**JAC444 - Lecture 4** Segment 1 - Exception **Objectives**

### Upon completion of this lecture, you should be able to:

* Separate Error-Handling Code from Regular Code
* Use Exceptions to Handle Exceptional Events
* Create Your Exceptions

# Exceptions

### In this lesson you will be learning about:

* What is and how to treat an exception in Java
* How to separate error handling from regular code
* How to write exception handler
* Exception class hierarchy
* How to create your own exception classes

# What is an exception?

* **Definition**: An exception is an event that occurs during the execution of a program that disrupts the normal flow of instruction.•
* **Examples**: Serious hardware errors, such as a hard disk crash, to simple programming errors, such as trying to access an out-of-bounds array element.
* **Java solution**: The Java method creates an exception object and hands it off to the runtime system.

# Definitions

* Throwing an exception

It happens when an error occurs the method creates an exception object and hands it off to the runtime system.

* The exception object

The exception object contains information about the exception, including its type and the state of the program when the error occurred.

* Catching an exception

Searching the the call stack until an appropriate exception handler is found. The handler catches the exception.

# Advantages of Exceptions

* Separating Error Handling Code from “Regular” Code
* Propagating Errors Up the Call Stack
* Grouping Error Types and Error Differentiation

# Error Handling Code

Problem: Read a file and copy its content into memory

**… readFile ( … ) {**

open the file;

determine its size; allocate that much memory; read the file into memory; close the file;

…

**}**

# Potential Errors

* What happens if the file can not be opened?
* What if the length of the file can not be determined?
* What happens if enough memory can not be allocated?
* What happens if the read fails?

* What happens if the file can not be closed?

# Error Detection Code Solution

int readFile ( … ) { initialize errorCode = 0;

//open the file; if (theFileIsOpen) {

//determine the length of the file; if (gotTheFileLength) {

//allocate that much memory; if (gotEnoughMemory) {

//read the file into memory; if (readFailed) { errorCode = -1;

}

} else { errorCode = -2;

}

} else { errorCode = -3;

} …

# Java Solution: Exception Handler

voidreadFile() { *try* { open the file; determine its size; allocate that much memory; read the file into memory; close the file;

} *catch* (fileOpenFailed) { doSomething;

} *catch* (sizeDeterminationFailed) { doSomething;

} *catch* (memoryAllocationFailed) { doSomething;

} *catch* (readFailed) { doSomething;

} *catch* (fileCloseFailed) { doSomething; …

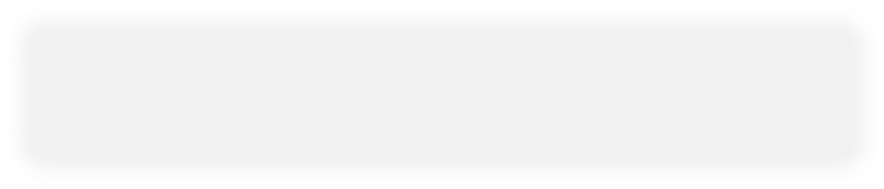
# Exception Hierarchy

* All exceptions that are thrown within a Java program are first-class objects.
* *Leaf* class (a class with no subclasses) represents a specific type of exception.
* *Node* class (a class with one or more subclasses) represents a group of related exceptions**.**

Node Exception

Leaf ExceptionLeaf Exception

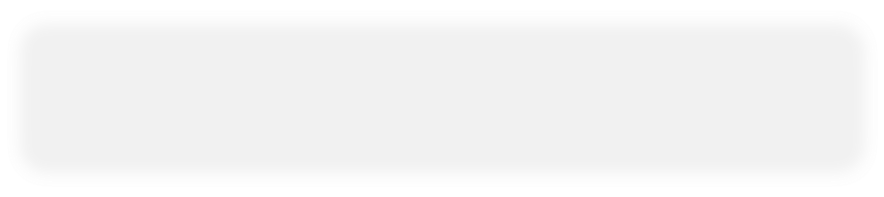
# ArrayException Example



**Exception**



**ArrayException**



**InvalidIndexExceptio**

**n**



**NoSuchElementException**



**E**

**lementTypeException**

# Java Exception: Catch / Specify

Java language requires that methods either:

Catch or

Specify an exception (checked exceptions)

If an exception is not caught or specified by a method the program does NOT compile

# Java Solution: Catch or Specify

**1public printString(...) {try {**

**Catch**

**findString(...);**

**} catch (Exception e) { doErrorProcessing(...);**

**}**

**}**

**2public findString(...) th~~r~~ows Exception { readFile(...);**

**Specify**

**}**

**3public readFile(...) throws Exception { if (...) throw new Exception();**

**}**

# Catch / Specify

* **Catch**

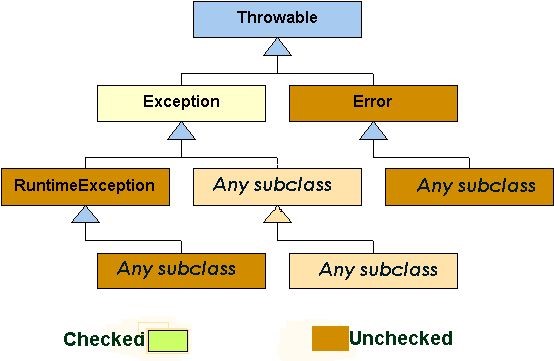
A method can catch an exception by providing an exception handler for that type of exception

* **Specify**

The method could specify that it can throw that exception *What are checked exceptions*?

*Checked exceptions are exceptions that are not runtime exceptions and are checked by the compiler*

# Exception Class Hierarchy



# How to Write an Exception Handler

1. Write the **try** block

It is a block that encloses the statements that might throw an exception

1. Write the **catch** block(s)

It defines the associate with a**try**block by providing one or moreblocks of statements directly after the**try** block.

1. Write the **finally** block **finally** block provides a mechanism that allows your method to clean up after itself

# The try Block

**try {**

Java statements

**}**

Example:

PrintWriter out = null; try {

out = new PrintWriter ( new FileWriter(“X”)); for (int i = 0; i < size; i ++)

System.out.println(vector.elementAt(i)); }

Important note:

A **try** statement must be accompanied by at least one **catch** block or one **finally** block.

# The catch Block(s)

One associates exception handlers with a try statement by providing one or more **catch** blocks directly after the try block: try {

. . .

} catch ( . . . ) { . . .

} catch ( . . . ) {

. . .

} . . .

The general form of catch statement:

catch(ThrowableClass variableName) {

Java statements

}

Catch statement requires **a single formal argument**.

# Catching Exception Types



**Object**



**Throwable**



**Error**



**Exception**



**RuntimeException**



. . .



. . .



. . .

**try {**

**. . .**

**}**

**catch (Exception e)**

**{**

**System.err.println(“Exception caught: “ + e.getMessage( ) );**

**}**

# The finally Block

For cleanup code use a **finally** block.

**try {**

**.** PrintWriter out ...

**} catch (. . .) { . . . } finally {**

if (out != null) {

System.out.println(“Closing PrintWriter”); out.close( );

} else {

System.out.println(“PrintWriter not open”);

}

**}**

**Try, Catch, Finally Blocks**

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 (“At:“ + i + “ = “ + vector.elementAt(i));

} **catch** (ArrayIndexOutOfBoundsException e) {

System.err.println(“Caught Exception: “ 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:);

}

}

}

# Java 8 Try with resources

Old exception handling with try and finally

static String readFirstLineFromFileWithFinallyBlock(String path) throws IOException {

BufferedReader br = new BufferedReader(new FileReader(path));

try {

return br.readLine();

} finally {

if (br != null) br.close();

}

}

New exception handling Java 8: **try with resources**

static String readFirstLineFromFileWithFinallyBlock(String path) throws IOException {

**try (BufferedReader br = new BufferedReader(new FileReader(path))) {**  return br.readLine();

}

}

# Specifying Exceptions

One can specify exceptions in the method definition with the keyword:

## throws

The **throws** clause is composed of the throws keyword followed by a comma-separated list of all the exceptions thrown by method.

Example:

**public void writeList(...)throws IOException, ArrayIndexOutOfBoundsException{ ...**

**}**

The **Throw** Statement

The **throw** statement is used to create an exception object. It requires a single argument as a constructor of an exception object:

**throw new Exception()**

Example: The method is taken from a class that implements common stack object.

public Object pop( ) throws EmptyStackException {

Object obj; if (size == 0) throw new EmptyStackException( ); obj = objectAt(size - 1); setObjectAt(size - 1, null); size -- ; return obj;

}

# The Throwable Class

* **Errors**

Java programs should not catch *Errors.*

* **Exceptions**

Most programs **throw** and **catch** objects that derive from the **Exception** class.

* **Runtime Exceptions**

The compiler allows runtime exceptions to go uncaught and unspecified.

# Conclusion

### After completion of this lesson you should:

1. Write programs using **java.lang.Exception** package and your defined exceptions

1. Apply the principal:

### If anything can go wrong, it will.

