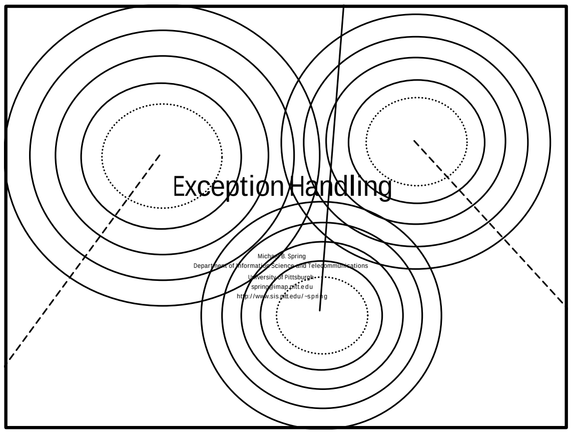
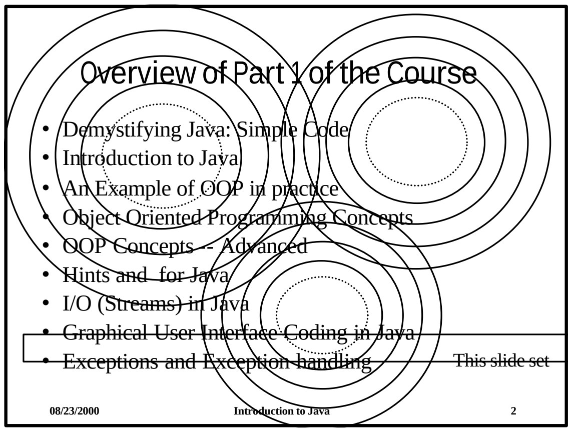


Exception Handling

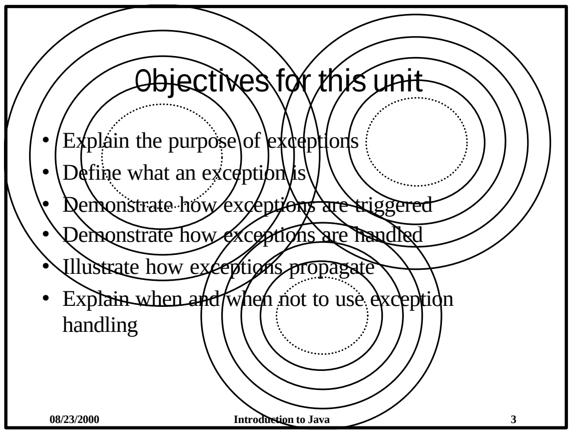
Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring



Overview of Part 1 of the Course

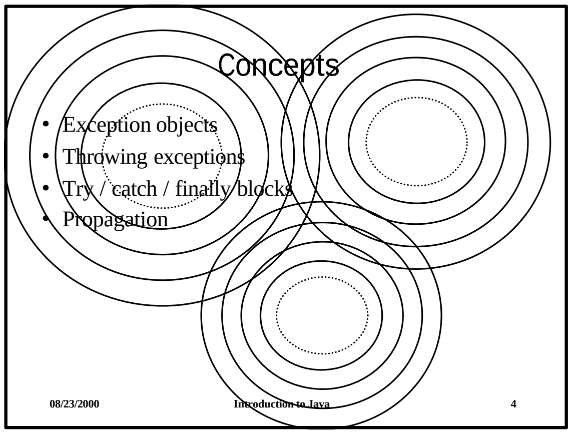
- Demystifying Java: Simple Code
- Introduction to Java
- An Example of OOP in practice
- Object Oriented Programming Concepts
- OOP Concepts -- Advanced
- Hints and for Java
- I/O (Streams) in Java
- Graphical User Interface Coding in Java
- Exceptions and Exception handling

This slide set



Objectives for this unit

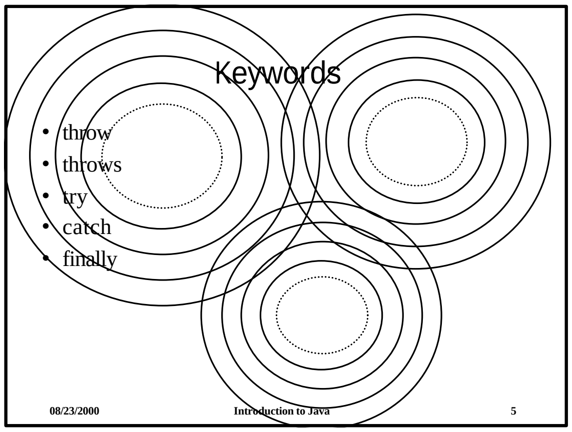
- Explain the purpose of exceptions
- Define what an exception is
- Demonstrate how exceptions are triggered
- Demonstrate how exceptions are handled
- Illustrate how exceptions propagate
- Explain when and when not to use exception handling



Concepts

- Exception objects
- Throwing exceptions
- Try / catch / finally blocks
- Propagation

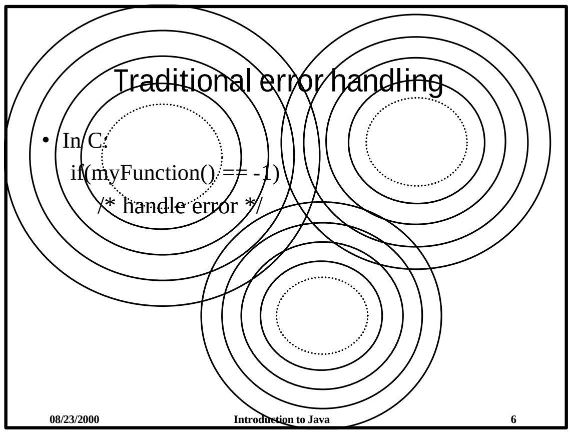
08/23/2000 Introduction to Java 4



Keywords

- throw
- throws
- try
- catch
- finally

08/23/2000 Introduction to Java 5

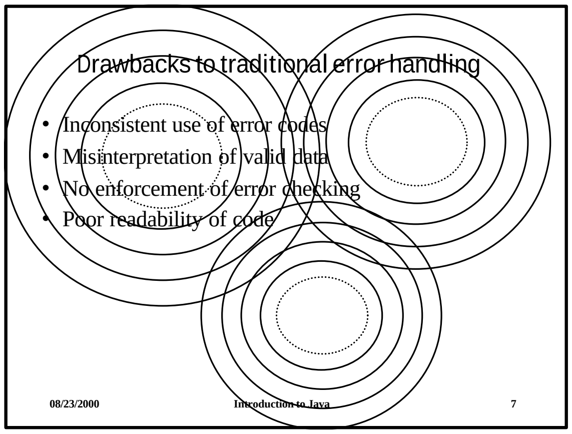


Traditional error handling

- In C:

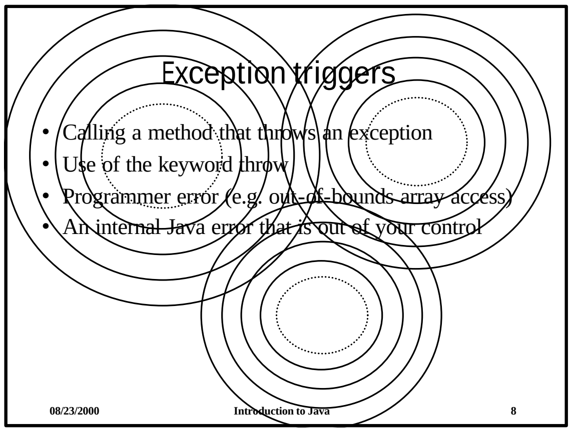
```
 if(myFunction() == -1)
     /* handle error */
```

08/23/2000 Introduction to Java 6



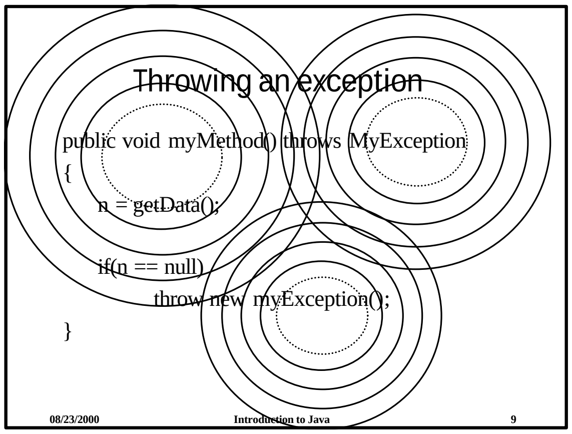
Drawbacks to traditional error handling

- Inconsistent use of error codes
- Misinterpretation of valid data
- No enforcement of error checking
- Poor readability of code



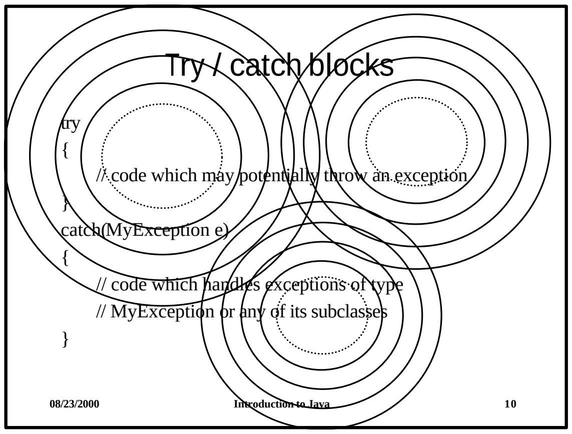
Exception triggers

- Calling a method that throws an exception
- Use of the keyword throw
- Programmer error (e.g. out-of-bounds array access)
- An internal Java error that is out of your control



Throwing an exception

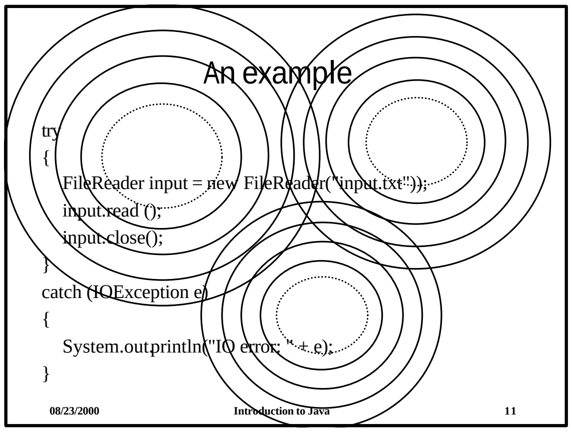
```
public void myMethod() throws MyException
{
     n = getData();

     if(n == null)
             throw new myException();
}
```



Try / catch blocks

```
try
{
    // code which may potentially throw an exception
}
catch(MyException e)
{
    // code which handles exceptions of type
    // MyException or any of its subclasses
}
```

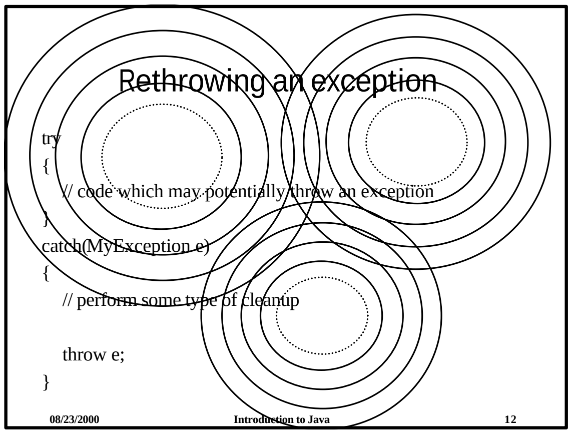
08/23/2000 Introduction to Java 10



An example

```
try
{
  FileReader input = new FileReader("input.txt");
  input.read ();
  input.close();
}
catch (IOException e)
{
  System.out.println("IO error: " + e);
}
```

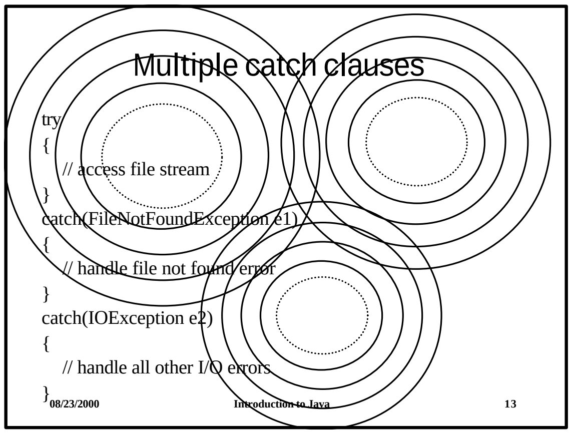
08/23/2000 Introduction to Java 11



Rethrowing an exception

```
try
{
  // code which may potentially throw an exception
}
catch(MyException e)
{
  // perform some type of cleanup

  throw e;
}
```

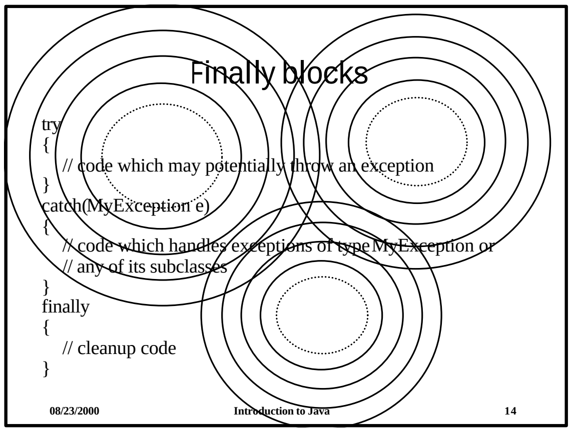
08/23/2000 Introduction to Java 12



Multiple catch clauses

```
try
{
    // access file stream
}
catch(FileNotFoundException e1)
{
    // handle file not found error
}
catch(IOException e2)
{
    // handle all other I/O errors
}
```

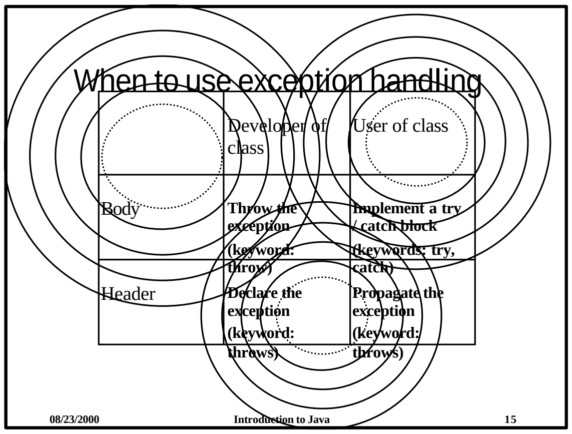
08/23/2000 Introduction to Java 13



Finally blocks

```
try
{
    // code which may potentially throw an exception
}
catch(MyException e)
{
    // code which handles exceptions of type MyException or
    // any of its subclasses
}
finally
{
    // cleanup code
}
```

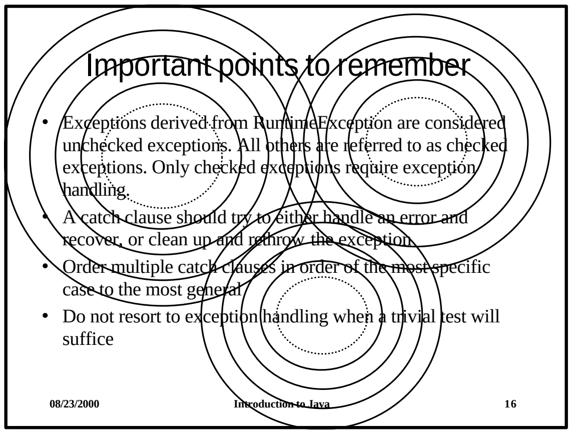
08/23/2000 Introduction to Java 14



When to use exception handling

	Developer of class	User of class
Body	**Throw the exception** **(keyword: throw)**	**Implement a try / catch block** **(keywords: try, catch)**
Header	**Declare the exception** **(keyword: throws)**	**Propagate the exception** **(keyword: throws)**

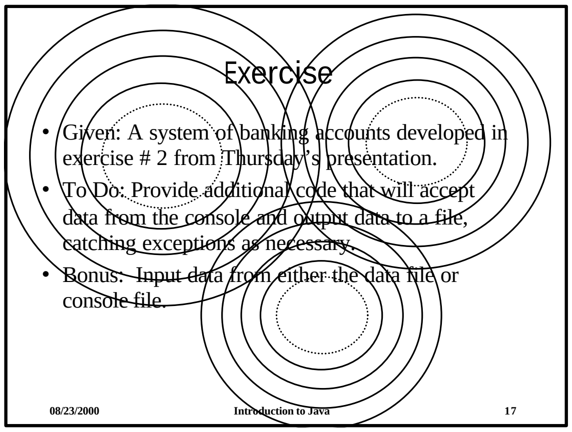
08/23/2000 Introduction to Java 15



Important points to remember

- Exceptions derived from RuntimeException are considered unchecked exceptions. All others are referred to as checked exceptions. Only checked exceptions require exception handling.
- A catch clause should try to either handle an error and recover, or clean up and rethrow the exception
- Order multiple catch clauses in order of the most specific case to the most general
- Do not resort to exception handling when a trivial test will suffice

08/23/2000 Introduction to Java 16



Exercise

- Given: A system of banking accounts developed in exercise # 2 from Thursday's presentation.
- To Do: Provide additional code that will accept data from the console and output data to a file, catching exceptions as necessary.
- Bonus: Input data from either the data file or console file.

08/23/2000 Introduction to Java 17