Polymorphism

Polymorphism

- Polymorphism is a Greek word meaning “many forms”.
- Polymorphism enables us to “program in the general” rather than “program in the specific.”

Light-morph jaguar

Dark-morph jaguar

- Example: Suppose we create a program that simulates the movement of several types of animals for a biological study.
  - Classes Fish, Frog and Bird represent the three types of animals under investigation.
  - Each class **extends superclass Animal**, which contains a **method move** and **maintains an animal’s current location** as x-y coordinates.
  - To simulate the animals’ movements, the program sends each object the same message once per second.

Polymorphism(cont'd)

- However, each specific type of Animal responds to a move message in a unique way
  - a Fish might swim three feet, a Frog might jump five feet and a Bird might fly ten feet.
- The program issues **the same** message (i.e., **move**) to each animal object generically, **but each object knows how to modify its x-y coordinates appropriately** for its specific type of movement.
- Relying on each object to know how to "do the right thing" in response to the same method call is the key concept of polymorphism.
- The same message (in this case, **move**) sent to a variety of objects has "**many forms**" of results — hence the term polymorphism.

- Late binding(dynamic binding): the process of associating a method definition with a method invocation at run-time
- Static binding(early binding) bind at compilation time

Polymorphism(cont'd)

- *Polymorphism* and *late binding* are essentially just different words for the same concept
- Polymorphism refers to the processes of assigning multiple meanings to the same method name using late binding
- With polymorphism, we can design and implement systems that are easily extensible
  - New classes can be added with little or no modification to the general portions of the program, as long as the new classes are part of the inheritance hierarchy that the program processes generically.
  - The only parts of a program that must be altered to accommodate new classes are those that require direct knowledge of the new classes that we add to the hierarchy.

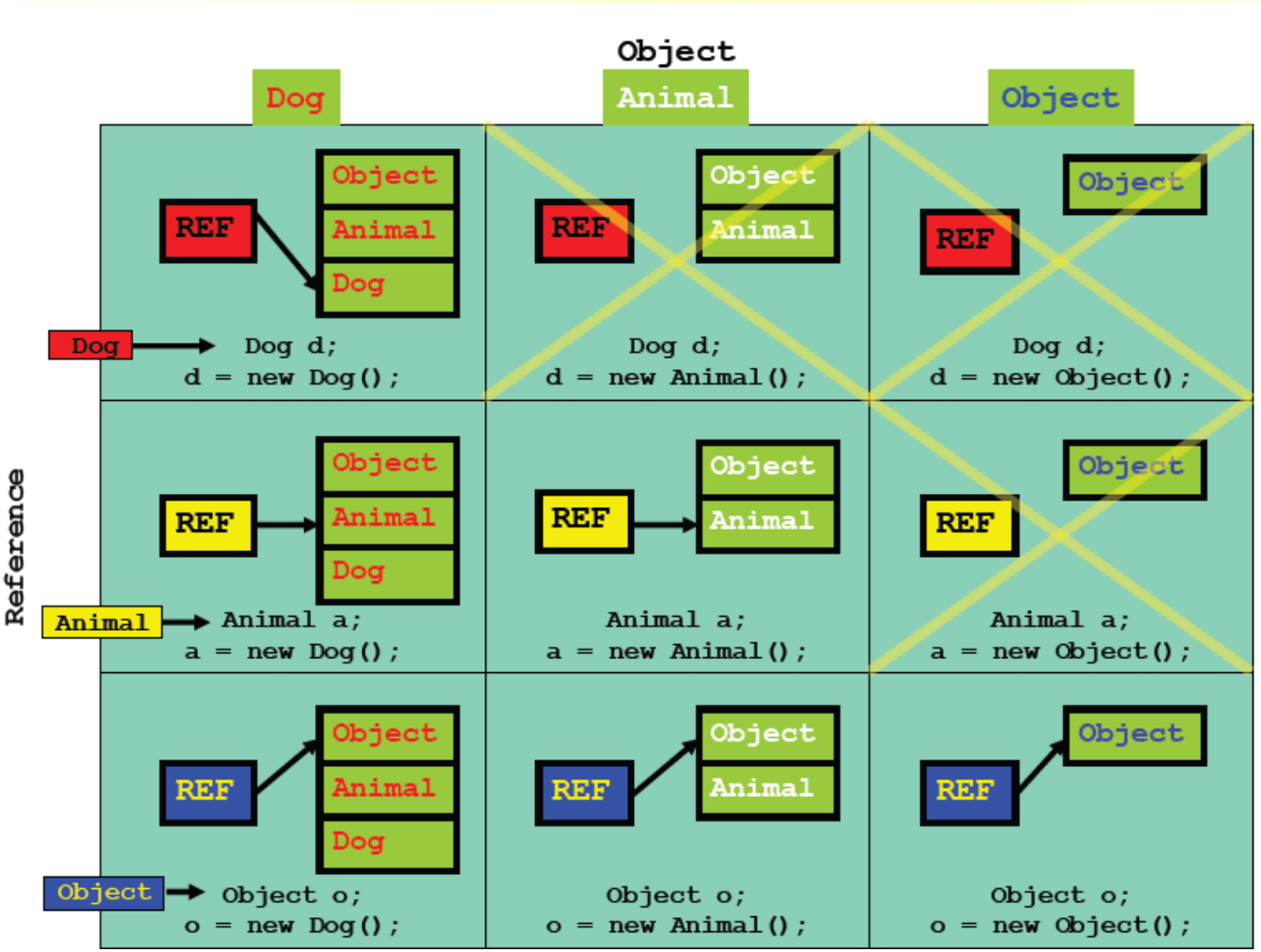
Polymorphism Example

- Example: Quadrilaterals
  - If Rectangle is derived from Quadrilateral, then a Rectangle object is a more specific version of a Quadrilateral.
  - Any operation that can be performed on a Quadrilateral can also be performed on a Rectangle.
  - These operations can also be performed on other Quadrilaterals, such as Squares, Parallelograms and Trapezoids.
  - Polymorphism occurs when a program invokes a method through a superclass Quadrilateral variable—at execution time, the correct subclass version of the method is called, based on the type of the reference stored in the superclass variable.

Polymorphic Behavior

- All Java objects are polymorphic since any object will pass the IS-A test for at least their own type and the class Object
  - A bird is an instance of Bird class, also an instance of Animal and Object
- We are able to assign an object of a sub-class into an object of a super-class as in:
  - Animal MyAnimal = new Dog();
- But the reverse is not true. We can't assign a superclass object into a sub-class object.
  - Dog MyDog = new Animal(); // illegal
- *All dogs are animals but not all animals are dogs*

Cont'd

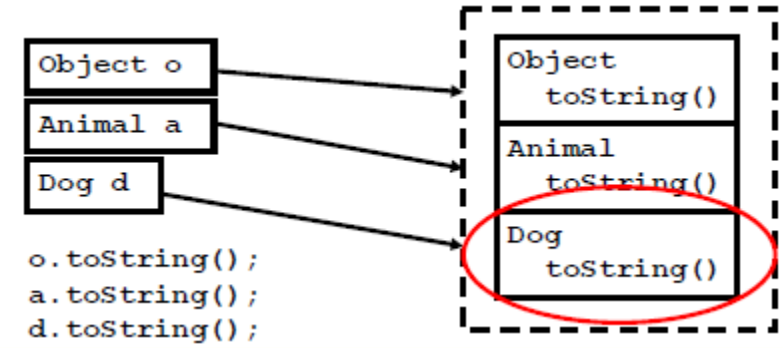

Object
Dog
Animal
Object
Reference
REF
Object
Animal
Dog
Dog
Dog d;
d = new Dog();
Dog d;
d = new Animal();
Dog d;
d = new Object();
Animal
Animal a;
a = new Dog();
Animal a;
a = new Animal();
Animal a;
a = new Object();
Object
Object o;
o = new Dog();
Object o;
o = new Animal();
Object o;
o = new Object();

Method Calls and Polymorphism

- When a superclass variable contains a reference to a subclass object, and that reference is used to call a method, the subclass version of the method is called.
- Assume the Dog class extends the Animal class, redefining the “makeNoise” method.
- Consider the following:
  - Animal myAnimal = new Dog();
  - myAnimal.makeNoise();
- Note: The Animal reference is referring to a Dog object. And it is the Dog’s makeNoise method that gets invoked!

Cont'd

- No matter what the reference type is, Java will search the object and execute the lowest occurrence of a method it finds.
- class Object has a toString method
- Assume that both Animal and Dog have overridden the toString method



Downcasting and Upcasting

- **Upcasting:** assigning an object of a derived class to a variable of a base class (or any ancestor class).
  - E.g. Object o = new Student();
- **Downcast:** cast from a base case to a derived class (or from any ancestor class to any descendent class).
  - used to restore an object back to its original class (back to what it was “born as” with the new operator)
  - E.g. Student b = (Student)o;

Example

- The following two output statements will produce different results, depending on whether p is a Dog or a Cat:

```
Pet p;
p = new Dog( );
System.out.println(p.speak( ));
p = new Cat( );
System.out.println(p.speak( ));
```

- Here the speak method is called a polymorphic method.

Example(cont'd)

```
class Pet {
     private String name;
     public String getName( ) {
      return name;
    }
    public void setName(String petName) {
         name = petName;
   }
   public String speak( ) {
          return "I'm your cuddly little pet.";
   }
}
 class Cat extends Pet {
    public String speak( ) {
       return "Don't give me orders.\n" + "I speak only when I want to.";
    }
}
```

Example(cont'd)

```
class Dog extends Pet {
    public String fetch( ) {
        return "Yes, master. Fetch I will.";
    }
}
==============================
public class RunPetDog{
   public static void main(String args[]){
      Pet p;
      p = new Dog( );
      System.out.println(p.speak( ));
      p = new Cat( );
     System.out.println(p.speak( ));

   }
}
```

Example(cont'd)

- Output

  I'm Your cuddly little pet

  Don't give me orders

  I speak only when I want

Activity 1

- Assuming SavingsAccount is a subclass of BankAccount, which of the following code fragments are valid in Java?

a. BankAccount account = new SavingsAccount();
b. SavingsAccount account2 = new BankAccount();
c. BankAccount account = null;
d. SavingsAccount account2 = account;

- Answer: a only.

Activity 2

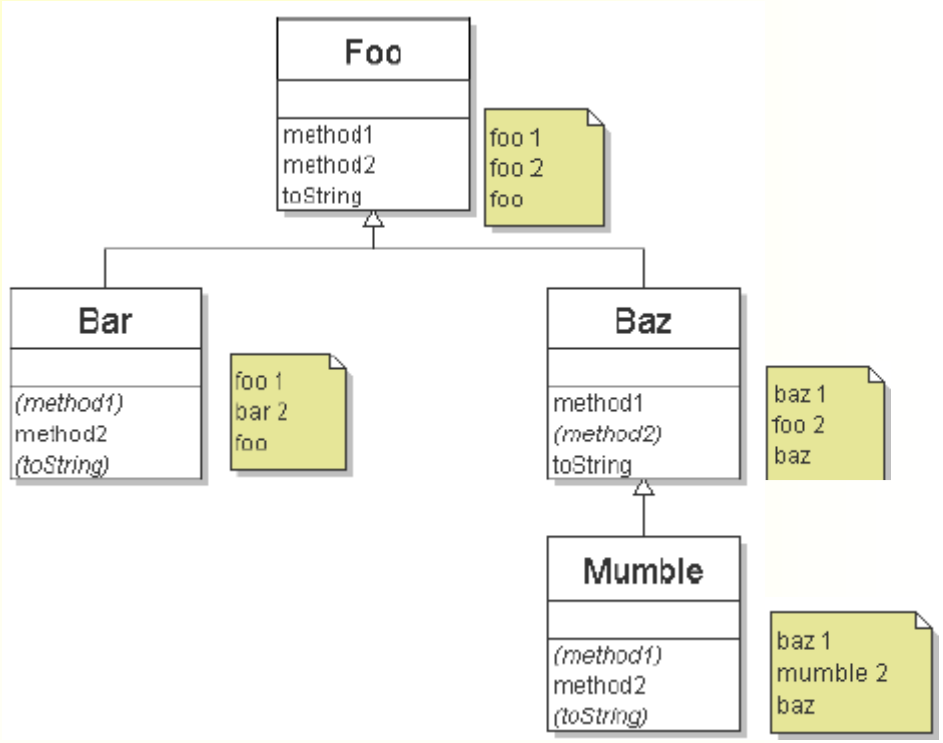
- If account is a variable of type BankAccount that holds a non-null reference, what do you know about the object to which account refers?
- **Answer:** It belongs to the class BankAccount or one of its subclasses.

Activity 3

```
public class Foo {
   public void method1() {
      System.out.println("foo 1");
   }
   public void method2() {
      System.out.println("foo 2");
   }
   public String toString() {
      return "foo";
   }
}
```

Activity 3(cont'd)

```
class Bar extends Foo {
    public void method2() {
        System.out.println("bar 2");
    }
}
class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }
    public String toString() {
        return "baz";
    }
}
class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble 2");
    }
}
```



Activity 4

What is output by the code to the right when run?

A. `!!live`

B. `!eggegg`

C. `!egglive`

D. `!!!`

E. `eggegglive`

```
public class Animal{
  public String bt(){ return "!"; }
}

public class Mammal extends Animal{
  public String bt(){ return "live"; }
}

public class Platypus extends Mammal{
  public String bt(){ return "egg";}
}

Animal a1 = new Animal();
Animal a2 = new Platypus();
Mammal m1 = new Platypus();
System.out.print( a1.bt() );
System.out.print( a2.bt() );
System.out.print( m1.bt() );
```

Polymorphism vs. Inheritance

- Inheritance is required in order to achieve polymorphism (we must have class hierarchies).
  - Re-using class definitions via extension and redefinition
- Polymorphism is not required in order to achieve inheritance.
  - An object of class A acts as an object of class B (an ancestor to A)