Data Structures and Algorithms

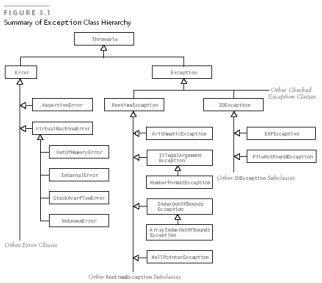
Prof. Nadeem Abdul Hamid
CSC 220 - Fall 2005
Lecture Unit 3 - Inheritance

Inheritance

- Besides reusing existing classes for new applications, OOP allows definition of new classes that **extend** existing ones to provide additional functionality
- *Subclass* inherits data fields and methods of the *superclass*
- In Java, all classes part of inheritance hierarchy
- Class **Object** is the superclass of all Java classes

Class Hierarchy

FIGURE 3.1
Summary of Exception Class Hierarchy

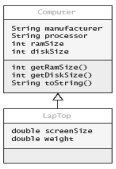


Is-A vs. *Has-A* Relationships

- "Jet plane is an airplane"
  - JetPlane class will extend Airplane
- "Jet plane has a jet engine"
  - JetPlane class will have a JetEngine field

```
public class JetPlane extends Airplane {
   private JetEngine[] jets;
   ...
}
```


FIGURE 3.2
Classes LapTop and Computer



Superclass/Subclass Example

```
/** Class that represents a computer.
 *  @author Koffman & Wolfgang
 *  */
public class Computer {
  // Data Fields
  private String manufacturer;
  private String processor;
  private int ramSize;
  private int diskSize;

  // Methods
  /** Initializes a Computer object with all properties specified.
        @param man The computer manufacturer
        @param processor The processor type
        @param ram The RAM size
        @param disk The disk size
   */
  public Computer(String man, String processor, int ram, int disk) {
    manufacturer = man;
    this.processor = processor;
    ramSize = ram;
    diskSize = disk;
  }

 // Insert other accessor and modifier methods here.

  public String toString() {
    String result = "Manufacturer: " + manufacturer +
        "\nCPU: " + processor +
        "\nRAM: " + ramSize + " megabytes" +
        "\nDisk: " + diskSize + " gigabytes";
    return result;
  }
}
```


Subclass - Laptop

```
/** Class that represents a lap top computer.
 *  @author Koffman & Wolfgang
 *  */

public class LapTop
    extends Computer {
  // Data Fields
  private static final String DEFAULT_LT_MAN = "MyBrand";
  private double screenSize;
  private double weight;

  /** Initializes a LapTop object with all properties specified.
        @param man The computer manufacturer
        @param proc The processor type
        @param ram The RAM size
        @param disk The disk size
        @param screen The screen size
        @param wei The weight
   */
  public LapTop(String man, String proc, int ram, int disk,
                double screen, double wei) {
    super(man, proc, ram, disk);
    screenSize = screen;
    weight = wei;
  }

}
```

Inheritance Issues

- Use of this.
- Initializing data fields in subclass
- Use of super()
- No-parameter constructor
- **protected** visibility

Method Overriding

```
public class TestComputerAndLaptop {

 /** Tests classes Computer and LapTop. Creates an object of each and
      displays them.
     @param args[] No control parameters
     @author Koffman & Wolfgang
  */
 public static void main(String[] args) {
   Computer myComputer = new Computer("Acme", "Intel P4 2.4", 512, 60);
   LapTop yourComputer = new LapTop("DellGate", "AMD Athlon 2000", 256, 40,
                                    15.0, 7.5);
   System.out.println("My computer is:\n" + myComputer.toString());
   System.out.println("\nYour computer is:\n" +
                      yourComputer.toString());
 }
}
```

```
My computer is:
Manufacturer: Acme
CPU: Intel P4 2.4
RAM: 512 megabytes
Disk: 60 gigabytes

Your computer is:
Manufacturer: DellGate
CPU: AMD Athlon 2000
RAM: 256 megabytes
Disk: 40 gigabytes
```


Method Overriding

- In the Laptop class:

```
public String toString() {
   String result = super.toString()
       + "\nScreen size: " + screenSize + " inches"
       + "\nWeight: " + weight + " pounds";
   return result;
 }
```

```
My computer is:
Manufacturer: Acme
CPU: Intel P4 2.4
RAM: 512 megabytes
Disk: 60 gigabytes

Your computer is:
Manufacturer: DellGate
CPU: AMD Athlon 2000
RAM: 256 megabytes
Disk: 40 gigabytes
Screen size: 15.0 inches
Weight: 7.5 pounds
```


FIGURE 3.1
Revised UML Diagram for Computer Class Hierarchy

Method Overloading

```
/** Initializes a LapTop object with all properties specified.
      @param man The computer manufacturer
      @param proc The processor type
      @param ram The RAM size
      @param disk The disk size
      @param screen The screen size
      @param wei The weight
 */
 public LapTop(String man, String proc, int ram, int disk,
               double screen, double wei) {
   super(man, proc, ram, disk);
   screenSize = screen;
   weight = wei;
 }

 /** Initializes a LapTop object with 5 properties specified. */
 public LapTop(String proc, int ram, int disk,
               double screen, double wei) {
   this(DEFAULT_LT_MAN, proc, ram, disk, screen, wei);
 }
```


Polymorphism

- Important feature of OOP languages
- Object variable may contain reference to the specified class, or any subclass thereof
- Enables JVM to determine which (overridden) method to invoke at runtime based on type of the object reference

```
Computer comp[] = new Computer[3];
comp[0] = new Computer("Acme", "Intel P4 2.4", 512, 60);
comp[1] = new LapTop("DellGate", "AMD Athlon 2000", 256, 40,
                 15.0, 7.5);
comp[2] = comp[0];
...
for ( int i = 0; i < comp.length; i++ )
    System.out.println( "Computer " + (i+1) + ": \n"
                        + comp[i] );
```


Abstract Classes

- Can declare abstract methods like an interface, which must be implemented by subclasses
- Cannot be instantiated
- Can have field definitions and method bodies

Summary Table

TABLE 3.1
Comparison of Actual Classes, Abstract Classes, and Interfaces

Property	Actual Class	Abstract Class	Interface
Instances (objects) of this can be created	Yes	No	No
This can define instance variables and methods	Yes	Yes	No
This can define constants	Yes	Yes	Yes
The number of these a class can extend	0 or 1	0 or 1	0
The number of these a class can implement	0	0	Any number
This can extend another class	Yes	Yes	No
This can declare abstract methods	No	Yes	Yes
Variables of this type can be declared	Yes	Yes	Yes

13

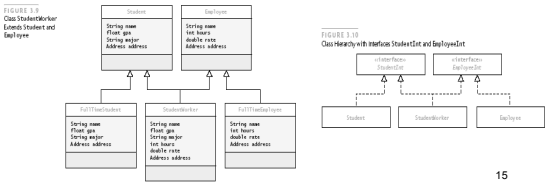
Multiple Inheritance

- Multiple inheritance: the ability to extend more than one class
- Multiple inheritance is a language feature that is difficult to implement and can lead to ambiguity
  - Therefore, Java does not allow a class to extend more than one class

14

Using Multiple Interfaces

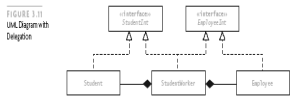
- If we define two interfaces, a class can implement both
- Multiple interfaces emulate multiple inheritance



15

Implementing Reuse Through Delegation

- You can reduce duplication of modifications and reduce problems associated with version control through a technique known as delegation
- In delegation, a method of one class accomplishes an operation by delegating it to a method of another class



16

Packages

- The Java API is organized into packages
- The package to which a class belongs is declared by the first statement in the file in which the class is defined using the keyword **package** followed by the package name
- All classes in the same package are stored in the same directory or folder
- All the classes in one folder must declare themselves to be in the same package
- Classes that are not part of a package may access only public members of classes in the package

17

The No-Package-Declared Environment and Package Visibility

- There exists a default package
  - Files that do specify a package are considered part of the default package
- If you don't declare packages, all of your packages belong to the same, default package
- Package visibility sits between private and protected
  - Classes, data fields, and methods with package visibility are accessible to all other methods of the same package but are not accessible to methods outside of the package
  - Classes, data fields, and methods that are declared protected are visible to all members of the package

18

Visibility (Access Controls)

TABLE 3.3
Summary of Kinds of Visibility

Visibility	Applied to Classes	Applied to Class Members
`private`	Applicable to inner classes. Accessible only to members of the class in which it is declared.	Visible only within this class.
Default or package	Visible to classes in this package.	Visible to classes in this package.
`protected`	Applicable to inner classes. Visible to classes in this package and to classes outside the package that extend the class in which it is declared.	Visible to classes in this package and to classes outside the package that extend this class.
`public`	Visible to all classes.	Visible to all classes. The class defining the member must also be public.

19