Accelerating Information Technology Innovation
http://aiti.mit.edu

Lecture 12: Exceptions

AITI Nigeria Summer 2012
University of Lagos.

Agenda

- What is an exception
- Some exception terminology
- Why we use exceptions
- How to cause an exception
- How to deal with an exception
- About checked and unchecked exceptions
- Some example Java exceptions
- How to write your own exception

What is an exception?

- An *exception* or *exceptional event* is an event that occurs during the execution of a program that disrupts the normal flow of instructions

- The following will cause exceptions:
  - Accessing an out-of-bounds array element
  - Writing into a read-only file
  - Trying to read beyond the end of a file
  - Sending illegal arguments to a method
  - Performing illegal arithmetic (e.g divide by 0)
  - Hardware failures

Exception Terminology

- When an exception occurs, we say it was *thrown* or *raised*
- When an exception is dealt with, we say it is *handled* or *caught*
- The block of code that deals with exceptions is known as an *exception handler*

Why Use Exceptions?

- Compilation cannot find all errors
- To separate error handling code from regular code
  - Code clarity (debugging, teamwork, etc.)
  - Worry about handling error elsewhere
- To separate error detection, reporting, and handling
- To group and differentiate error types
  - Write error handlers that handle very specific exceptions

Decoding Exception Messages

```
public class ArrayExceptionExample {
    public static void main(String args[]) {
        String[] names = {“Bilha", “Robert"};
        System.out.println(names[2]);
    }
}
```

- The println in the above code causes an exception to be thrown with the following exception message:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 2 at
  ArrayExceptionExample.main(ArrayExceptionExample.j
  ava:4)
```

Exception Message Format

- Exception messages have the following format:

```
[exception class]: [additional description
  of exception] at
  [class].[method]([file]:[line number]
```

Exception Messages Example

- Exception message from array example

```
java.lang.ArrayIndexOutOfBoundsException: 2 at
  ArrayExceptionExample.main(ArrayExceptionExample.j
  ava:4)
```

- What is the exception class?

  `java.lang.ArrayIndexOutOfBoundsException`
- Which array index is out of bounds?

  `2`
- What method throws the exception?

  `ArrayExceptionExample.main`
- What file contains the method?

  `ArrayExceptionExample.java`
- What line of the file throws the exception?

  `4`

Throwing Exceptions

- Use the *throw* statement to throw an exception
  - `if (student == null)`
    ```
    throw new NullPointerException();
    ```

- *throw* statement requires a single argument: a Throwable object
  - *Throwable* objects are instances of any subclass of the Throwable class
    - Include all types of errors and exceptions
    - Check the API for a full listing of Throwable objects

Handling Exceptions

- Can use a *try-catch* block to handle exceptions that are thrown

```
try {
   // code that might throw exception
}
catch ([Type of Exception] e) {
   // what to do if exception is thrown
}
```

Handling Multiple Exceptions

- Can handle multiple possible exceptions by multiple successive catch blocks

```
try {
    // code that might throw multiple
    // exceptions
}
catch (IOException e) {
    // handle IOException
}
catch (ClassNotFoundException e2) {
    // handle ClassNotFoundException
}
```

Finally Block

- Can also use the optional *finally* block at the end of the try-catch block

- *finally* block provides a mechanism to clean up regardless of what happens within the try block
  - Can be used to close files or to release other system resources

Try-Catch-Finally Block

```
try {
    // code that might throw exception
}
catch ([Type of Exception] e) {
    // what to do if exception is thrown
}
finally {
    // statements here always get
    // executed, regardless of what
    // happens in the try block
}
```

Unchecked Exceptions

- *Unchecked exceptions* or *RuntimeException*s occur within the Java runtime system

- Examples of unchecked exceptions
  - arithmetic exceptions (dividing by zero)
  - pointer exceptions (trying to access an object's members through a null reference)
  - indexing exceptions (trying to access an array element with an index that is too large or too small)

- A method does not have to catch or specify that it throws unchecked exceptions, although it may

More on Unchecked Exceptions

- Can occur at many points in the program
- Program handling such exceptions would be cluttered, pointlessly
  - Only handle unchecked exceptions at important program points

Checked Exceptions

- Those other exceptions that the compiler can detect easily
- Usually originate in library code
- For example, exceptions occurring during I/O, SMSLib, Files
- Compiler ensures that:checked exceptions are:
  - caught using try-catch or
  - are specified to be passed up to calling method

Handling Checked Exceptions

- Every method must catch checked exceptions **OR** specify that it passes them to the caller (using the ***throws*** keyword)

```
void readFile(String filename) {
   try {
    FileReader reader = new
        FileReader("myfile.txt");
        // read from file . . .
    } catch (FileNotFoundException e) {
        System.out.println("file was not found");
    }
}
```

OR

```
void readFile(String filename) throws
   FileNotFoundException {
     FileReader reader = new FileReader("myfile.txt");
     // read from file . . .
}
```

Writing Your Own Exceptions

- At least 2 types of exception constructors exist:

1. Default constructor: No arguments

```
NullPointerException e = new
  NullPointerException();
```

2. Constructor that has a detailed message: Has a single `String` argument

```
IllegalArgumentExceptione e =
  new IllegalArgumentException(“Number must
  be positive");
```

Writing Your Own Exceptions

- Your own exceptions must be a subclass of the Exception class and have at least the two standard constructors

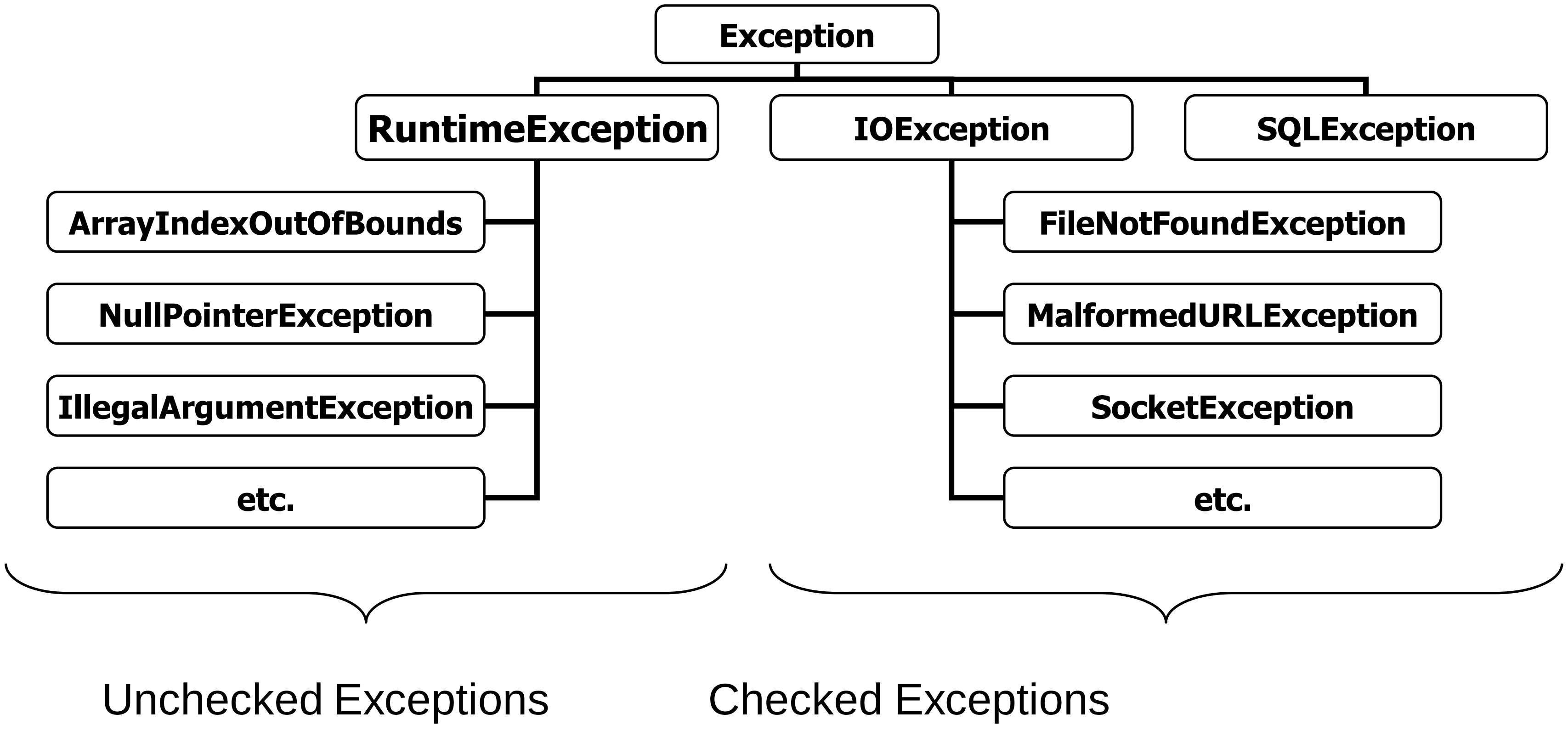
```
public class MyCheckedException extends IOException
  {
    public MyCheckedException() {}
    public MyCheckedException(String m){
   super(m);}
}


public class MyUncheckedException extends
  RuntimeException {
    public MyUncheckedException() {}
    public MyUncheckedException(String m)
   {super(m);}
}
```

Checked or Unchecked?

- If a user can reasonably be expected to recover from an exception, make it a checked exception
- If a user cannot do anything to recover from the exception, make it an unchecked exception
- Judgment call on the part of the designers of the Java programming language
- http://java.sun.com/docs/books/jls/second_edition/html/exceptions.doc.html

Exception Class Hierarchy



- Look in the Java API for a full list of exceptions

Lecture Summary

- Exceptions disrupt the normal flow of the instructions in the program
- Exceptions are handled using a try-catch or a try-catch-finally block
- A method throws an exception using the throw statement
- A method does not have to catch or specify that it throws unchecked exceptions, although it may

Lecture Summary

- Every method must catch possible checked exceptions or specify that it may throw them

- If you write your own exception, it must be a subclass of the Exception class
  - Define the two standard constructors