Practice of Programming using Java
Lecture 7 June 24, 2006. 2-5 pm

Rushikesh K Joshi
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Constructors in Inheritance

```
class A {
        ....
        A (int x) {...}
        ....
}
class B extends A {
        ....
        B(int y) { ....}          how to pass parameters to A(int x)?
        ...
}
class User {
            public static void main (String args[]) {
                B b = new B(10);
            }
}
```

Constructors in Inheritance

```
class A {
        ....
        A (int x) {...}
        ....
}
class B extends A {
        ....
        B(int y) {
                super (2*y);
                 ....}
        ...
        }
}
```

Default Constructors in Inheritance

```
class A {
        int id;
        A (int x) {id=x}
        A () {id=unassigned;}
        ....
}
class B extends A {
        int hostelid;
        B(int y) {hostelid=y}
        B(int y, int x) { super(x); hostelid = y;}
        ...
        }
}
```

Method Overloading

More than one method implementation with same method name

```
Class A {
    public void f (int x) {  ... }
    public void f () {  ...  }
    public void f (String s) {......}
    public void f (String s[]) {.....} }
a.f();        a.f(10);        a.f(args[i]);        a.f(args);
```

Overloading vs. Overriding

- Overloading
  - Signatures of methods are different
  - Which method to select?
    - resolved with the help of type signatures
- Overriding
  - Signatures of methods are same, but they appear in different classes within an inheritance tree
  - Which method to select?
    - Resolved at runtime as discussed in last class

Multiple inheritance

A class cannot be extended from multiple classes

A class can implement multiple interfaces

An interface can extend multiple interfaces

* Diamonds formulated with these rules do not create any problems of ambiguities

Abstraction control through interfaces

i1 = {1,2} i2 extends i1 {3}, i3 = {4,5}, i4 extends i3 {6}

C implements i2, i4 { 1,2,3,4,5,6 code}

1,2,3,4,5,6 functions implemented in class C

c=new C();

class A1(i1 anobj) will be able to use 1 and 2 of C

as in A1 a1 = new A1 (c)

class A2(i2 anobj) should use only 1,2,3 of C

class A3 should use only 4,5 of C

class A4 should use only 4,5,6 of C

Extending Interfaces

```
Interface A {
        public void f();
}
interface B  {
        public void g();
}
interface C extends A, B {
        public void h();
}
class Impl implements  C {...}
```

Multiple Inheritance with Interfaces

```
Interface A { f }
interface B { g }
interface C { h }
interface D extends A, B  { k }
class Myclass implements C { h}
class MyAnotherClass extends Myclass implements D, C { f g k }
```

Abstract Classes

Specify partial behavior

Some behavior is unknown and can be left out for variants

Mostly used when there are multiple possible variant subclasses

Cannot be instanciated

```
abstract class A {
        protected int i;
        public void m() {i=20;}
        public abstract int n();
}
class B extends A {
        public int n() {return i;}
}
```

Types of Methods

Abstract – no implementation

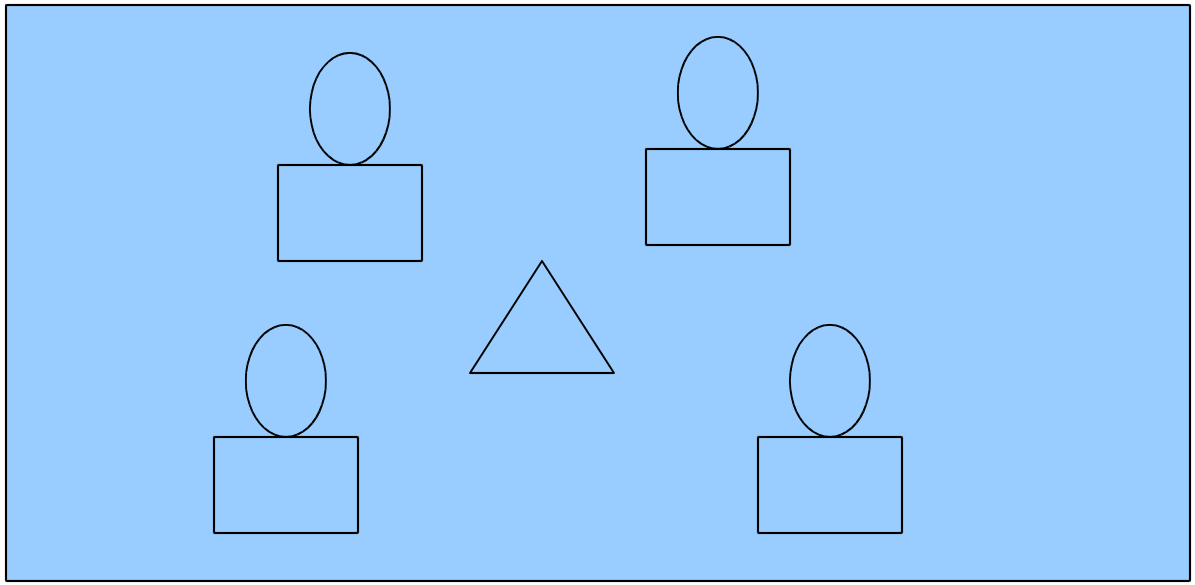
Fully Concrete – full implementation

Partly concrete, i.e. some steps in it are abstract

Overridable

Nonoverridable

An application : A Drawing Tool



Design of the central hierarchy

Think of classes Shape, Circle, Rectangle and Triangle to support following abilities for all shapes:

- Create, move, clone, resize etc

What do you keep in the superclass?

- For use *as it is*
- For specialization and subsequent polymorphism

Benefits of inheritance

- Superclass (Interface) Shape contains most common properties
- It also contains abstract member functions which are applicable to all shapes
- Abstract member functions are concretely defined in subclasses
- Application has a lot of code written in terms of superclass shape