

# Introduction to Programming

*with Java, for Beginners*

Method Overloading and Overriding  
Inheritance with Abstract Classes  
Interfaces

## Polymorphism

- Polymorphism means *many* (poly) *shapes* (morph)
- In Java, **polymorphism** refers to the fact that you can have multiple methods with the same name in the same class
- There are two kinds of polymorphism:
  - **Overloading**
    - > Two or more methods with *different signatures*
  - **Overriding**
    - > A method in a subclass to “override” a method in the superclass that has the *same signature*
- We’ve already seen Overloading scenario with Constructors  
E.g. `public PersonDB() {..}`  
`public PersonDB(Person [] ) {..}`

ESE112

1/17

## Method Overloading

Method *overloading* occurs when

- A class has two or more methods with the same *name* but different *signatures*
  - > i.e. the number, order, or types of their parameters differ

```
// the foo method is overloaded
public void foo() { }
public void foo(int x) { }
public void foo(int x, double y) { }
```

- When the `foo(..)` method is called, Java picks the one that “matches”. E.g.

```
foo(10, 350.5);
```

ESE112

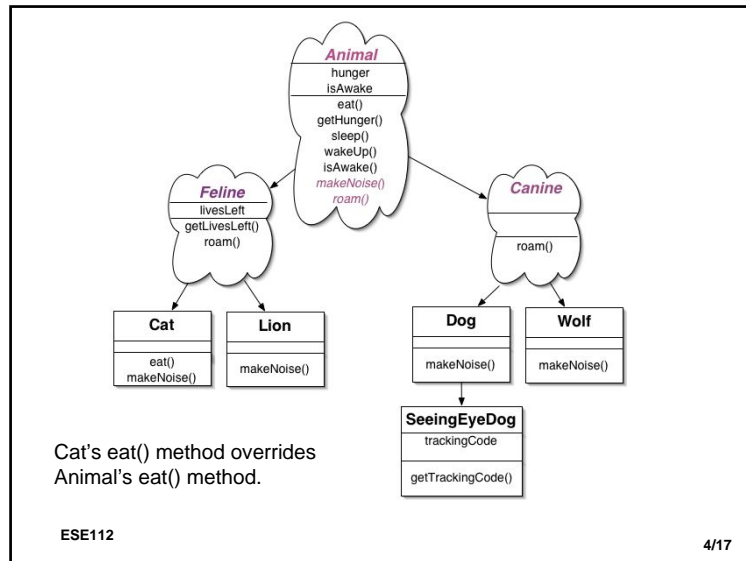
2/17

## Overriding

- **Overriding** occurs if
  - There are two or more methods with the same name *and the same signature* in an inheritance chain
  - For example, the Object class has a `toString()` method
    - > It can be *overridden* in a subclass simply by creating a method with the same signature  
`public String toString() { . . }`
  - Java picks the “lowest” method in the inheritance chain possible

ESE112

3/17



## Abstract Classes

- An abstract class is a class
  - With keyword *abstract* placed before declaring a class
    - E.g. `public abstract class Animal { .. }`
  - It cannot be instantiated
    - Illegal: `Animal a = new Animal ()`
  - It may have “abstract methods” i.e. methods with keyword `abstract`
    - Abstract methods are body-less i.e. no code within them
  - They also can have regular/concrete methods
    - Methods with code in them

ESE112 5/17

## Setting up Inheritance with an Abstract Class

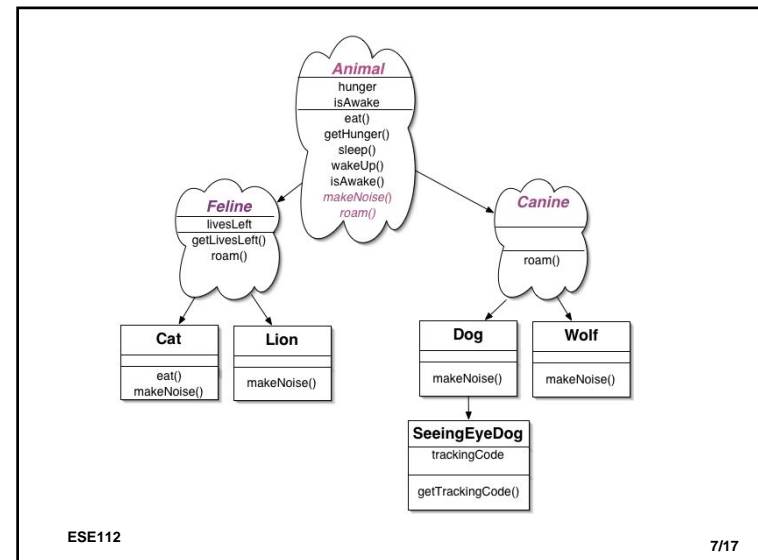
```

public abstract class Animal{
    private double hunger;
    private boolean isAwake;

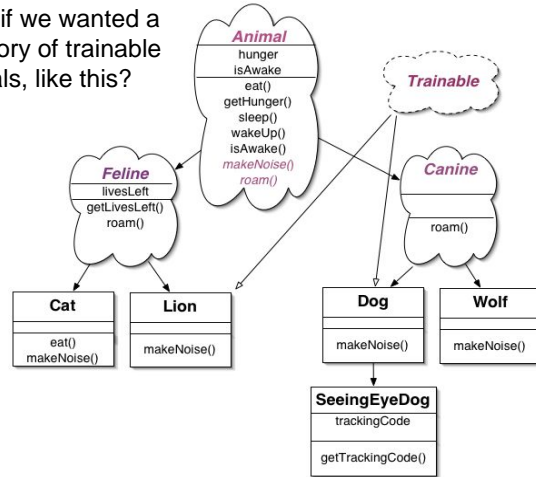
    public void eat(){
        hunger = 0;
    }
    public abstract String makeNoise();
}

/* Dog class */
public class Dog extends Animal {
    // The Dog class must have a concrete makeNoise method.
    // Otherwise, it won't compile.
    public String makeNoise(){
        return "woof!";
    }
}
  
```

ESE112 6/17



What if we wanted a category of trainable animals, like this?



ESE112

8/17

## Creating an Interface

```
public interface Trainable{
    public void sit();
    public void stand();
}
```

- An interface is like a class except:
  - The keyword **interface** is used instead of **class**
  - All of its methods are body-less
  - It has no instance variables
  - It can not be instantiated (Illegal: new Trainable())
- An interface is like a contract, protocol, role, or point of view
- Code written for an interface type works with *any object* whose class *implements* it
  - It can assume that all of the subtypes have the methods listed in the interface (e.g. sit and stand)

ESE112

9/17

## A Class implements an Interface

```
public class Dog implements Trainable{
    public void sit(){ // code for sit method }

    public void stand() (){ // code for stand method }
}
```

A class that implements an interface

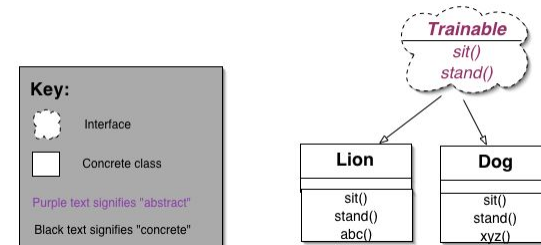
- Must provide concrete methods for each interface method
- May have additional methods
- May implement multiple interfaces

```
public class Dog implements Trainable, Comparable{
    // code
}
```

ESE112

10/17

## Example: Trainable Interface



```
> Lion lion = new Lion();
> Dog dog = new Dog();
> Trainable beast;
> beast = lion; // static type is Trainable; dynamic type is Lion
> beast.sit();
> beast = dog;
> beast.sit(); // static type is Trainable; dynamic type is Dog
```

ESE112

11/17

## Examples of Interfaces

- Interfaces are typically “lightweight”, with only a few method headers

```
public interface Displayable{
    public void display();
}

public interface XYZProtocol {
    public void connect();
    public void disconnect();
    public Object receive();
}
```

ESE112

12/17

## Summary

	Regular Class	Abstract Class	Interface
Methods	all concrete	concrete and/or abstract	all abstract
May have instance variables	yes	yes	no*
May be instantiated	yes	no	no

\* Except they may have constants

ESE112

13/17

## instanceof Operator with Interfaces

- Usage: *Ivalue instanceof T*
- The expression is true if *Ivalue* has type T or is a subtype of T.
- Example:

```
public class Rectangle implements Displayable { .. }
public class Square extends Rectangle { .. }
```

```
> Rectangle r = new Rectangle(2,3);
> r instanceof Rectangle
true
> r instanceof Displayable
true
> r instanceof Square
false
> r = new Square(10);
> r instanceof Square
true

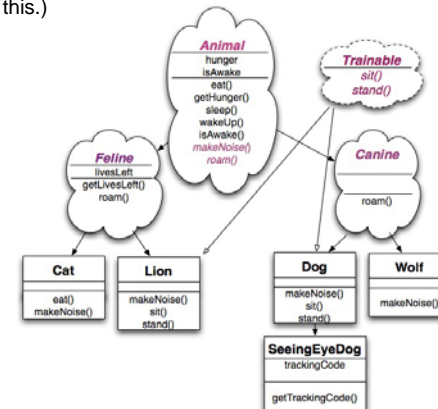
> Square s = new Square(8);
> s instanceof Square
true
> s instanceof Rectangle
true
> s instanceof Displayable
true
```

ESE112

14/17

## A Design Problem

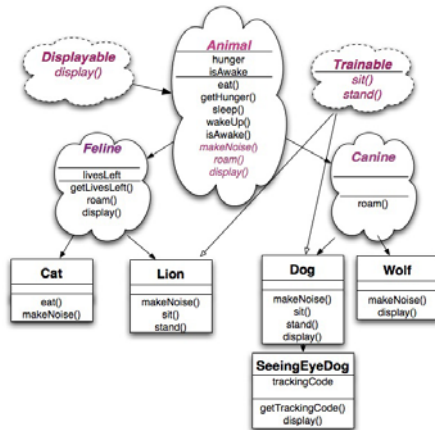
- What if we want to write graphics code for all the animals, and want to require that they all have a display(..) method? (There are many ways to accomplish this.)



ESE112

15/17

## One of Many Solutions



ESE112

16/17

## Syntax for Inheriting a class & Implementing a class

```

public class SubclassName extends SuperclassName
    implements Interface1, Interface2 {

}
  
```

ESE112

17/17