CSIS 3103

Ch 1: Object-oriented Programming

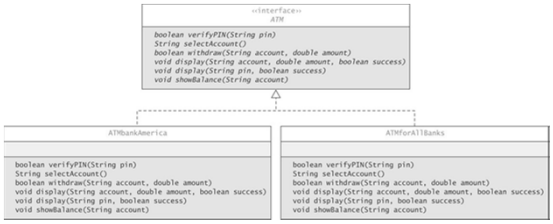
Interfaces, Inheritance, Exceptions

Interfaces

A way to specify an ADT
- Declare names, parameters, and return values of the methods
- No implementation of methods operations

- Classes that implement an interface must satisfy this *contract*
  - Implement ***all*** methods declared in the interface
- No specification of internal representation of data

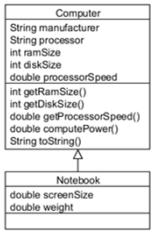
Interface Example



Reusable Code

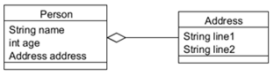
- Write less new code
- Tested and debugged so more reliable
  - Classes in the Java API
  - Java Collections Framework for data structures
- Extend existing classes (inheritance)

An *is-a* relationship indicates inheritance



A Laptop *is a* Computer

The *has-a* relationship means that one class has the second class as an attribute



A Person *has an* Address

Overload, Override, and Polymorphism

Override - Same method name and same parameters in subclass

Overload - Same method name, different parameters

Polymorphism - method implementation used at runtime is selected based on the type of object

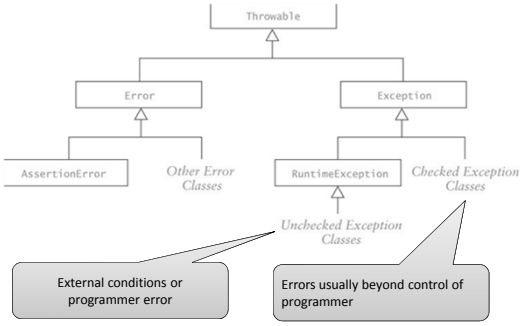
Program Defects and "Bugs"

Types of defects
- Syntax errors
- Run-time errors
- Logic errors

Example Exceptions

Class	Cause/Consequence
`ArithmeticException`	An attempt to perform an integer division by zero.
`ArrayIndexOutOfBoundsException`	An attempt to access an array element using an index (subscript) less than zero or greater than or equal to the array's length.
`NumberFormatException`	An attempt to convert a string that is not numeric to a number.
`NullPointerException`	An attempt to use a `null` reference value to access an object.
`NoSuchElementException`	An attempt to get a next element after all elements were accessed.
`InputMismatchException`	The token returned by a `Scanner next...` method does not match the pattern for the expected data type.

The `Exception` Class Hierarchy



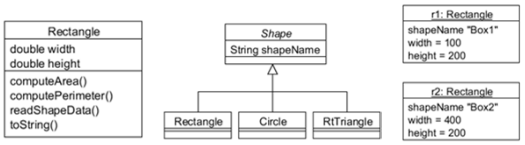
Handling Exceptions

`try - catch` : Use when an error could occur and there is a reasonable way to recover

`throw` : Use when an error could occur and there's no obvious way to recover

Three Inter-related OOP Constructs

Classes, inheritance, and objects



A *class* defines the structure of its objects

Classes are organized in a hierarchy defined by "is-a" relationships

Each object has its own state

Packages

- A Java *package* is a group of *cooperating classes*
- The Java API is organized into packages
- Indicate the package of a class at the top of the file:
  `package classPackage;`
- Classes in the *same package* must be in the *same directory* (folder)
- The folder must have the same name as the package

Packages and Visibility

- Classes *not* part of a package can only access `public` members of classes in the package
- If a class is not part of the package, it must access the public classes by their complete name, which would be `packagename.className`
- For example,

```
x = Java.awt.Color.GREEN;
```

- If the package is imported, the `packageName` prefix is not required.

```
import java.awt.Color;
...
x = Color.GREEN;
```

The Default Package

- Files which do not specify a package are part of the default package
- If you do not declare packages, all of your classes belong to the default package
- The default package is intended for use during the early stages of implementation or for small prototypes
- When you develop an application, declare its classes to be in the same package