Today's topics

- Go over lab?
- Over-riding and over-loading
- Interactive OOP demo
- Exceptions
- Examples

Overloading methods

- Declaring multiple methods with the same name is called “overloading”.
- Rule for overloading: Java must have enough information to disambiguate.
  - Different parameter lists
  - Overloading example
- Contrast with “Overriding” methods:
  - Same name, same parameters
    - Overridden method is not explicitly declared
  - E.g. Changing default constructor or toString()
  - Related to inheritance (more later)

Interactive OOP demo

```
public class StudentList {

/**
 * Prepends a student to a list.  The resulting list is returned.
 * @param stu the student to prepend
 * @param list the list to which student is prepended
 */
public StudentList(Student stu, StudentList list) {…}
/**
 * Gets the student at position i in the list.
 * Indexing is circular, e.g. get(1) == get(listLength + 1)
 * @param i the position in the list
 * @return a reference to the student at position i
 */
public Student get(int i) {…}

}
```

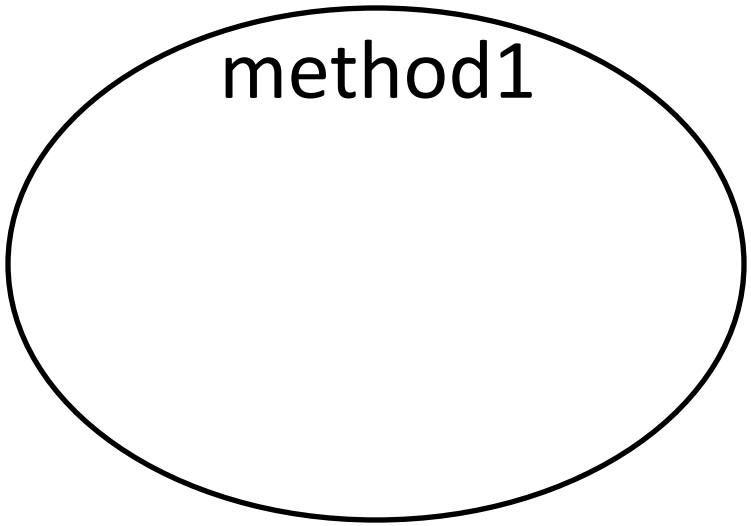
Exceptions

"An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions."

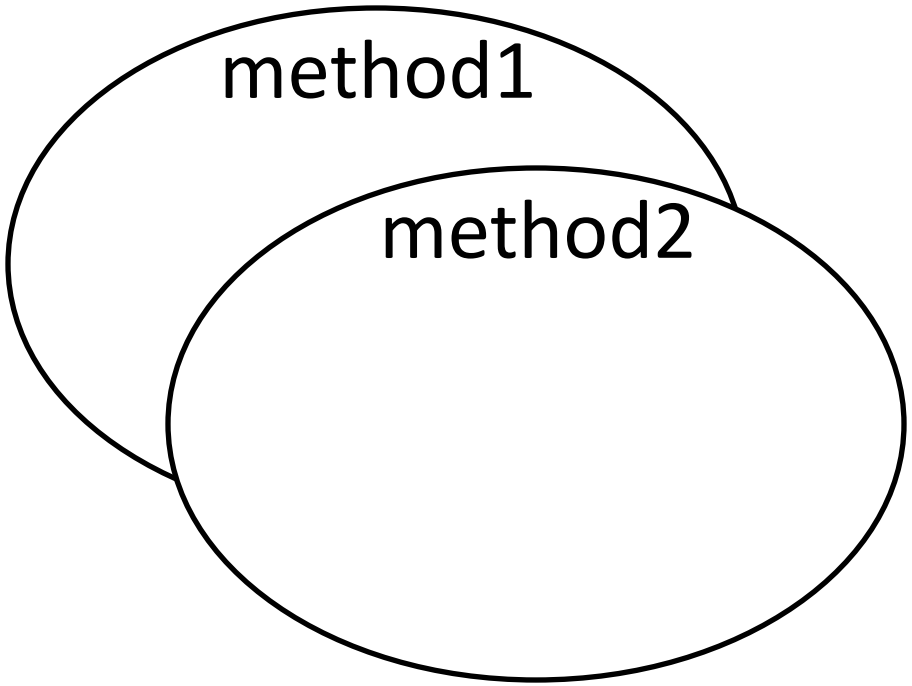
http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

- ArithmeticException: divide by zero
- NullPointerException: dereferencing null
- IOException: hardware malfunction while reading a file
- etc…

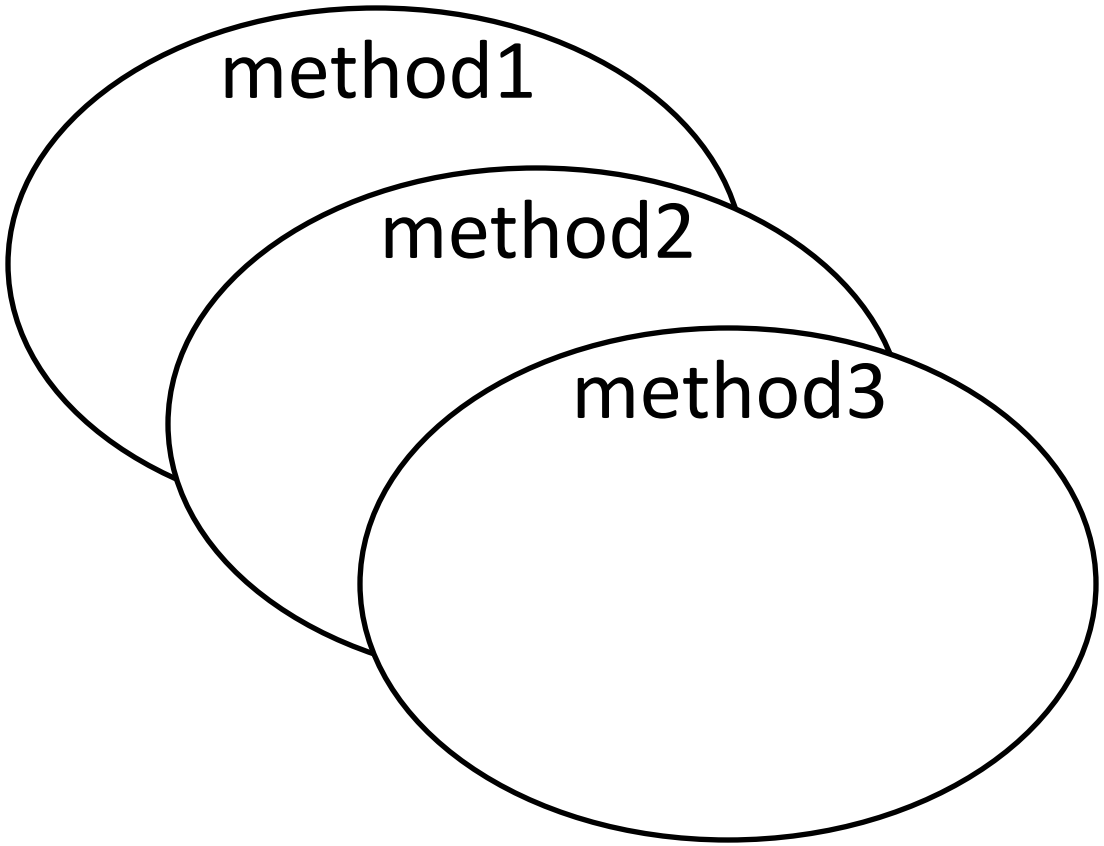
Exceptions and the Stack



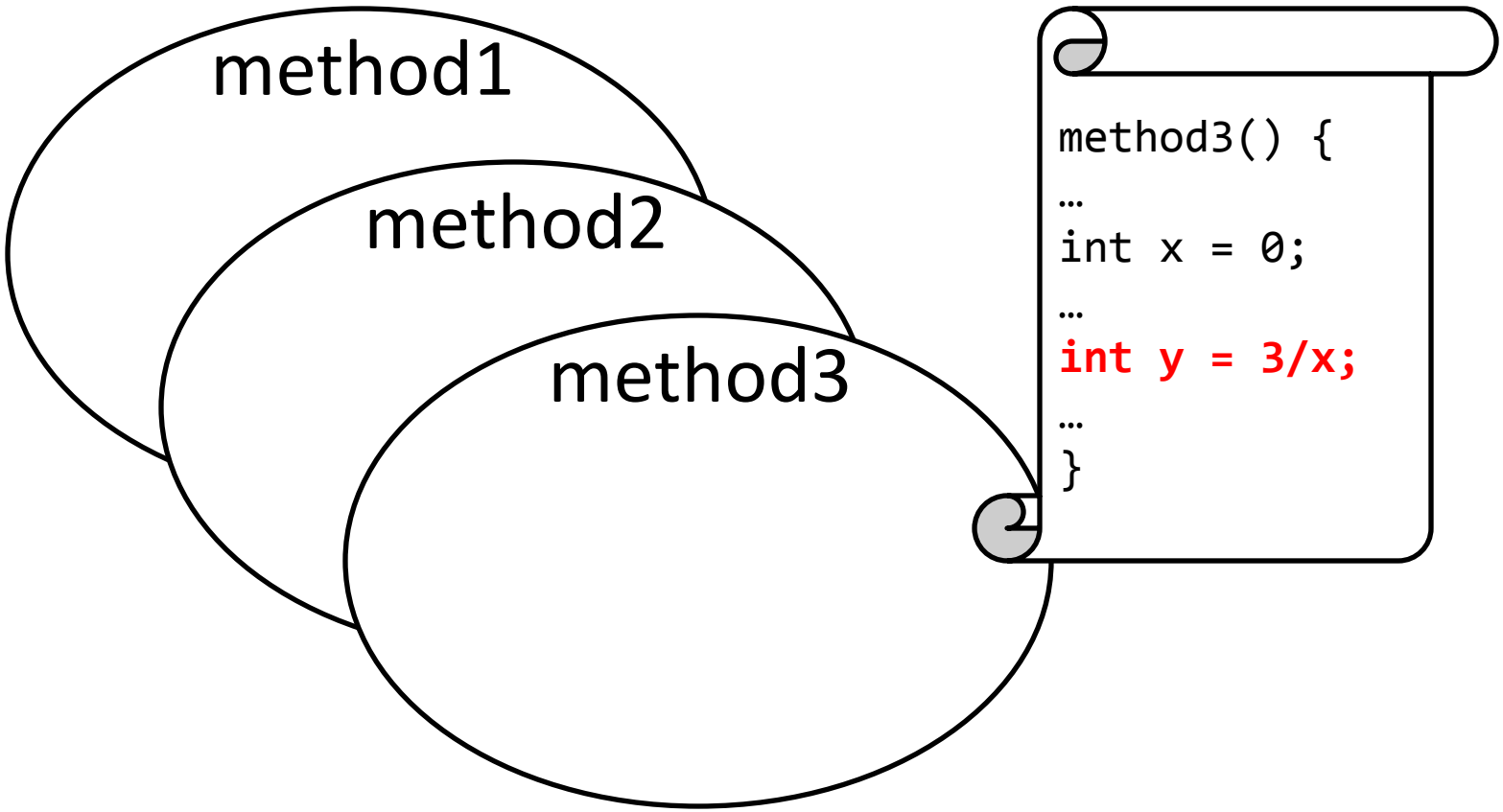
Exceptions and the Stack



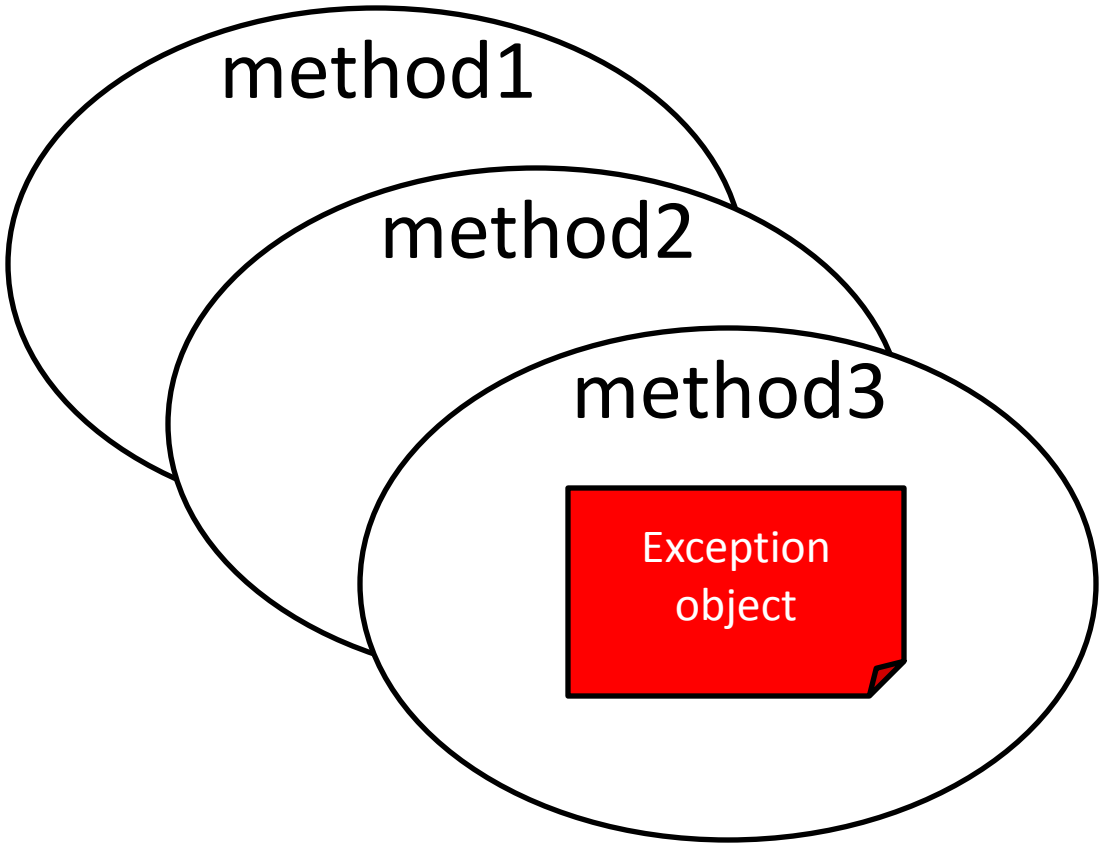
Exceptions and the Stack



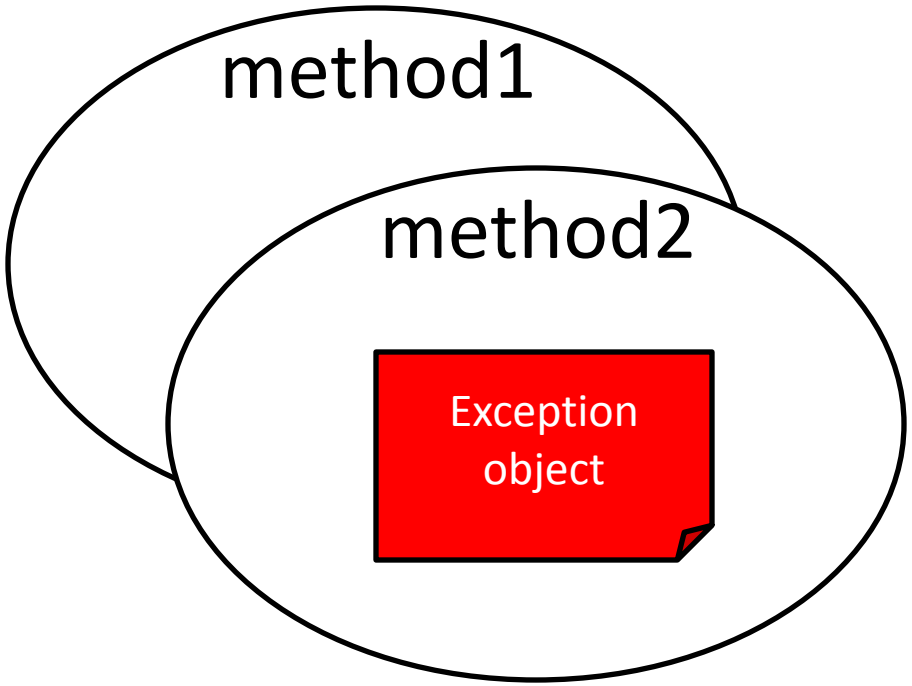
Exceptions and the Stack



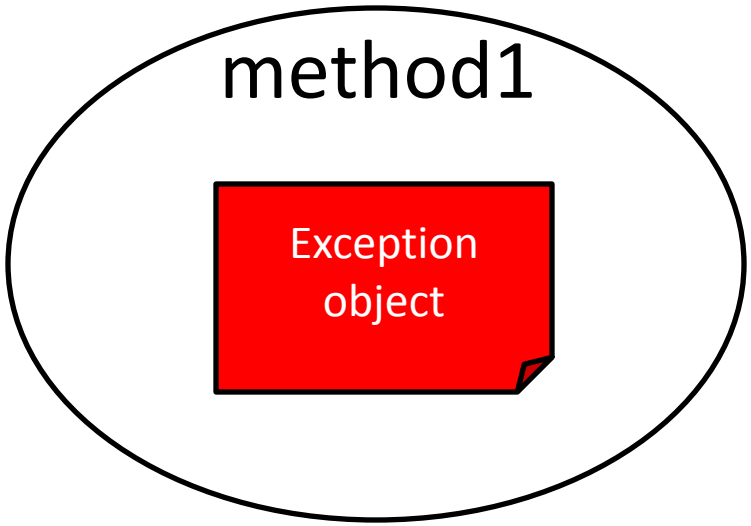
Exceptions and the Stack



Exceptions and the Stack



Exceptions and the Stack



Handling exceptions

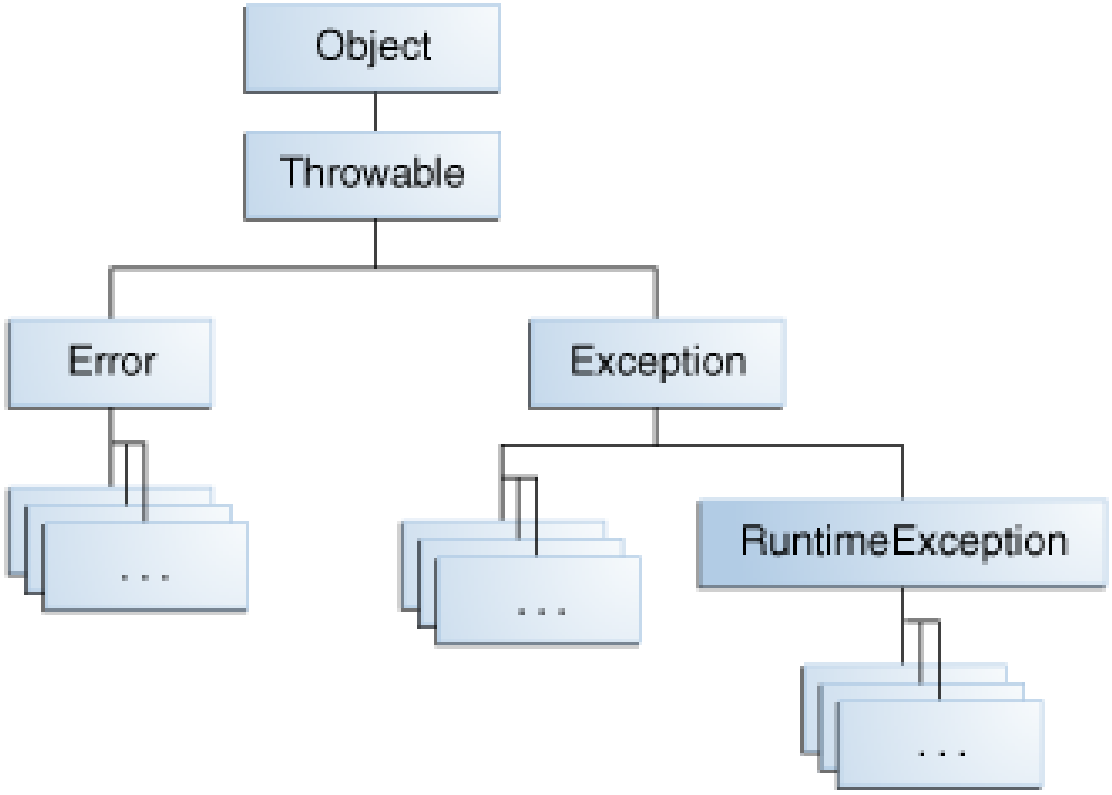
```
try {
      //Exception may or may not occur
} catch(Exception e) {
      //Handle an exception if it does occur
} finally {
      //Always do this (exception or not)
      //e.g. close a file resource like Scanner
}
```

- Multiple catch blocks allowed for multiple types of exceptions
- Finally block may be omitted

Exceptions vs If-else

- Manually handling with if-else
  - Less “bloated” (?)
  - http://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html
- Throwing exceptions
  - You can leave the handling to someone else
    - E.g. your class is a library; different users may need to handle the exceptions in different ways
  - Your method does not have to return when an exception occurs

Exceptions and Errors



http://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html

Exceptions and Errors

- “Unchecked” Exceptions: Issues that should not be handled
  - Errors: serious issues at the JVM level
  - RuntimeExceptions: issues that suggests bad programming
    - E.g. arithmetic or null-pointer
- “Checked” Exceptions: Issues that might reasonably occur and should be handled by the program
  - File not found, prompt user for new choice
  - Any method that throws a checked exception must be surrounded with try/catch

Throwing Exceptions

```
//Methods that might throw checked exceptions
//need the “throws” keyword with the exception
public static void badFile(String str) throws IOException
{
	FileReader f = new FileReader(str);
	//Read from f
	f.close();
}
//You can manually throw exceptions with “throw”
//You can implement your own exceptions (more later)
public static void myMethod() throws MyException {
	throw new MyException();
}
```

Examples

- BigInt factorial
- Console graphics