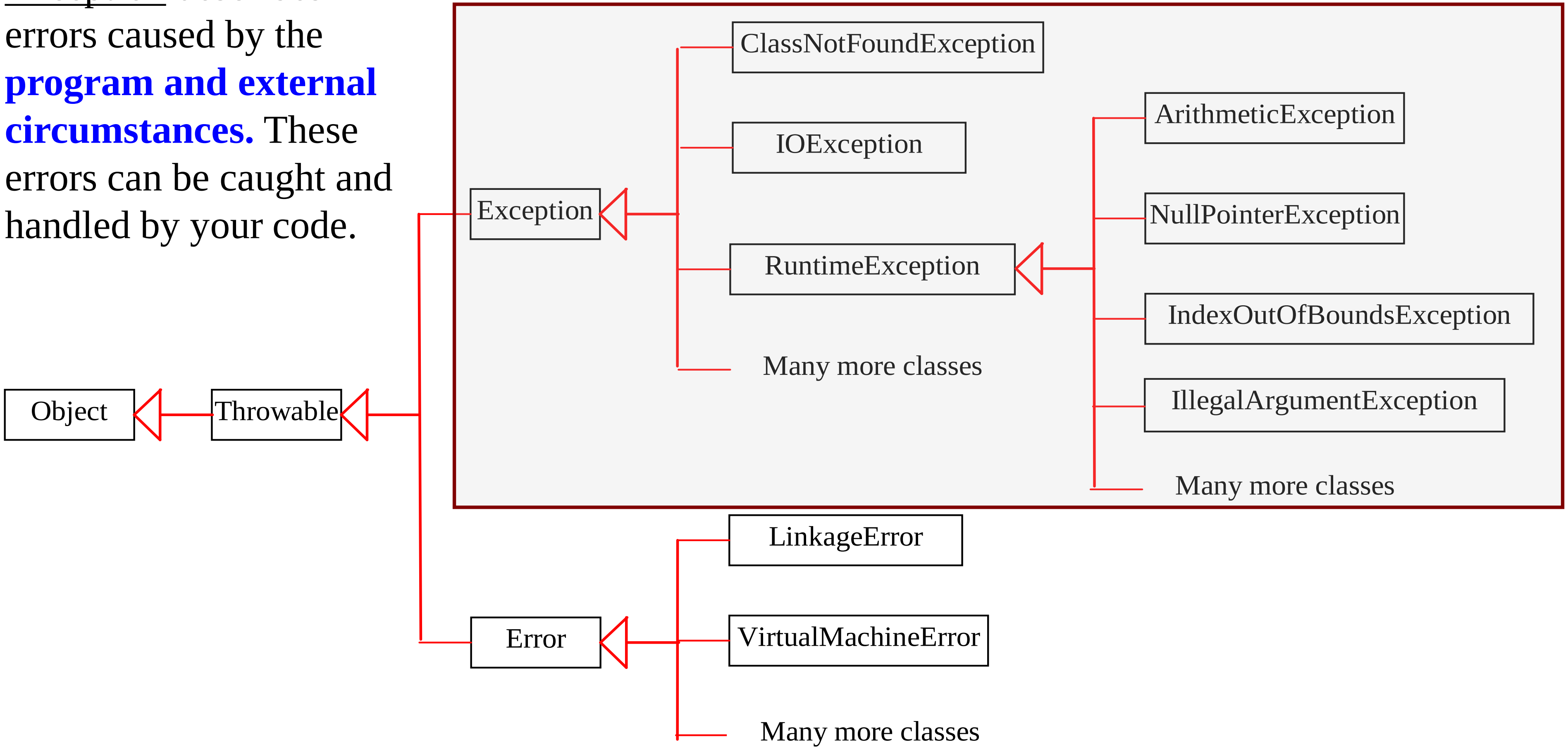
System Errors



System errors are thrown by JVM and represented in the Error class. The Error class describes **internal system errors.** Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

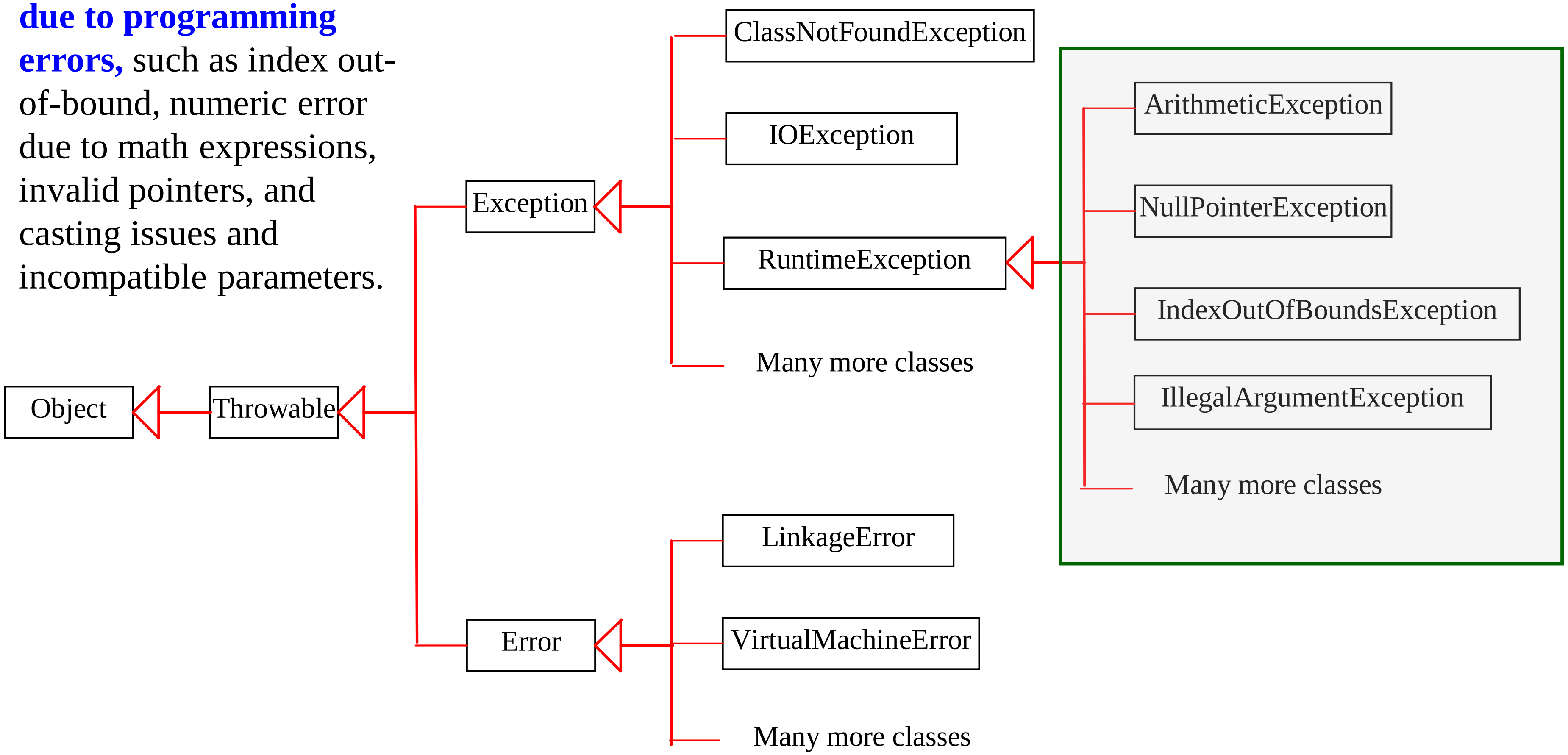
Program Errors

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



Runtime Errors

Runtime errors are thrown by JVM during **runtime due to programming errors,** such as index out-of-bound, numeric error due to math expressions, invalid pointers, and casting issues and incompatible parameters.

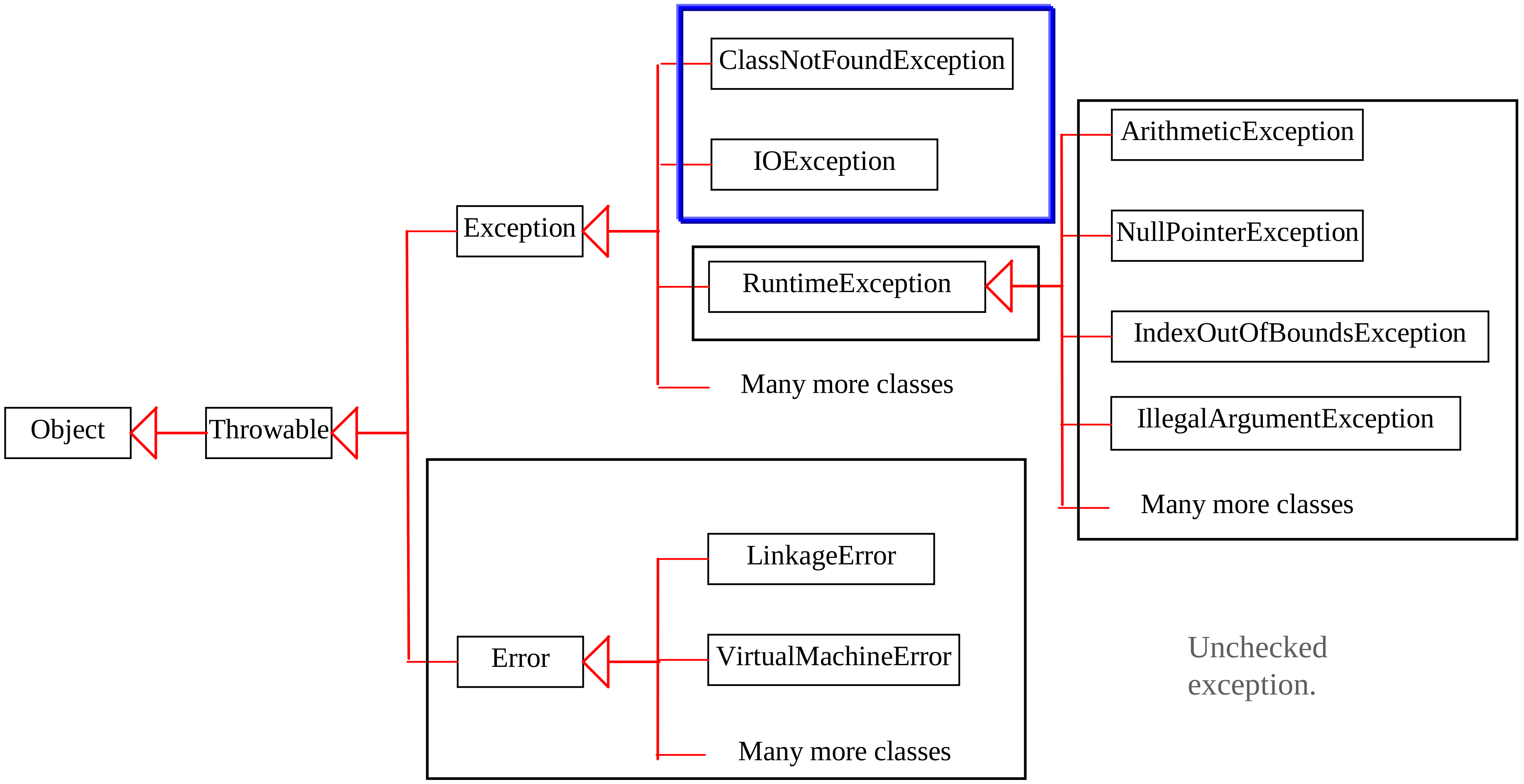


Checked vs. Unchecked Exceptions

Classes *RuntimeException* and *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 (potential errors).

Unchecked Exceptions



Unchecked exception.

Unchecked Exceptions Examples

Unchecked exceptions reflect programming logic errors that are not recoverable. For example,

NullPointerException: Access an object through a reference variable before an object is assigned to it.

IndexOutOfBoundsException: Access an element in an array outside the bounds of the array.

ArithmeticException: Invalid math operation.

IllegalArgumentException: Parameter mismatch.

When to Throw Exceptions

- If you can handle the exception in the method where it occurs, there is no need to throw it.
- If you want the exception to be processed (handled) by the method's caller, you should create an exception object and throw it back to the caller.

Example

```
import java.util.Scanner;

public class Quotient {
  public static void main(String[] args)
 {
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter two integers
    System.out.print("Enter two integers: ");
    int number1 = input.nextInt();
    int number2 = input.nextInt();

    System.out.println(number1 + "/" + number2 + " is " +
       (number1 / number2));
   } // runtime error if number 2 = 0!!
}
```

Example with **try-catch** Block

```
import java.util.Scanner;
public class QuotientWithException1 {

  public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.print("Enter two integers: ");
   int number1 = input.nextInt();
   int number2 = input.nextInt();
   try {
     int result = number1/number2; //may cause runtime error
     System.out.println(number1 + " / " + number2 + " is " +
                        result);
   }
   catch (ArithmeticException ex) {
     System.out.println("Exception: cannot divide an " +
                        "integer by zero.");
   }
   System.out.println("Execution continues ...");
 }
}
```

Example with **`try-catch`** Block

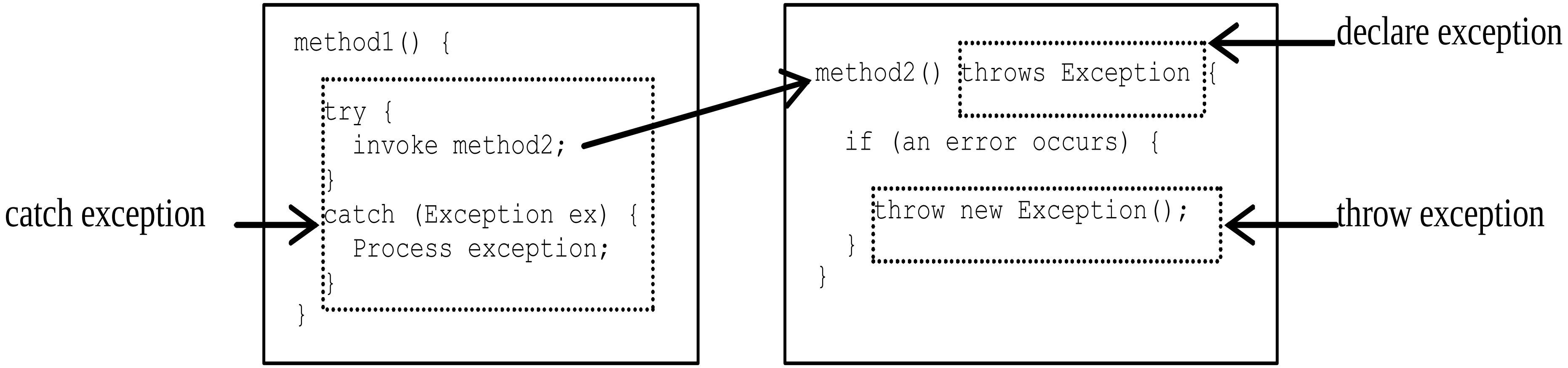
```
import java.util.Scanner;
public class QuotientWithException2 {
  public static int quotient(int num1, int num2) {
   { if (num2 == 0)
         throw new ArithmeticException("Cannot divide by be zero");
     return num1 / num2;
   }
  }

  public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.print("Enter two integers: ");
   int number1 = input.nextInt(); int number2 = input.nextInt();
   try {
     int result = quotient(number1, number2); //method call
     System.out.println(number1 + " / " + number2 + " is " +
                       result);
   }
   catch (ArithmeticException ex) {
     System.out.println("Exception: " + ex.getMessage());
   }
   System.out.println("Execution continues ...");
 }
}
```

Library Methods Throw Exceptions

```
import java.util.*;
public class InputMismatchExceptionDemo {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean continueInput = true;
    do {
       try {
         System.out.print("Enter an integer: ");
         int number = input.nextInt(); // may be invalid input
         // Display the result
         System.out.println("Number entered: " + number);
         continueInput = false;
       }
       catch (InputMismatchException ex) {
         System.out.println("Try again, incorrect input. " +
                            "Integer is required.");
         input.nextLine(); // discard input
       }
    } while (continueInput);
    System.out.println("Execution continues ...");
  }
}
```

Declaring, Throwing, and Catching Exceptions



```
method1() {

  try {
    invoke method2;
  }
  catch (Exception ex) {
    Process exception;
  }
}
```

```
method2() throws Exception {

  if (an error occurs) {

    throw new Exception();
  }
}
```

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 to the caller. This is known as *throwing an exception*. Here is an example,

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

OR

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

Throwing Exceptions Example

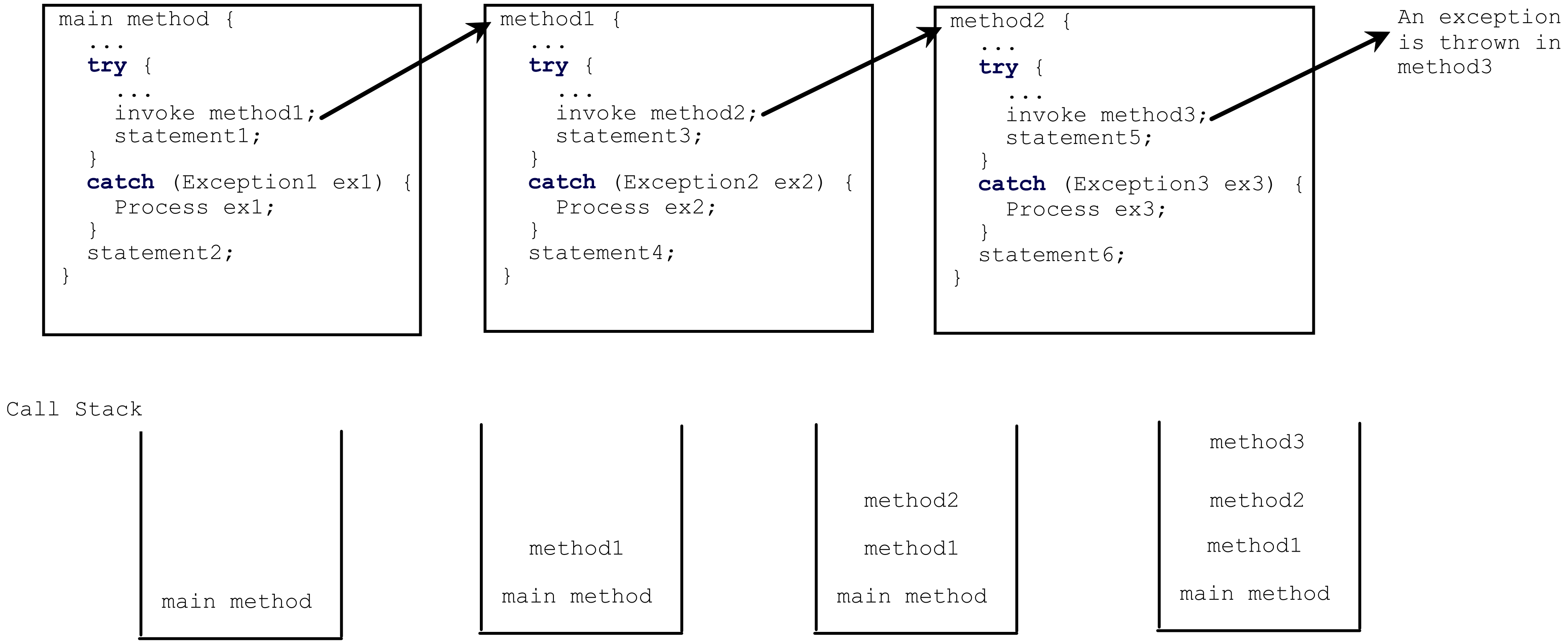
```
// Set a new radius
public void setRadius(double newRadius)
    throws IllegalArgumentException
{
  if (newRadius >= 0)
     radius = newRadius;
  else
    throw new IllegalArgumentException(
                "Radius cannot be negative");
}
```

Catching Exceptions by Caller

```
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;
}
```

Catching Exceptions - Call Stack



```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

An exception is thrown in method3

Catch or Declare Checked Exceptions

Suppose p2 is defined as follows:

```
void p2() throws IOException {
  if (file.exists()){
     throw new IOException("File already exists");
  }

  ...
}
```

HW: Type, compile, and run code in listing 12.12, page 474, in the recommended textbook.

Catch or Declare Checked Exceptions

Java forces you to deal with checked exceptions. If a method declares a checked exception (i.e., an exception other than Error or RuntimeException), you must invoke it in a try-catch block or declare to throw the exception in the calling method. For example, suppose that method p1 invokes method p2 and p2 may throw a checked exception (e.g., IOException), you have to write the code as shown in (a) or (b).

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

(a)

```
void p1() throws IOException {

  p2();

}
```

(b)

The **`finally`** Clause

```
try
{
  statements;
}

catch(TheException ex)
{
  handling ex;
}

finally
{
  finalStatements; //always executes
}
```

Trace a Program Execution - 1

Suppose no exceptions in the statements

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

animation

Trace a Program Execution - 1

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The final block is always executed

animation

Trace a Program Execution - 1

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Next statement in the method is executed

animation

Trace a Program Execution - 2

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Suppose an exception of type Exception1 is thrown in statement2

animation

Trace a Program Execution - 2

```
try {
  statement1;
  statement2;
  statement3; //skipped
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The exception is handled.

animation

Trace a Program Execution - 2

```
try {
   statement1;
   statement2;
   statement3; //skipped
}
catch(Exception1 ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

The final block is always executed.

animation

Trace a Program Execution - 2

```
try {
  statement1;
  statement2;
  statement3; //skipped
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The next statement in the method is now executed.

animation

Trace a Program Execution - 3

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

statement2 throws an exception of type Exception2.

animation

Trace a Program Execution - 3

```
try {
   statement1;
   statement2;
   statement3; //skipped
}
catch(Exception1 ex) {
   handling ex;
}
catch(Exception2 ex) {
   handling ex;
   throw ex;
}
finally {
   finalStatements;
}

Next statement;
```

Handling exception

animation

Trace a Program Execution - 3

```
try {
   statement1;
   statement2;
   statement3; //skipped
}
catch(Exception1 ex) {
   handling ex;
}
catch(Exception2 ex) {
   handling ex;
   throw ex;
}
finally {
   finalStatements;
}

Next statement;
```

Execute the final block

animation

Trace a Program Execution - 3

```
try {
  statement1;
  statement2;
  statement3; //skipped
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;          //control returns to the caller method
}
finally {
  finalStatements;
}

Next statement; //skipped
```

Rethrow the exception and control is transferred to the caller

animation

Trace a Program Execution - 3

```
try {
  statement1;
  statement2;
  statement3; //skipped
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement; //skipped
```

Control is back to the caller method.

When to Use Exceptions

When should you use the try-catch block in the code?

You should use it to deal with unexpected error conditions. **Do not use it to deal with simple, expected situations.** For example, the following code

```
try { //replace with if statement
  System.out.println(refVar.toString());
}
catch (NullPointerException ex) {
  System.out.println("refVar is null");
}
```

When to Use Exceptions

is better to be replaced by

```
if (refVar != null)
  System.out.println(refVar.toString());
else
  System.out.println("refVar is null");
```

Defining Custom Exception Classes

- Use the exception classes in the API whenever possible.
- Define custom exception classes if the predefined classes are not sufficient.
- Define custom exception classes by extending class Exception or a subclass of Exception.

See recommended textbook, listing 13.8, page 470. Method setRadius() throws an exception if the radius is negative.

Example Custom Exception Class

```
public class InvalidRadiusException extends Exception
{
  // Construct an exception
  public InvalidRadiusException(double newRadius)
  {
    super("Invalid radius " + newRadius);
  }
}
```

Example Custom Exception Class

```
// Code from class CircleWithCustomException

// Construct a circle with radius 1
public CircleWithCustomException()
       throws InvalidRadiusException { this(1.0); }

// Construct a circle with a specified radius
public CircleWithCustomException(double newRadius)
       throws InvalidRadiusException {
  setRadius(newRadius);
  numberOfObjects++; }

// Set a new radius
public void setRadius(double newRadius)
            throws InvalidRadiusException {
  if (newRadius >= 0)
     radius = newRadius;
  else
     throw new InvalidRadiusException(newRadius);
}
```

End of Chapter 12