

Inheritance

COP3330 - Fall 2006
University of Central Florida

H. Schwartz

Outline

- Overriding and Overloading
- Inheritance in general
- Polymorphism
- Abstract Classes
- Interfaces

Overriding

- Methods of extended class are inherited
- Every class in java automatically extends the Object class
- Overriding is when a method of the extended class is redone.
 - Ex. toString

COP 3330 - Fall 2006

Overloading

- Writing a method which takes in different parameters
 - Ex. multiple constructors

```
public void method(String input){
    //does something
}
public void method(int input){
    //does same thing, but based on different input
}
public void method(){
    //does same thing with no input
}
```

COP 3330 - Fall 2006

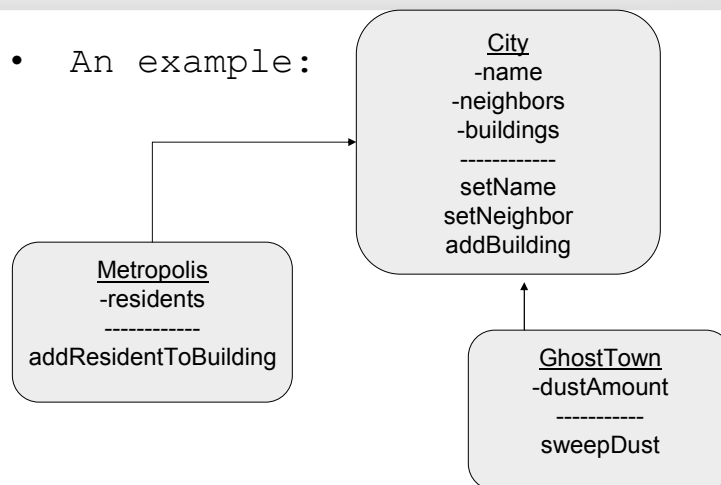
Inheritance in general

- Why inherit?
 - Organization
 - From general to specific
 - Reuse of code

COP 3330 - Fall 2006

Inheritance in general

- An example:



COP 3330 - Fall 2006

Inheritance

- super keyword
 - similar to "this", but accesses the superclasses methods and variables
 - works with constructor as well

```
public class Metropolis extends City {
    public Metropolis(String name, Resident[] residents){
        super(name); //calls City constructor
    }
    public addResidentToBuilding(Resident r, int buildNum){
        //adds a resident based on a building number
        Building b = super.getBuilding(buildNum);
        b.addResident(r);
    }...
}
```

COP 3330 - Fall 2006

Polymorphism

"a code expression can invoke different methods
depending on the types of objects using the code"

- Java 5.0 Program Design

```
Object [] fiveObjs = new Object[5];
fiveObjs[0] = new City("Citia");
fiveObjs[1] = new World("Worldia");
fiveObjs[2] = new PlayerScore("namia", 10);
fiveObjs[3] = new AdvancedHelloWorld();
fiveObjs[4] = new String("stringia");
for(Object o: fiveObjs){
    System.out.println(o.toString());
    //would call toString anyway, but demonstrating point
```

COP 3330 - Fall 2006

Default Constructor

- A constructor with no parameters (no external guidance)
- Unless a super constructor is explicitly called, the default constructor of the super class is called when a new object is constructed.

COP 3330 - Fall 2006

Abstract Class

- A class that must be extended to be used (can't instantiate directly)
- An abstract class may have abstract methods, which must be implemented by classes which inherit the abstract class.

COP 3330 - Fall 2006

Abstract Class

- An Example:

```
abstract public class Vehicle {
    //an abstract class for living organisms to inherit

    //instance variables keep track of World which vehicle
    //moves within
    private World world;

    public World getWorld(){
        //this non-abstract method does need to be redefined
        return this.world;
    }

    //must be defined by inheriting classes
    abstract public void move(String direction, int distance);
}
```

COP 3330 - Fall 2006

Interfaces

- Not a class
- *Partial template of what must be in a class that implements the interface.*

Interface

- An Example:

```
public interface Inhabitable {  
    //interface to specify that something has Residents  
  
    public void addResident(Resident r);  
    //must be able to add a resident  
  
    public void getResdient(index i);  
    //and must be able to get a resident  
}
```

COP 3330 - Fall 2006

Abstract Class vs. Interface

Abstract Class	Interface
Methods defined	No method definitions
Methods have any visibility	Methods have only public visibility
Variables with any modifiers	Variables only with public, final, and static modifiers
Classes can only extend one at a time	Classes can implement multiple at a time

COP 3330 - Fall 2006

Late Binding

- With inheritance we may have a methods defined many times at different levels.
- Which method is actually run depends on which object was originally called.
 - Thus, a method may do different things depending on the object it is being called from.

COP 3330 - Fall 2006

Late Binding

- Example

```
...  
//from some other class:  
A a = new A();  
B b = new B();  
a.m();  
b.m();  
//What is the output?
```

```
public class A {  
    public void m(){  
        System.out.println("m from A");  
        this.n();  
    }  
    public void n(){  
        System.out.println("n from A");  
    }  
}
```

```
public class B extends A {  
    public void n(){  
        System.out.println("n from B");  
    }  
}
```

COP 3330 - Fall 2006