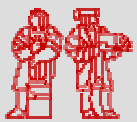
MIT-AITI
Lecture 14: Exceptions
Handling Errors with Exceptions

Kenya 2005

In this lecture, you will learn…

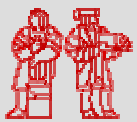
- What an exception is
- Some exception terminology
- Why we use exceptions
- How to cause an exception
- How to deal with an exception
- About checked and unchecked exceptions
- Some example Java exceptions
- How to write your own exception



What is an exception?

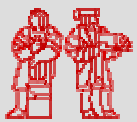
- An *exception* or *exceptional event* is an event that occurs during the execution of a program that disrupts the normal flow of instructions

- 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



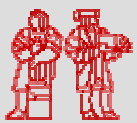
Exception Terminology

- 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

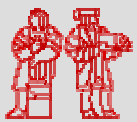


Decoding Exception Messages

```
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:

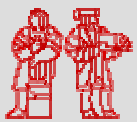
```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 2 at
  ArrayExceptionExample.main(ArrayExceptionExampl
  e.java:4)
```



Exception Message Format

- Exception messages have the following format:

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



Exception Messages Mini Pop-Quiz

- Exception message from array example

```
java.lang.ArrayIndexOutOfBoundsException: 2 at
   ArrayExceptionExample.main(ArrayExceptionExampl
   e.java:4)
```

- What is the exception class?

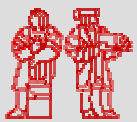
  `java.lang.ArrayIndexOutOfBoundsException`
- Which array index is out of bounds?

  `2`
- What method throws the exception?

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

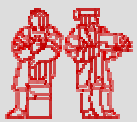
  `ArrayExceptionExample.java`
- What line of the file throws the exception?

  `4`



Throwing Exceptions

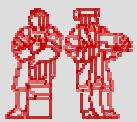
- All methods use the *throw* statement to throw an exception
  - `if (student.equals(null))`
    `throw new NullPointerException();`
- The throw statement requires a single argument: a throwable object
- *Throwable* objects are instances of any subclass of the Throwable class
  - Include all types of errors and exceptions
  - Check the API for a full listing of throwable objects



Handling Exceptions

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

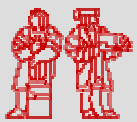
```
try {
   // code that might throw exception
}
catch ([Type of Exception] e) {
   // what to do if exception is thrown
}
```



Handling Multiple Exceptions

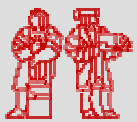
- You can handle multiple possible exceptions by multiple successive catch blocks

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



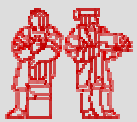
Finally Block

- You can also use the optional *finally* block at the end of the try-catch block
- The 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



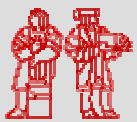
Try-Catch-Finally Block

```
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
}
```



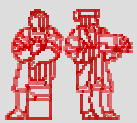
Unchecked Exceptions

- *Unchecked exceptions* or *runtime exceptions* 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



Checked Exceptions

- *Checked exceptions* or *nonruntime exceptions* are exceptions that occur in code outside of the Java runtime system
- For example, exceptions that occur during I/O (covered next lecture) are nonruntime exceptions
- The compiler ensures that nonruntime exceptions are caught or are specified to be thrown (using the *throws* keyword)



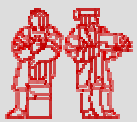
Handling Checked Exceptions

- Every method must catch checked exceptions **OR** specify that it may throw them (using the ***throws*** keyword)

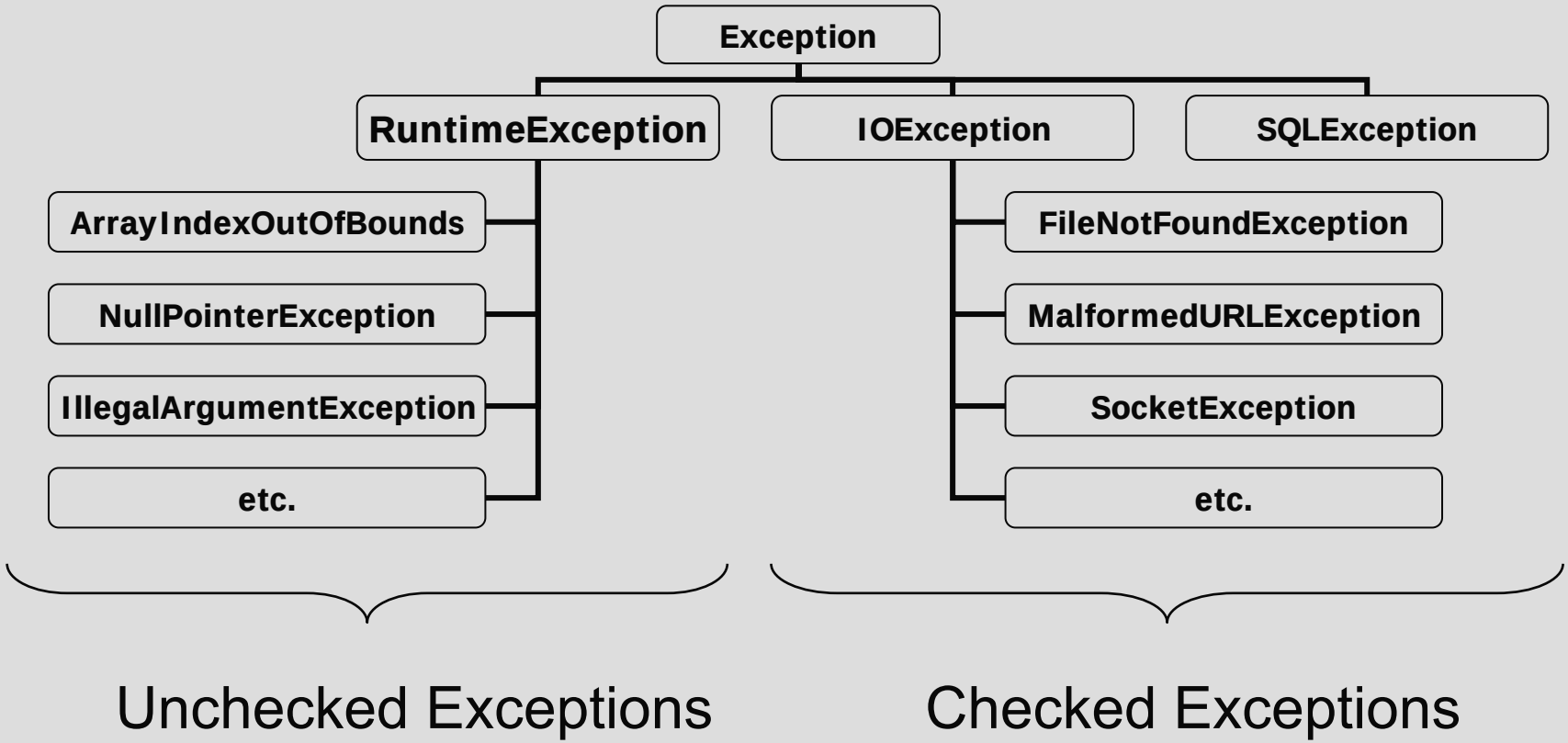
```
void readFile(String filename) {
   try {
       FileReader reader = new
              FileReader("myfile.txt");
       // read from file . . .
    } catch (FileNotFoundException e) {
       System.out.println("file was not found");
    }
}
```

OR

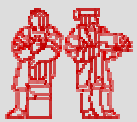
```
void readFile(String filename) throws
   FileNotFoundException {
     FileReader reader = new FileReader("myfile.txt");
     // read from file . . .
}
```



Exception Class Hierarchy



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

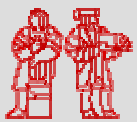


Exceptions and Inheritance

- A method can throw less exceptions, but not more, than the method it is overriding

```
public class MyClass {
   public void doSomething()
               throws IOException, SQLException {
       // do something here
   }
}

public class MySubclass extends MyClass {
   public void doSomething() throws IOException {
       // do something here
   }
}
```



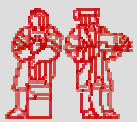
Writing Your Own Exceptions

- There are at least 2 types of exception constructors:
  - Default constructor: No arguments

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

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

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

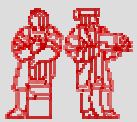


Writing Your Own Exceptions

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

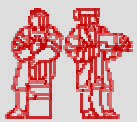
```
public class MyCheckedException extends
   IOException {
     public MyCheckedException() {}
     public MyCheckedException(String m){
       super(m);}
}

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



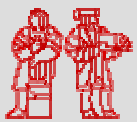
Checked or Unchecked?

- 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



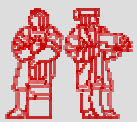
Lecture Summary

- 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 checked exceptions or specify that it may throw them
- If you write your own exception, it must be a subclass of the Exception class and have at least the two standard constructors



EC.S01 Internet Technology in Local and Global Communities
Spring 2005-Summer 2005