

CISC 3120

# C09: Interface and Abstract Class and Method

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Outline

- Recap
  - Polymorphism
  - In-class group exercise
- Abstract method
- Abstract class
- Interfaces
- The Object super class
- The instanceof operator

# In-Class Group Exercise

- Shape, Circle, and Rectangle
- Discuss your solution with your team members

# The Shape Class

- Do you like the area() method here?

```
public class Shape {
```

```
    ...
```

```
    public double area() {
```

```
        System.out.println("This method is not supposed to be called.");
```

```
        return 0;
```

```
    }
```

```
    ...
```

```
}
```

- Are we ever going to instantiate a Shape object?

# Abstract Class

- A class that is declared abstract
- Example

```
abstract class Animal {  
  
    ...  
  
}
```

- Abstract classes cannot be instantiated, but they can be subclassed.

# Subclass & Instantiation

- Abstract classes *cannot* be instantiated, but they *can be* subclassed.

```
abstract class Animal {  
    ...  
}
```

- How about these examples?

```
Animal animal = new Animal();
```


```
class Dog extends Animal {...}
```

# Subclass & Instantiation

- Abstract classes *cannot* be instantiated, but they *can be* subclassed.

```
abstract class Animal {  
...  
}
```

```
Animal animal = new Animal();
```



```
class Dog extends Animal {...}
```



# Abstract Method

- A method that is declared without an implementation

```
abstract void makeNoise();
```

- A class that has an abstract method must be declared abstract
  - How about these examples?

```
class Animal {  
    abstract void makeNoise();  
}
```

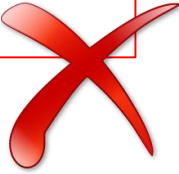
```
abstract class Animal {  
    abstract void makeNoise();  
}
```




# Class with Abstract Method

- A class that has an abstract method must be declared abstract

```
class Animal {  
    abstract void makeNoise();  
}
```



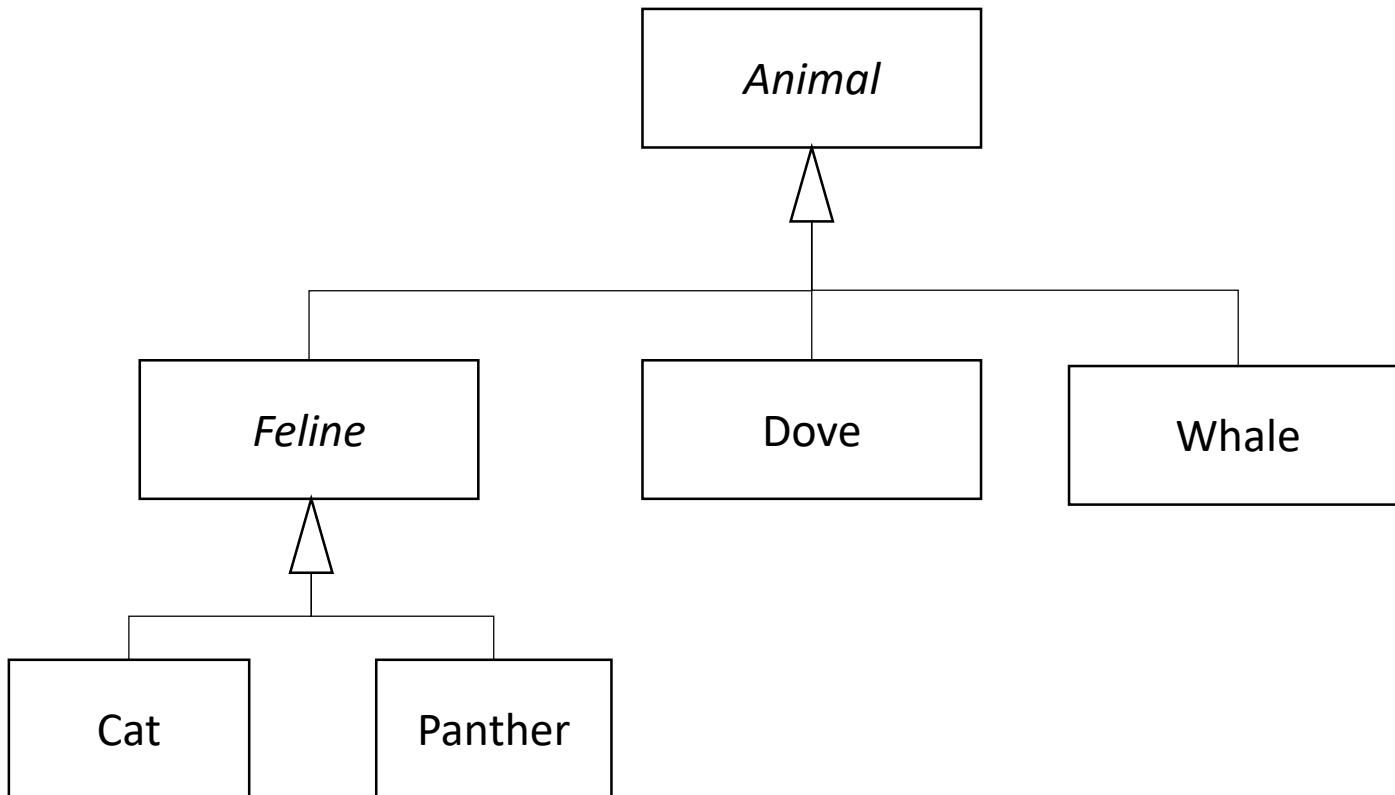
```
abstract class Animal {  
    abstract void makeNoise();  
}
```



# Subclass an Abstract Class

- Concrete subclass
  - A subclass may provide implementations for all of the abstract methods in its parent class.
- Abstract subclass
  - The subclass must also be declared abstract if it does not provide implementation of all of the abstract methods in its parent class.

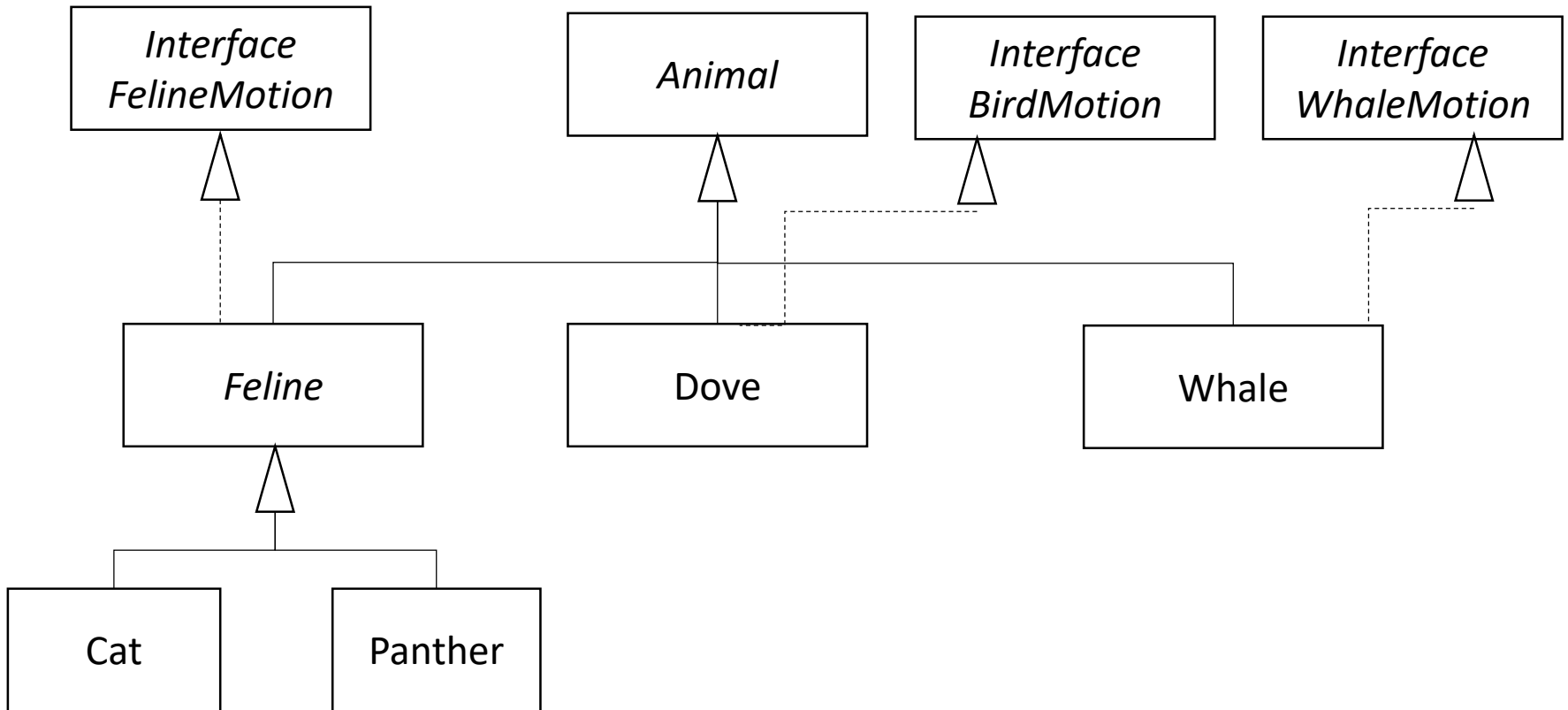
# Example: Animal Game



# Interfaces

- Not the "interface" in "Graphical User Interface"
- Java has a reference type, called interface
  - Typically contain abstract methods only.
    - Java 8 introduces the concept of default methods and permits static methods
  - Interfaces cannot be instantiated
    - can only be implemented by classes or extended by other interfaces

# Example: Animal Game



# Example: Birds Fly, Whales Swim, and Cats ...

```
public interface BirdMotion {  
    public void fly(Direction direction, double speed, double distance);  
}
```

```
public interface WhaleMotion {  
    public void swim(Direction direction, double speed, double distance);  
}
```

```
public interface FelineMotion {  
    public void walk(Direction direction, double speed, double distance);  
    public void pounce(Animal prey);  
}
```

# Example: Implementing Interfaces

```
abstract class Feline implements FelineMotion {
```

```
...
```

```
    public void walk(Direction direction, double speed, double distance) { ... }
```

```
    public void pounce(Animal prey) { ... }
```

```
...
```

```
}
```

```
class Dove extends Animal implements BirdMotion { ...
```

```
    public void fly(Direction direction, double speed, double distance) { ... }
```

```
}
```

# Using an Interface as a Type

- Interfaces are data types

```
void flyAll(ArrayList<BirdMotion> flyingAnimals) {  
    ...  
}
```

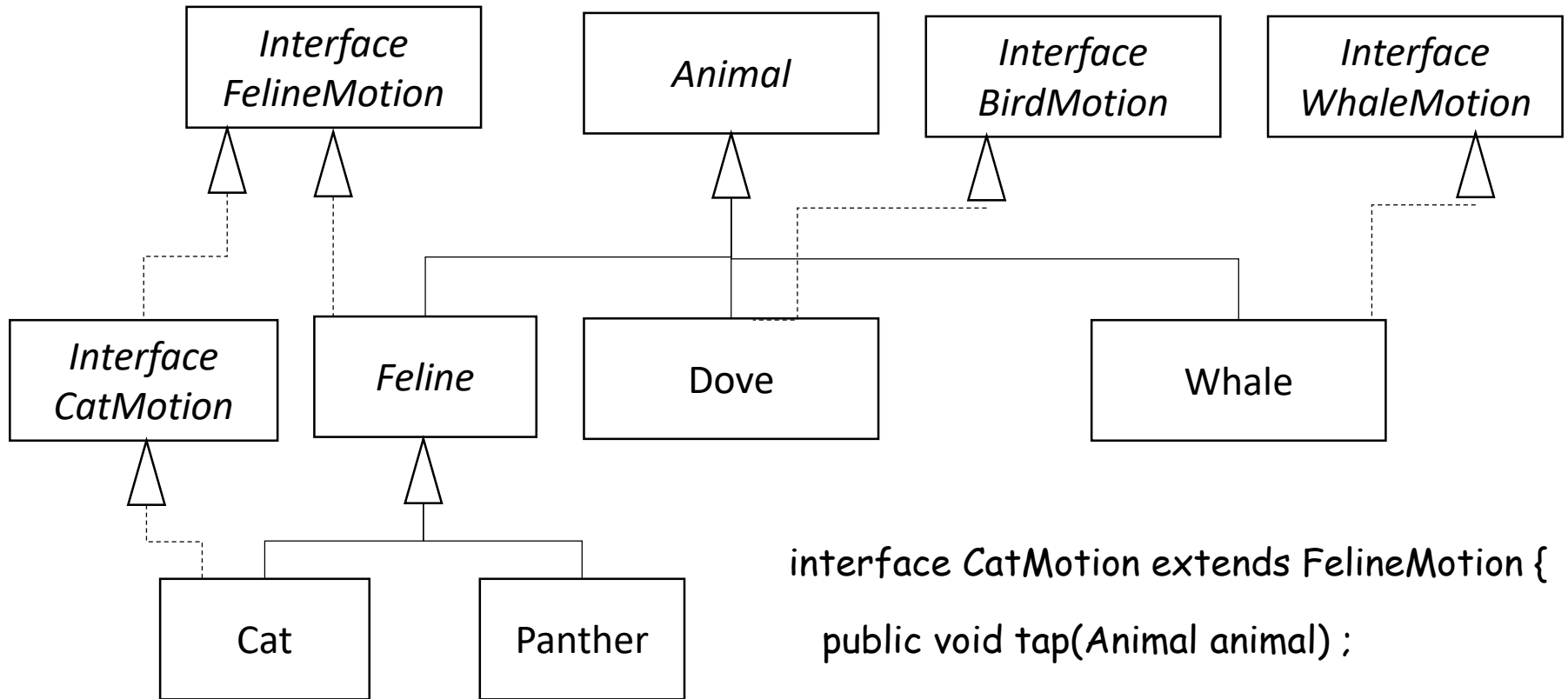


# Evolving Interfaces

- Interfaces can be extended (like classes)

```
interface CatMotion extends FelineMotion {  
    public void tap(Animal animal) ;  
}
```

# Example: Extending FelineMotion



```
interface CatMotion extends FelineMotion {
    public void tap(Animal animal) ;
}
```

# Implementing Multiple Interfaces

- A class can implement multiple interfaces
- But a class cannot extend multiple classes
- Which one of the following are is allowed in Java?

```
class FlyingCat extends Cat, Dove  
{  
...  
}
```

```
class FlyingCat implements  
BirdMotion, CatMotion {  
...  
}
```

# Implementing Multiple Interfaces

- A class can implement multiple interfaces
- But a class cannot extend multiple classes

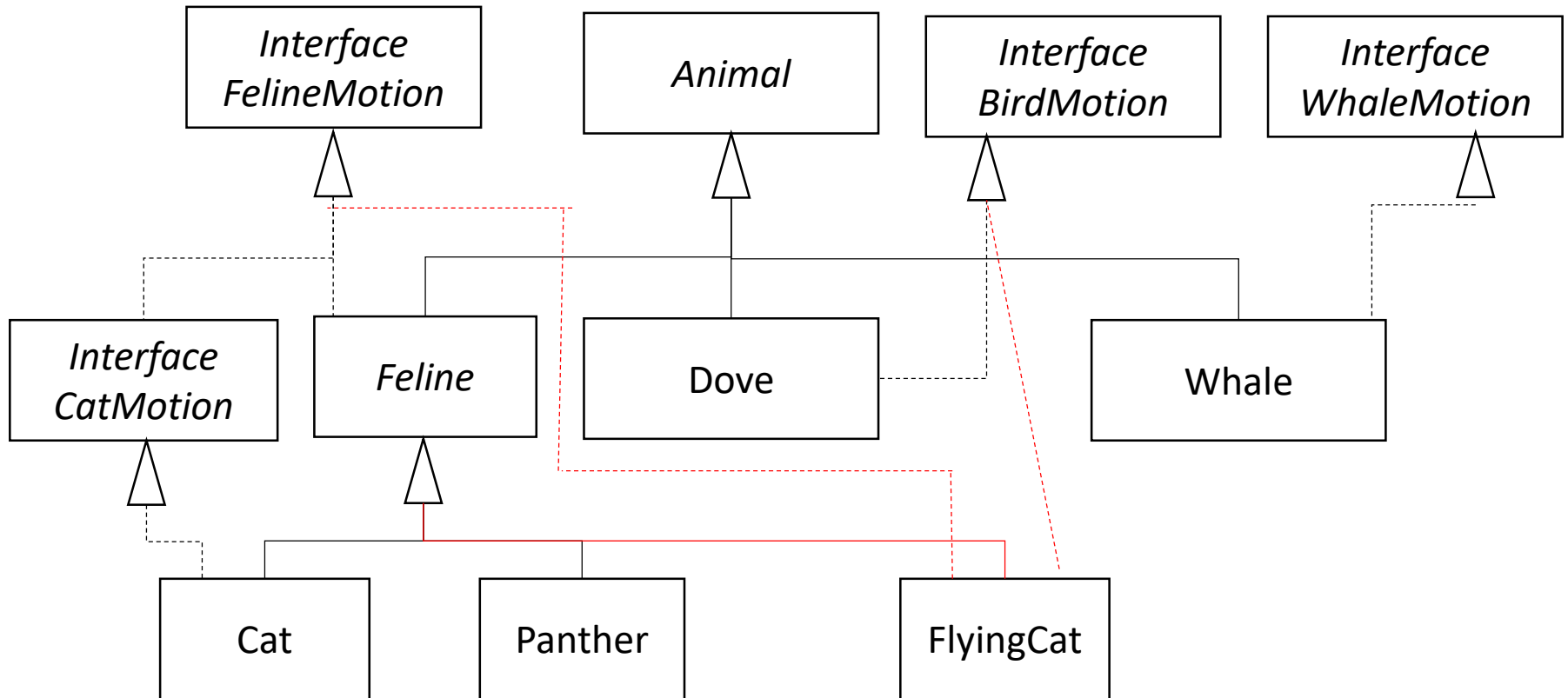
```
class FlyingCat extends Cat, Dove  
{  
...  
}
```



```
class FlyingCat implements  
BirdMotion, CatMotion {  
...  
}
```



# Example: Flying Cat in the Magic Kindom



# What an object can do?

- Java may know what method an object can invoke only at runtime.
- As a programmer how do we cope?

- Use appropriate data types

```
void flyAll(ArrayList<BirdMotion> flyingAnimals) {  
    ...  
}
```

- Check object type at runtime (using instanceof)

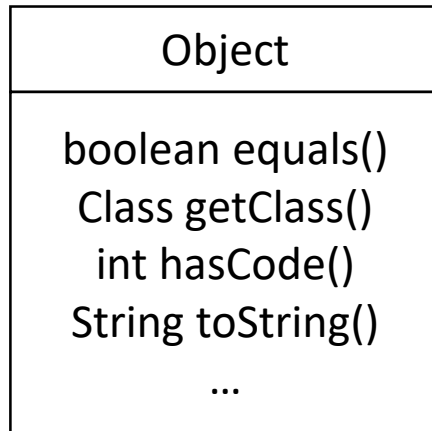
# Operator instanceof

- Evaluates to true if the object is a given type; false otherwise

```
public static void move(Animal animal) {  
    if (animal instanceof Cat) {  
        ...  
    }  
}
```

# The Object Super Class

- Java has a class called **Object**, like
- **All classes are subclass of Object in Java**





# Questions

- Abstract class
- Abstract method
- Interfaces
- Extending abstract classes
- Implementing interfaces
- The instanceof operator
- The Object superclass

# Assignment

- To be available via Blackboard
- Project 2