



# Exception Handling

Charles R. Hardnett

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346



## Motivation

- Programmers may handle errors with selection constructs
  - Read input
    - Use **if..stmts** to check the validity of the input
  - Perform computation
    - Use **switch/case** statement to react to the results including errors
- What if the error is external to the application?
  - Divide-by-zero
  - Core dump
  - Bus Error
  - User sends **Ctrl-C** or **Ctrl-Break** to the program
  - Error reading file
- These errors are unusual events called **Exceptions**
  - Languages provide special facilities to handle exceptions

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #2)

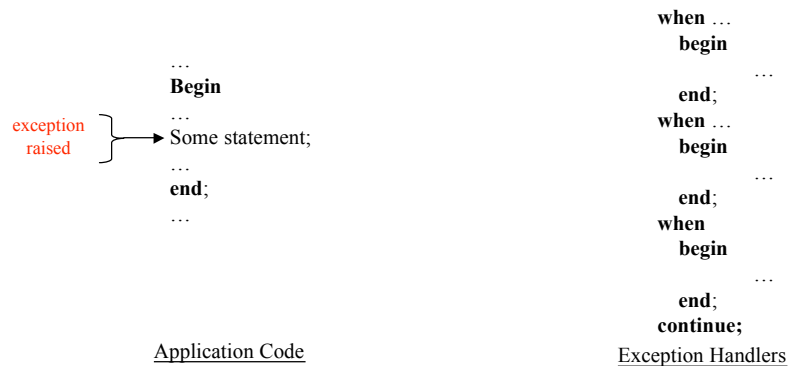


## Exceptions

- **Exception**  
An unusual event, erroneous or not, that is detectable by either hardware or software and that may require special processing.
- **Raising an Exception**  
The act of the hardware or software alert the system when the event occurs.
- **Exception Handler**  
A special unit of code that is invoked when a given exception is raised. This unit of code belongs to the application.



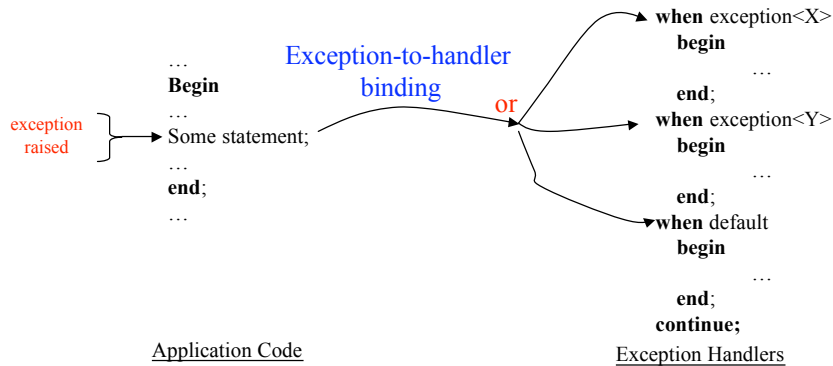
## Control-Flow of Exception Handling



- Exception is raised at the some statement in the application
- The exception handlers may be in the application or some library



## Control-Flow of Exception Handling



- The application is suspended
- Control transferred to the run-time system
- The exception is bound to a specific handler

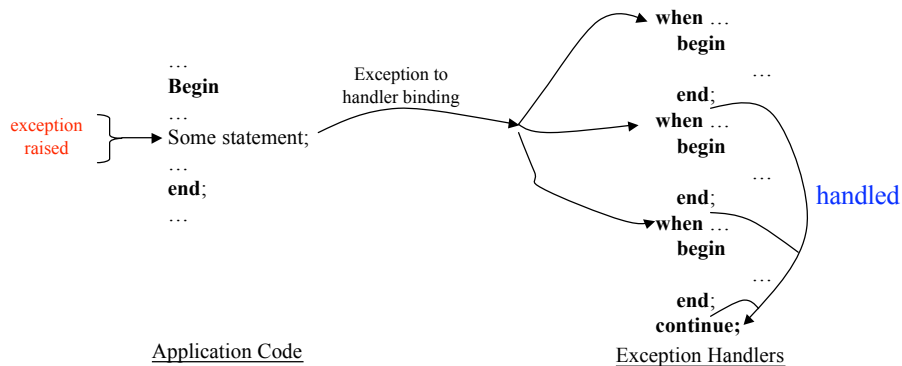
Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #5 )



## Control-Flow of Exception Handling



- Exception-Handler is executed to properly handle the exception
- Application still suspended

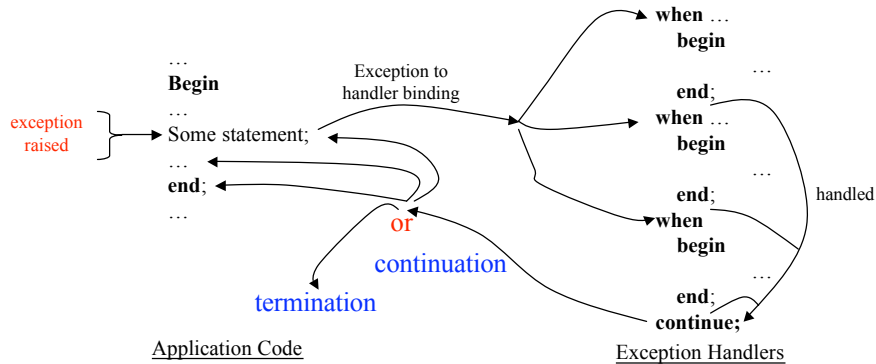
Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #6 )



## Control-Flow of Exception Handling



- Control is transferred back to the application
    - May resume at the “some statement”
    - May resume at the next statement
    - May exit the current block, function, loop, etc.
    - May terminate execution of the application
- language design choice



## Java Exception Handling

- Uses an OOP philosophy to support exception handling
  - A special *Throwable* class is the parent of all classes that handle exceptions
    - **Error** : A subclass of *Throwable*
      - Handled by the Java interpreter
    - **Exception** : A subclass of *Throwable*
      - User-defined exceptions
      - *IOException*, *ArrayIndexOutOfBoundsException*, *NullPointerException* are predefined subclasses
- The *try* clause indicates the region of the program where exceptions can be raised and handled
- The *catch* clauses identify the handlers



## Java Example: Where exceptions can be thrown

```
public class ListOfNumbers {  
    private Vector victor;  
    private static final int size = 10;
```

```
    public ListOfNumbers () {  
        victor = new Vector(size);  
        for (int i = 0; i < size; i++)  
            victor.addElement(new Integer(i));  
    }
```

```
    public void writeList() {  
        PrintWriter out = new PrintWriter(new  
            FileWriter("OutFile.txt");  
  
        for (int i = 0; i < size; i++)  
            out.println("Value at: " + i + " = " +  
                victor.elementAt(i);  
  
        out.close();  
    }  
}
```

if the file does not exist, then an exception is raised:

**java.io.IOException**

Exceptions thrown by the run-time system

if I is outside of the bounds of the Vector:

**java.ArrayOutOfBoundsException**

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #9 )



## Java Example: Solution Part I

```
public class ListOfNumbers {  
    private Vector victor;  
    ...  
  
    try {  
        public void writeList() {  
            PrintWriter out = new PrintWriter(new  
                FileWriter("OutFile.txt");  
  
            for (int i = 0; i < size; i++)  
                out.println("Value at: " + i + " = " +  
                    victor.elementAt(i);  
  
            out.close();  
        }  
    }  
}
```

•The **try block** contains the code that may throw an exception

•There must be a **finally clause** or **catch clause** following the try block

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #10 )



## Java Example: Solution Part II

```
public class ListOfNumbers {
...
try {
public void writeList() {
...FileWriter("OutFile.txt");
...vector.elementAt(i);
}
} catch (ArrayIndexOutOfBoundsException e) {
System.err.println("Caught Index Out Of Bounds
Exception: " + e.getMessage());
} catch (IOException e) {
System.err.println("Caught IO Exception: " +
e.getMessage());
} finally {
if (out != null) {
System.out.println("Closing PrintWriter");
out.close();
} else {
System.out.println("PrintWriter not open");
}
}
}
```

Parameter: Exception Object  
(OOP Design!)

handler: Out of Bounds

handler: IO Exception

Clean up for either exception

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #11)



## Java Example: What happens when the exception is thrown?

```
public class ListOfNumbers {
...
try {
public void writeList() {
...PrintWriter(new
FileWriter("OutFile.txt"));
...vector.elementAt(i);
}
} catch (ArrayIndexOutOfBoundsException e) {
System.err.println("Caught Index Out Of Bounds
Exception: " + e.getMessage());
} catch (IOException e) {
System.err.println("Caught IO Exception: " +
e.getMessage());
} finally {
if (out != null) {
System.out.println("Closing PrintWriter");
out.close();
} else {
System.out.println("PrintWriter not open");
}
}
}
```

**IOException Raised!**

1. The FileWriter("OutFile.txt") statement fails!
2. The run-time system looks for a **try-block**
  - a. **Dynamic Scoping Call Stack:**  
FileWriter(...) ← Top  
PrintWriter(...)  
**WriteList(...)**
3. Look for handler in WriteList try block.
4. Execute handler for IOException
5. Execute the finally block
6. Execute statement following the finally block

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #12)



## Java Example: Where exceptions can be thrown

```
public class ListOfNumbers {  
    private Vector victor;  
    private static final int size = 10;
```

```
    public ListOfNumbers () {  
        victor = new Vector(size);  
        for (int i = 0; i < size; i++)  
            victor.addElement(new Integer(i));  
    }
```

```
    public void writeList() {  
        PrintWriter out = new PrintWriter(new  
            FileWriter("OutFile.txt");  
  
        for (int i = 0; i < size; i++)  
            out.println("Value at: " + i + " = " +  
                victor.elementAt(i);  
  
        out.close();  
    }  
}
```

if the file does not exist, then an  
exception is raised:  
**java.io.IOException**

Exceptions thrown by the  
run-time system

if I is outside of the bounds of the  
Vector:  
**java.ArrayOutOfBoundsException**

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #13 )



## Java Example: Passing the Buck!

```
public class ListOfNumbers {  
    private Vector victor;  
    private static final int size = 10;
```

```
    public ListOfNumbers () {  
        victor = new Vector(size);  
        for (int i = 0; i < size; i++)  
            victor.addElement(new Integer(i));  
    }
```

```
    public void writeList() throws IOException, ArrayOutOfBoundsException {  
        PrintWriter out = new PrintWriter(new  
            FileWriter("OutFile.txt");  
  
        for (int i = 0; i < size; i++)  
            out.println("Value at: " + i + " = " +  
                victor.elementAt(i);  
  
        out.close();  
    }  
}
```

**writeList()** does not have a  
handler, but propagates the  
exception to another method  
that has the handler.

Fall/Spring 2003  
Copyright © 2003 Spelman College

Department of Computer and  
Information Sciences (CIS)

CIS346  
(Lecture #13 Slide #14 )



## Manually Throwing Exceptions

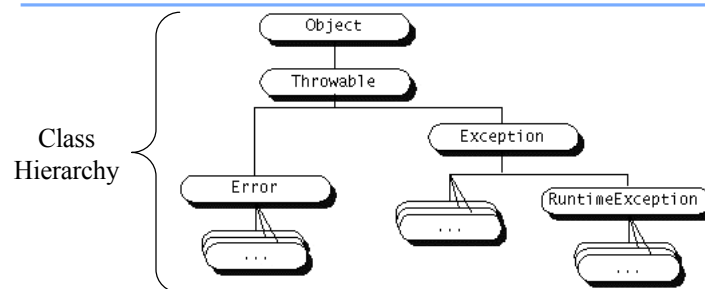
```
public Object pop() throws EmptyStackException {
    Object obj;

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

- The **throws** clause states that the method `pop()` throws the exception `EmptyStackException`
- The **throw** clause manually throws the exception `EmptyStackException`
- `EmptyStackException` is part of the Java Class Library



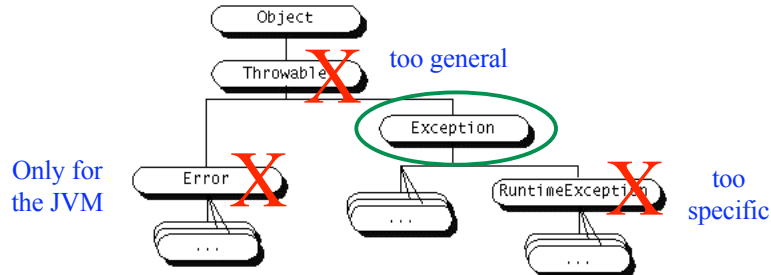
## Creating Your Own Exception



- `Throwable` is the class that supports **throw** and **throws** clauses
- **Error Class**: Hard failure in the JVM, Failure to link at Run-time
- **Exception**: Application level failure, not severe
- **RuntimeException**: Failure in the application execution - Divide-By-Zero



## Choosing Your Classes Parent



```

public class MyException extends Exception {
    public MyException() {...};
    public MyException(String reason) {...};
}
  
```

Required Constructors

- Benefits from inheritance and code-reuse from other classes



## Using MyException

- ```

...
//Uses default constructor
throw new MyException;
...

// Uses 1-parm constructor
throw new
    MyException("Something
        bad happened here");
...
  
```
- Default version does not have a "reason" string
  - The 1-parm version has a "reason" string
  - The throwable class contains a method:  
`String getMessage();`
  - This method can be used to output the "reason" in the handler



## Conclusions

---

- Exceptions are unexpected errors that can be handled by user defined handlers
  - **Control-Flow for handlers is like a “goto”**
- Java uses objects for exceptions
  - **Clean interfaces**
  - **Inheritance used to create user-defined exceptions**
- Other languages may simply use integer types for exceptions
- Return / continuation protocol
  - **Early languages such as PL/I allowed many ways to do it**
  - **Java follows the finally clause**
  - **Other modern languages exit the program or return to the point after the exception is called**
- Ada and Java have very complete exception handling