SRI AKILANDESWARI WOMEN'S COLLEGE, WANDIWASH

INHERITANCE

Class : II B.C.A

Mrs.R.SAIKUMARI

Assistant Professor

Department of Computer Applications

SWAMY ABEDHANADHA EDUCATIONAL TRUST, WANDIWASH

Inheritance

- On the surface, inheritance is a code re-use issue.
  - we can *extend* code that is already written in a manageable manner.
- Inheritance is more
  - it supports polymorphism at the language level

Inheritance

- Take an existing object type (collection of fields and methods) and extend it.
  - create a special version of the code without re-writing any of the existing code (or even explicitly calling it!).
  - End result is a more *specific* object type, called the sub-class / derived class / child class.
  - The original code is called the superclass / parent class / base class.

Inheritance Example

- Employee: name, email, phone
  - FulltimeEmployee: also has salary, office, benefits, …
    - Manager: CompanyCar, can change salaries, rates contracts, offices, etc.
  - Contractor: HourlyRate, ContractDuration, …

- A *manager* is a special kind of FullTimeEmployee, which is a special kind of Employee.

Polymorphism

- Create code that deals with general object types, without the need to know what specific type each object is.
- Generate a list of employee names:
  - all objects derived from Employee have a name field since Employee has a name field
  - no need to treat managers differently from anyone else.

Method Polymorphism

- The real power comes with methods/behaviors.
- A better example:
  - shape object types used by a drawing program.
  - we want to be able to handle any kind of shape someone wants to code (in the future).
  - we want to be able to write code now that can deal with shape objects (without knowing what they are!).

Shapes

- Shape:
  - color, layer fields
  - draw() draw itself on the screen
  - calcArea() calculates it's own area.
  - serialize() generate a string that can be saved and later used to re-generate the object.

Kinds of Shapes

- Rectangle
- Triangle
- Circle

Each could be a kind of shape (could be specializations of the shape class).

Each knows how to draw itself, etc.

Could write code to have all shapes draw themselves, or save the whole collection to a file.

class definition

```
class classname {
    field declarations
    { initialization code }
    Constructors
    Methods
}
```

`Abstract` Class modifier

- Abstract modifier means that the class can be used as a superclass only.
  - no objects of this class can be created.
  - can have attributes, even code
    - all are inherited
    - methods can be overridden
- Used in inheritance hierarchies

Interesting Method Modifiers

- **`private/protected/public`**:
  - protected means private to all but subclasses
  - what if none of these specified?
- **`abstract`**: no implementation given, must be supplied by subclass.
  - the class itself must also be declared **`abstract`**
- **`final`**: the method cannot be changed by a subclass (no alternative implementation can be provided by a subclass).

Interesting Method Modifiers

(that have nothing to do with this slide set)

- **`native`**: the method is written in some local code (C/C++) - the implementation is not provided in Java (recall assembler routines linked with C)
- **`synchronized`**: only one thread at a time can call the method (later)

Inheritance vs. Composition

- When one object type depends on another, the relationship could be:
  - *is-a*
  - *has-a*
- Sometimes it's hard to define the relationship, but in general you use composition (aggregation) when the relationship is *has-a*

Composition

- One class has instance variables that refer to object of another.
- Sometimes we have a collection of objects, the class just provides the glue.
  - establishes the relationship between objects.
- There is nothing special happening here (as far as the compiler is concerned).

Inheritance

- One object type is defined as being a special version of some other object type.
  - a *specialization*.
- The more general class is called:
  - base class, super class, parent class.
- The more specific class is called:
  - derived class, subclass, child class.

Inheritance

- A derived class object is an object of the base class.
  - is-a, not has-a.
  - all fields and methods are *inherited*.
- The derived class object also has some stuff that the base class does not provide (usually).

Java Inheritance

- Two kinds:
  - implementation: the code that defines methods.
  - interface: the method prototypes only.

- Other OOP languages often provide the same capabilities (but not as an explicit option).

Implementation Inheritance

- Derived class inherits the implementations of all methods from base class.
  - can replace some with alternatives.
  - new methods in derived class can access all non-private base class fields and methods.

- This is similar to (simple) C++ inheritance.

Accessing superclass methods from derived class.

- Can use **`super()`** to access all (non-private) superclass methods.
  - even those replaced with new versions in the derived class.
- Can use **`super()`** to call base class constructor.
  - use arguments to specify desired constructor

Single inheritance only (implementation inheritance).

- You can't *extend* more than one class!
  - the derived class can't have more than one base class.

- You can do multiple inheritance with *interface inheritance.*

Casting Objects

- A object of a derived class can be cast as an object of the base class.
  - this is much of the power!
- When a method is called, the selection of which version of method is run is totally dynamic.
  - overridden methods are dynamic.

Note: Selection of *overloaded* methods is done at compile time. There are some situations in which this can cause confusion.

The class **`Object`**

- Granddaddy of all Java classes.
- All methods defined in the class Object are available in every class.
- Any object can be cast as an `Object`.

Interfaces

- An interface is a definition of method prototypes and possibly some constants (static final fields).
- An interface does not include the implementation of any methods, it just defines a set of methods that could be implemented.

interface implementation

- A class can **`implement`** an interface, this means that it provides implementations for all the methods in the interface.

- Java classes can implement any number of interfaces (multiple interface inheritance).

Interfaces can be extended

- Creation (definition) of interfaces can be done using inheritance:
  - one interface can extend another.

- Sometimes interfaces are used just as labeling mechanisms:
  - Look in the Java API documentation for interfaces like **`Cloneable`**.
- Example: BubbleSort w/ SortInterfaceDemo