

Exceptions

Reference:

java.sun.com/docs/books/tutorial/essential/exceptions/

Issues

- What to do when you catch an exception?
- How and when to generate exceptions.
- **RunTime** exceptions.
- Custom Exception types.
- Using **finally**.

Exception Reminder

```
try {
    readFromFile("datafile");
} catch (FileNotFoundException e) {
    System.err.println("Error: File not found");
}
```

Exception Handling: Some Options

- Print something
- Throw a new exception
- Re-Throw the exception
- Fix the problem
- Exit

Java Programming: Exceptions

4

Exception Handling: Printing

- You can print a stack trace by calling the exception method `printStackTrace()`
- Sometimes it's better to send error messages to `stderr`:
 - `System.err.println("Error: invalid thingy");`
- Some applications log error messages
 - file
 - logging service (syslog).

Java Programming: Exceptions

5

Exception Handling: **throw**

- You can **throw** an exception from an exception handler (a **catch** block).
 - Allows you to change exception type and/or error message.
 - You can also alter the base of the stack trace
 - `fillInStackTrace()`

Sample code: [ThrowUp.java](#)

Java Programming: Exceptions

6

Exception Handling: Re-throw

- You can **throw** an exception from an exception handler (a **catch** block) without changing anything:
 - called *rethrowing*
 - The caller needs to deal with the exception.
 - This also happens if you don't catch the exception!
 - sometimes you need to take some action and then rethrow the exception.

Java Programming: Exceptions

7

Another way to re-throw

- You can allow selected types of exceptions to be propagated to the caller of your method:

```
void blah() throws IOException {
```
- Within **blah()** you don't need to catch these exceptions (to be able to compile).

Java Programming: Exceptions

8

Exception Handling: Fix the problem.

- You can't fix things and then *resume* execution automatically
 - you can do this in C++.
- You can have a loop the retries the code again.

Sample code: [Wait.java](#)

Java Programming: Exceptions

9

Exception Handling: exiting

- Sometimes the error is fatal, and you want to stop the program immediately.

```
System.exit();
```

Sample code: [Wait.java](#)

Java Programming: Exceptions

10

How/when do you *generate* exceptions?

- Use throw:

```
throw new Exception("broken!");
```

- You can use throw anywhere.
 - you detect some error that means the following code should not be executed.
- In some cases, you can think of throw as a alternate **return**

Java Programming: Exceptions

11

Exception Enforcement

- In general, you do the following:
 - specify what exceptions each method can generate.
 - write code to catch all exceptions that can be generated by a method call.
- The compiler (usually) enforces this
 - it is a compilation error to call a method without catching it's declared exception types.

Java Programming: Exceptions

12

Sample code: [NullPointerException.java](#)

RunTime Exceptions

- There are exceptions that are generated by the system (that are usually caused by programming mistakes):
 - NullPointerException (null references)
 - ArrayIndexOutOfBoundsException
- If you don't catch these, a stack trace will be generated and the program will terminate.
- The compiler does not force you to catch these exceptions.

Java Programming: Exceptions

13

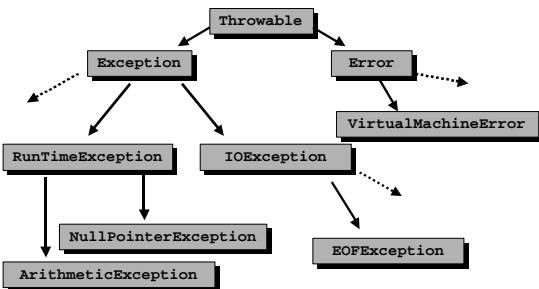
Exception Types

- Exceptions are objects!
- Exception types are classes.
 - A (quite large!) hierarchy of classes.
- All exception types are derived from the class **Exception**
 - there are some methods defined in this base class.

Java Programming: Exceptions

14

Exception Type Hierarchy (partial)



Java Programming: Exceptions

15

Some Exception Methods

- These are actually inherited from **throwable**

`printStackTrace()`

`fillInStackTrace()`

`getMessage()`

Java Programming: Exceptions

16

Creating Your Own Exception Types

- It is often useful to create your own type of exception.
 - generally all you create is a name.
 - you can get fancy and add new methods to your exception class(es).

Java Programming: Exceptions

17

Sample code: [CmdLine.java](#) [CmdLine2.java](#)

Custom Exception Type

```
class FooException extends Exception {}
```

```
class BlahException extends Exception {  
    BlahException(){}  
    BlahException(String s) { super(s); }  
}
```

```
throw new BlahException("Invalid blah");
```

Java Programming: Exceptions

18

using **finally**

```
try {  
    statements . . .  
} catch (ExceptionType1 eName1) {  
    error handling statements . . .  
} catch (ExceptionType2 eName2) {  
    error handling statements . . .  
} finally {  
    ... this code always executed ...  
}
```

Java Programming: Exceptions

19

Why **finally**?

- What is there to clean up?
 - No memory cleanup required in Java!
 - No destructors to call!
- Sometimes you need to set the state of things (fields) to some *stable (acceptable)* state.
Sample code: [FinallyPlay.java](#)

Java Programming: Exceptions

20
