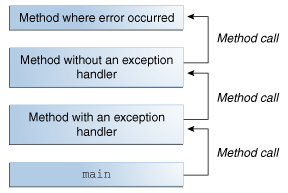
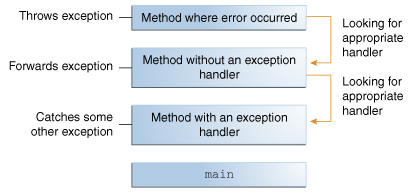
# What Is an Exception?

**Definition:** An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an exception object, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called throwing an exception.

After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred. The list of methods is known as the call stack (see the next figure).





The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code is called an exception handler. The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which the methods were called. When an appropriate handler is found, the runtime system passes the exception to the handler. An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler.

# Catching and Handling Exceptions

This section describes how to use the three exception handler components — the try, catch, and finally blocks — to write an exception handler. Then, the try-with-resources statement, introduced in Java SE 7, is explained. The try-with-resources statement is particularly suited to situations that use Closeable resources, such as streams.

// Note: This class will not compile yet.

import java.io.\*;

import java.util.List;

import java.util.ArrayList;

public class ListOfNumbers {

private List<Integer> list;

private static final int SIZE = 10;

public ListOfNumbers () {

list = new ArrayList<Integer>(SIZE);

for (int i = 0; i < SIZE; i++) {

list.add(new Integer(i));

}

}

public void writeList() {

// The FileWriter constructor throws IOException, which must be caught.

PrintWriter out = new PrintWriter(**new FileWriter("OutFile.txt")**);

for (int i = 0; i < SIZE; i++) {

// The get(int) method throws IndexOutOfBoundsException, which must be caught.

out.println("Value at: " + i + " = " + **list.get(i)**);

}

out.close();

}

}

The first line in boldface is a call to a constructor. The constructor initializes an output stream on a file. If the file cannot be opened, the constructor throws an IOException. The second boldface line is a call to the ArrayList class's get method, which throws an IndexOutOfBoundsException if the value of its argument is too small (less than 0) or too large (more than the number of elements currently contained by the ArrayList).

# The try Block

The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block. In general, a try block looks like the following:

try {

code

}catch and finally blocks . . .

private List<Integer> list;

private static final int SIZE = 10;

public void writeList() {

PrintWriter out = null;

try {

System.out.println("Entered try statement");

out = new PrintWriter(new FileWriter("OutFile.txt"));

for (int i = 0; i < SIZE; i++) {

out.println("Value at: " + i + " = " + list.get(i));

}

}

catch and finally blocks . . .

}

# The catch Blocks

You associate exception handlers with a try block by providing one or more catch blocks directly after the try block. No code can be between the end of the try block and the beginning of the first catch block.

try {

} catch (*ExceptionType name*) {

} catch (*ExceptionType name*) {

}

Each catch block is an exception handler that handles the type of exception indicated by its argument. The argument type, ExceptionType, declares the type of exception that the handler can handle and must be the name of a class that inherits from the Throwable class. The handler can refer to the exception with name.

The catch block contains code that is executed if and when the exception handler is invoked. The runtime system invokes the exception handler when the handler is the first one in the call stack whose ExceptionType matches the type of the exception thrown. The system considers it a match if the thrown object can legally be assigned to the exception handler's argument.

try {

} catch (IndexOutOfBoundsException e) {

System.err.println("IndexOutOfBoundsException: " + e.getMessage());

} catch (IOException e) {

System.err.println("Caught IOException: " + e.getMessage());

}

try {

} catch (IndexOutOfBoundsException e) {

System.err.println("IndexOutOfBoundsException: " + e.getMessage());

} catch (IOException e) {

System.err.println("Caught IOException: " + e.getMessage());

}

# Putting It All Together

public void writeList() {

PrintWriter out = null;

try {

System.out.println("Entering" + " try statement");

out = new PrintWriter(new FileWriter("OutFile.txt"));

for (int i = 0; i < SIZE; i++) {

out.println("Value at: " + i + " = " + list.get(i));

}

} catch (IndexOutOfBoundsException e) {

System.err.println("Caught IndexOutOfBoundsException: "

+ e.getMessage());

} catch (IOException e) {

System.err.println("Caught IOException: " + e.getMessage());

} finally {

if (out != null) {

System.out.println("Closing PrintWriter");

out.close();

}

else {

System.out.println("PrintWriter not open");

}

}

}