Principles of Computer Science I

Prof. Nadeem Abdul Hamid

CSC 120A - Fall 2004

Lecture Unit 11

Inheritance and Polymorphism

Keeping Track of Pets

```
public class FirstPetProgram {

   public static void main(String[] args) {
      Dog doggy = new Dog("Rufus", 4);
      Cat kitty = new Cat("Panther", 2);

      doggy.makeSound();
      System.out.println(doggy.getName()
                  + " likes " + doggy.getFood());
      kitty.makeSound();
      System.out.println(kitty.getName()
                  + " likes " + kitty.getFood());
   }

}
```

```
>java FirstPetProgram
Rufus is ready to go for a walk!
Panther is ready to catch some mice!
woof woof
Rufus likes doggy bites
meow meow
Panther likes kitty food
```



```
class Dog {
   private String name;
   private int age;
   private int maxAge;
   private String sound;
   private String food;

   public Dog(String n, int a) {
      name = n;
      age = a;
      maxAge = 12;
      sound = "woof woof";
      food = "doggy bites";

      System.out.println(name + " is ready to go for a walk!");
   }

   public String getName() { return name; }
   public int    getAge()  { return age; }
   public int    getMaxAge() { return maxAge; }
   public String getFood() { return food; }

   public void makeSound() {
      System.out.println(sound);
   }
}
```



```
class Cat {
   private String name;
   private int age;
   private int maxAge;
   private String sound;
   private String food;

   public Cat(String n, int a) {
      name = n;
      age = a;
      maxAge = 10;
      sound = "meow meow";
      food = "kitty food";

      System.out.println(name + " is ready to catch some mice!");
   }

   public String getName() { return name; }
   public int    getAge()  { return age; }
   public int    getMaxAge() { return maxAge; }
   public String getFood() { return food; }

   public void makeSound() {
      System.out.println(sound);
   }
}
```


Compare Class Definitions

```
class Dog {
   private String name;
   private int age;
   private int maxAge;
   private String sound;
   private String food;

   public Dog(String n, int a) {
      name = n;
      age = a;
      maxAge = 12;
      sound = "woof woof";
      food = "doggy bites";

      System.out.println(name + " is ready
to go for a walk!");
   }

   public String getName() { return name; }
   public int    getAge()  { return age; }
   public int    getMaxAge() { return
maxAge; }
   public String getFood() { return food; }

   public void makeSound() {
      System.out.println(sound);
   }
}
```

```
class Cat {
   private String name;
   private int age;
   private int maxAge;
   private String sound;
   private String food;

   public Cat(String n, int a) {
      name = n;
      age = a;
      maxAge = 10;
      sound = "meow meow";
      food = "kitty food";

      System.out.println(name + " is ready
to catch some mice!");
   }

   public String getName() { return name; }
   public int    getAge()  { return age; }
   public int    getMaxAge() { return
maxAge; }
   public String getFood() { return food; }

   public void makeSound() {
      System.out.println(sound);
   }
}
```


Pet Classes

- Fields and methods are mostly duplicates
- We factor out common data and responsibilities into a superclass, Pet
- Dog and Cat classes will be subclasses of the Pet class
  - They will automatically inherit its fields and methods…

The Pet Superclass

```
class Pet {

    protected String name;
    protected int age;
    protected int maxAge;
    protected String sound;
    protected String food;

    public String getName() { return name; }
    public int    getAge()  { return age; }
    public int    getMaxAge() { return maxAge; }
    public String getFood() { return food; }

    public void makeSound() {
        System.out.println(sound);
    }

}
```


Subclasses

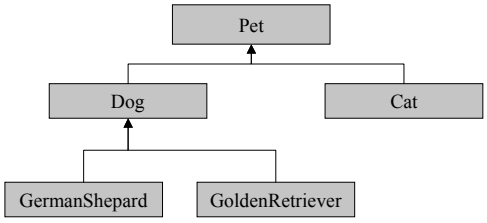
extends clause

```
class Dog extends Pet {
    public Dog(String n, int a) {
        name = n;
        age = a;
        maxAge = 12;
        sound = "woof woof";
        food = "doggy bites";

        System.out.println(name + " is ready to go for a walk!");
    }
}

class Cat extends Pet {
    public Cat(String n, int a) {
        name = n;
        age = a;
        maxAge = 10;
        sound = "meow meow";
        food = "kitty food";

        System.out.println(name + " is ready to catch some mice!");
    }
}
```


Main Program as Before...

```
public class SecondPetProgram {

    public static void main(String[] args) {
        Dog doggy = new Dog("Rufus", 4);
        Cat kitty = new Cat("Panther", 2);

        doggy.makeSound();
        System.out.println(doggy.getName()
                           + " likes " + doggy.getFood());

        kitty.makeSound();
        System.out.println(kitty.getName()
                           + " likes " + kitty.getFood());
    }

}
```


Building a Class Hierarchy



- Arrows indicate "is a ..." relationship

Subclasses and Superclasses

- An object of a subclass can be assigned to a variable of a superclass

```
public class SecondPetProgram {

    public static void main(String[] args) {
        Pet doggy = new Dog("Rufus", 4);
        Pet kitty = new Cat("Panther", 2);

        doggy.makeSound();
        System.out.println(doggy.getName()
                           + " likes " + doggy.getFood());
        kitty.makeSound();
        System.out.println(kitty.getName()
                           + " likes " + kitty.getFood());
    }

}
```


Polymorphism

- Allowing the same object to be used with different types (classes)
  - In Java, a reference to a subclass object can be stored in a superclass variable
- Ability to process objects differently depending on their class
  - Achieved by overriding methods in subclasses
- Useful for
  - Arrays
  - Passing parameters
  - Returning values from a method

Storing Pets in an Array

```
public class SecondPetProgram {

   public static void main(String[] args) {
      Pet[] pets = new Pet[2];

      pets[0] = new Dog("Rufus", 4);
      pets[1] = new Cat("Panther", 2);

      for (int i = 0; i < pets.length; i++) {
         pets[i].makeSound();
         System.out.println(pets[i].getName()
                            + " likes " + pets[i].getFood());
      }
   }

}
```

```
>java SecondPetProgram
Rufus is ready to go for a walk!
Panther is ready to catch some mice!
woof woof
Rufus likes doggy bites
meow meow
Panther likes kitty food
```


Subclasses Can Define New Fields and Methods

```
class Cat extends Pet {

   protected boolean goodMouser;

   public Cat(String n, int a, boolean mouser) {
      name = n;
      age = a;
      goodMouser = mouser;
      maxAge = 10;
      sound = "meow meow";
      food = "kitty food";

      if (goodMouser)
         System.out.println(name + " is ready to catch some mice!");
      else
         System.out.println(name + " is ready to unroll some yarn!");
   }

   public boolean catchesMice() { return goodMouser; }

}
```



```
class Dog extends Pet {

   protected boolean paperFetcher;

   public Dog(String n, int a, boolean getsPaper) {
      name = n;
      age = a;
      paperFetcher = getsPaper;
      maxAge = 12;
      sound = "woof woof";
      food = "doggy bites";

      System.out.println(name + " is ready to go for a walk!");
   }

   public boolean fetchesPaper() { return paperFetcher; }

}
```



```
public class ThirdPetProgram {

   public static void main(String[] args) {

      Dog doggy = new Dog("Rufus", 4, true);
      Cat kitty = new Cat("Panther", 2, false);

      doggy.makeSound();
      System.out.print(doggy.getName()
                       + " likes " + doggy.getFood());
      if (doggy.fetchesPaper())
         System.out.println(" and fetches the paper");
      else
         System.out.println(" but can't fetch a paper");

      kitty.makeSound();
      System.out.print(kitty.getName()
                       + " likes " + kitty.getFood());
      if (kitty.catchesMice())
         System.out.println(" and will eliminate pesky rodents");
      else
         System.out.println(" but can't stand mice");
   }
}
```

can't be Pet


```
   public static void main(String[] args) {
      Pet[] pets = new Pet[2];
      pets[0] = new Dog("Rufus", 4, true);
      pets[1] = new Cat("Panther", 2, false);

      if (pets[0].fetchesPaper())
         System.out.println(pets[0].getName() + " fetches the paper");

   }
```

ThirdPetProgram.java:9: cannot resolve symbol
symbol : method fetchesPaper ()
location: class Pet
if (pets[0].fetchesPaper())

Casting

```
public static void main(String[] args) {
      Pet[] pets = new Pet[2];
      pets[0] = new Dog("Rufus", 4, true);
      pets[1] = new Cat("Panther", 2, false);

      if ( ((Dog)pets[0]).fetchesPaper() )
         System.out.println(pets[0].getName() + " fetches the paper");

   }
```

Passing Parameters

```
class Vet {
   public void vaccinate(Pet p) {
      System.out.println("Vaccinating " + p.getName() + "...");
      p.makeSound(true);
   }
}

public class ThirdPetProgram {
   public static void main(String[] args) {
      Pet[] pets = new Pet[2];
      pets[0] = new Dog("Rufus", 4, true);
      pets[1] = new Cat("Panther", 2, false);

      Vet doc = new Vet();

      for (int i = 0; i < pets.length; i++) {
         doc.vaccinate(pets[i]);
      }
   }
}
```

```
>java ThirdPetProgram
Rufus is ready to go for a walk!
Panther is ready to unroll some yarn!
Vaccinating Rufus...
WOOF WOOF
Vaccinating Panther...
MEOW MEOW
```


Overloading Methods

- Defining methods with the same name but different signature (number and types of parameters)

```
class Pet {
   ...

   public void makeSound() {
      System.out.println(sound);
   }

   public void makeSound(boolean loud) {
      if (loud) System.out.println(sound.toUpperCase());
      else System.out.println(sound);
   }
}
```


Overloaded Methods

- Can make calls to other versions of the same method…

```
class Pet {
   ...

   public void makeSound() {
      makeSound(false);      // default behavior
   }

   public void makeSound(boolean loud) {
      if (loud) System.out.println(sound.toUpperCase());
      else System.out.println(sound);
   }
}
```


Overloaded Constructors

```
class Dog extends Pet {
   protected boolean paperFetcher;

   public Dog(String n, int a) {
      name = n;
      age = a;
      paperFetcher = true;            // default
      maxAge = 12;
      sound = "woof woof";
      food = "doggy bites";
      System.out.println(name + " is ready to go for a walk!");
   }

   public Dog(String n, int a, boolean getsPaper) {
      name = n;
      age = a;
      paperFetcher = getsPaper;
      maxAge = 12;
      sound = "woof woof";
      food = "doggy bites";
      System.out.println(name + " is ready to go for a walk!");
   }

   public boolean fetchesPaper() { return paperFetcher; }
}
```


Calling Other Constructors

```
public Dog(String n, int a) {
   this(n, a, true);   // call the full constructor with default
                       // value for paperFetcher initializer
}

public Dog(String n, int a, boolean getsPaper) {
   name = n;
   age = a;
   paperFetcher = getsPaper;
   maxAge = 12;
   sound = "woof woof";
   food = "doggy bites";

   System.out.println(name + " is ready to go for a walk!");
}
```

special Java keyword

Object-Oriented Concepts to Know

- Overloading
- Inheritance
  - Overriding
  - Hiding
  - Polymorphism
- Scope and access rules
- Copy Constructors

Overloading

- Reuse of a method name with a different signature (list of parameters)
- Can apply to constructors as well as regular methods
- Special keyword "this" can be used to invoke another version of a constructor in the same class
- Other overloaded versions of regular methods can be accessed directly by name
- Overloading is not related to inheritance or overriding, although the features often interact in subtle ways

```
class A {
   protected int x;
   protected int y;

   /* overloaded constructors */
   public A() { this(0, 0); }

   public A(int newx, int newy) {
      x = newx;
      y = newy;
   }

   /* overloaded methods */
   public int foo(int z) {
   /* return foo(z, z); */
      x += z;
      y += z;
      return x+y;
   }

   public int foo(int z1, int z2) {
      x += z1;
      y += z2;
      return x+y;
   }
}
```


Inheritance

- Acquiring a field or method from a superclass
  - superclass: B
  - subclass: C
  - A subclass is also called a "derived" class
- Use "super" (Java keyword) to invoke the superclass' constructor
  - If you don't explicitly invoke it, Java tries by default to invoke super()
- Subclass has access to superclass' fields and methods
  - except if they are "private"
- References to subclasses can be stored in variables of the superclass type

```
class B {
   protected int x;

   public B() { this(0); }
   public B(int newx) {
      x = newx;
   }

   public int foo(double z) {
      return (x = (int)(x * z));
   }
}

class C extends B {
   protected int y;

   public C() { this(5,0); }

   public C(int newx, int newy) {
      super(newx);
      y = newy;
   }

   public int bar(double z) {
      return (y = y + foo(z));
   }
}
```


Overriding

- When a subclass redefines a method in the superclass
  - by providing a method with exactly the same header as the superclass
- If an object reference of class D is being stored in a variable of class B, the foo method used will be the overridden version

```
B b = new D();
System.out.println(b.foo(1.0));
```

  - Prints out 1 (not 0)
- Can refer to the overridden method in the superclass using "super.methodName(…)"

```
class B {
   protected int x;

   public B() { this(0); }
   public B(int newx) {
      x = newx;
   }

   public int foo(double z) {
      return (x = (int)(x * z));
   }
}

class D extends B {
   public int foo(double z) {
      return (x = (int)(x + z));
   }
}
```


Hiding

- Same concept as overriding, but applies to fields instead of methods
  - Redefining a field with the same name in a subclass "hides" the field in the superclass

```
E e = new E();
System.out.println(e.getX());
System.out.println(e.getDoubleX());
```

- Can use "super" to access hidden fields in the superclass

```
class B {
   protected int x;

   public B() { this(0); }
   public B(int newx) {
      x = newx;
   }

   public int foo(double z) {
      return (x = (int)(x * z));
   }
}

class E extends B {
   protected double x;

   public E() { x = 6.0; }
   public int getX() { return super.x; }
   public double getDoubleX() { return x; }
}
```


Polymorphism

- In general, allowing the same object to be used with different types
- In Java, especially refers to the ability to process objects differently depending on their (actual) class
  - Achieved by overriding methods in subclasses

Polymorphism Example

```
Shape[] shapes = new Shape[5];
shapes[0] = new Shape();
shapes[1] = new Square(5.0);
shapes[2] = new Circle(3.0);
shapes[3] = new Square(6.0);
shapes[4] = new Shape();

for (int i = 0; i < shapes.length; i++) {
   System.out.println(shapes[i]);
   System.out.println("   Area: " + shapes[i].area());
   System.out.println("   Perim: " + shapes[i].perimeter());
}
```

```
Shape@192d342
   Area: 0.0
   Perim: 0.0
Square [side=5.0]
   Area: 25.0
   Perim: 20.0
Circle [radius=3.0]
   Area: 28.259999999999998
   Perim: 18.84
Square [side=6.0]
   Area: 36.0
   Perim: 24.0
Shape@6b97fd
   Area: 0.0
   Perim: 0.0
```

```
class Shape {
   public double area() { return 0.0; }

   public double perimeter() { return 0.0; }
}

class Square extends Shape {
   double side;

   public Square(double side) { this.side = side; }

   public double area() { return side*side; }

   public double perimeter() { return 4*side; }

   public String toString() {
      return "Square [side=" + side + "]";
   }
}

class Circle extends Shape {
   public final static double PI = 3.14;

   double radius;

   public Circle(double radius) { this.radius = radius; }

   public double area() { return PI*radius*radius; }

   public double perimeter() { return 2*PI*radius; }

   public String toString() {
      return "Circle [radius=" + radius + "]";
   }
}
```

Instanceof Operator

- Determine the run-time type (class) of an object variable or expression

```
Shapes shapes[] = new Shapes[10];
...
if (shapes[0] instanceof Circle) {
    System.out.println("shapes[0] is a Circle");
}
```