

COMP 110/L Lecture 20

Maryam Jalali

Slides adapted from Dr. Kyle Dewey

Outline

- `super` in methods
- `abstract` **Classes** and **Methods**
- **Polymorphism**

super in Methods

Recap

You've seen `super` in constructors...

Recap

You've seen `super` in constructors...

```
public class Base {  
    public Base(int x) { ... }  
}
```

Recap

You've seen `super` in constructors...

```
public class Base {  
    public Base(int x) { ... }  
}
```

```
public class Sub extends Base {  
    public Sub(int x) {  
        super(x);  
    }  
}
```

`super` in Methods

`super` can also be used in methods when overloading.

Used to execute a superclass' implementation of a method.

super in Methods

`super` can also be used in methods when overloading.

Used to execute a superclass' implementation of a method.

```
public class Base {  
    public int returnNum() {  
        return 17;  
    }  
}
```


super in Methods

super can also be used in methods when overloading.

Used to execute a superclass' implementation of a method.

```
public class Base {  
    public int returnNum() {  
        return 17;  
    }  
}
```

```
public class Sub extends Base {  
    public int returnNum() {  
        return super.returnNum() + 3;  
    }  
}
```

super in Methods

super can also be used in methods when overloading.

Used to execute a superclass' implementation of a method.

```
public class Base {  
    public int returnNum() {  
        return 17;  
    }  
}
```

```
public class Sub extends Base {  
    public int returnNum() {  
        return super.returnNum() + 3;  
    }  
}
```

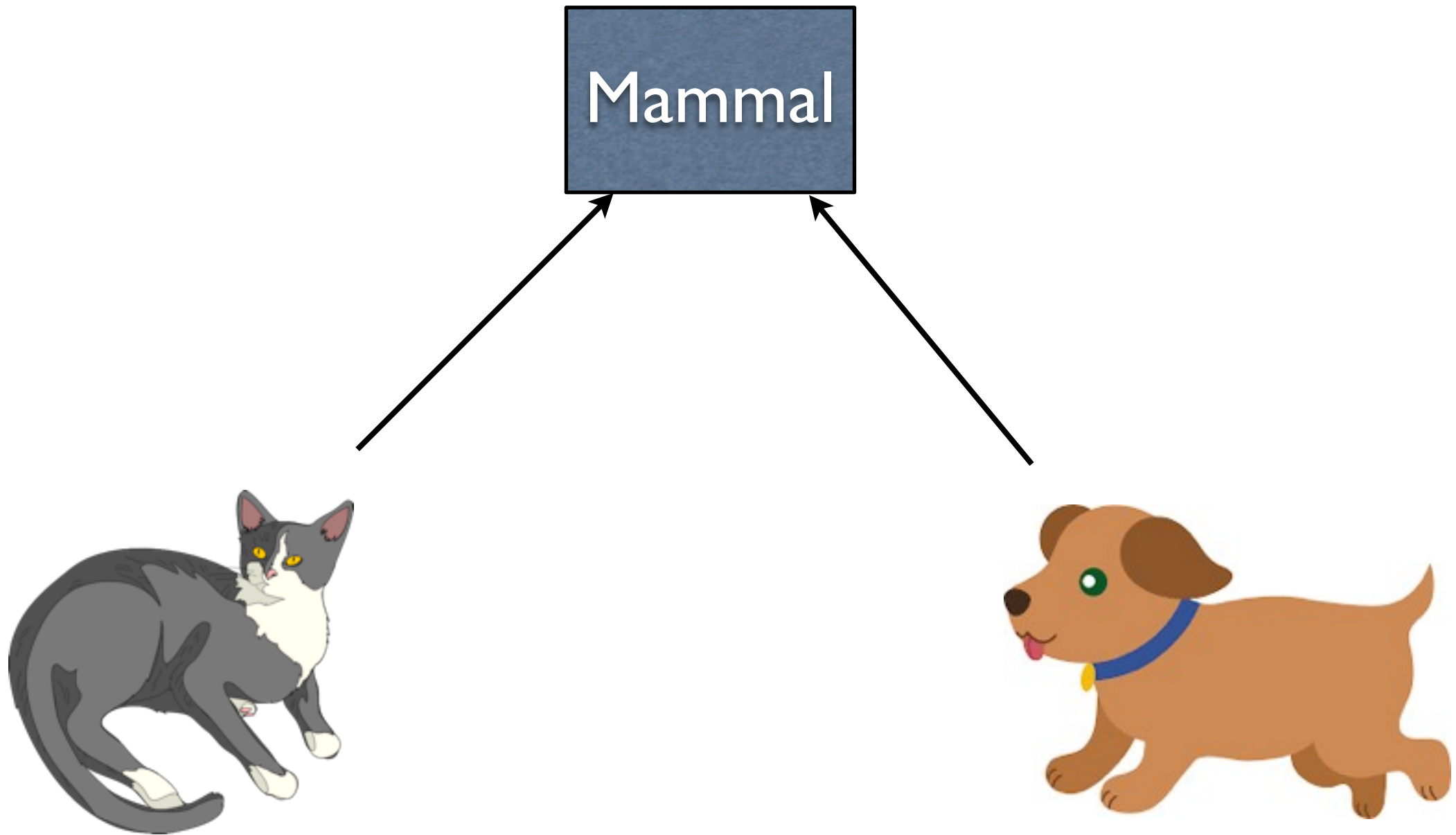
Returns 17

Example

- `Base.java`
- `Sub.java`
- `SuperMethodMain.java`

abstract Classes and Methods

Recap - A Problem



abstract Classes

Allows a class to be extended,
but disallows the creation of instances of that class

abstract **Classes**

Allows a class to be extended,
but disallows the creation of instances of that class

```
public class Mammal {  
    public Mammal(String s) { ... }  
}
```

abstract **Classes**

**Allows a class to be extended,
but disallows the creation of instances of that class**

```
public class Mammal {  
    public Mammal(String s) { ... }  
}  
  
    new Mammal("some string")
```


abstract Classes

Allows a class to be extended,
but disallows the creation of instances of that class

```
public class Mammal {  
    public Mammal(String s) { ... }  
}  
  
new Mammal("some string")
```

```
public abstract class Mammal {  
    public Mammal(String s) { ... }  
}
```

abstract Classes

Allows a class to be extended,
but disallows the creation of instances of that class

```
public class Mammal {  
    public Mammal(String s) { ... }  
}  
  
new Mammal("some string")
```

```
public abstract class Mammal {  
    public Mammal(String s) { ... }  
}  
  
new Mammal("some string")
```

Does not compile

Example

- `AbstractBase.java`
- `AbstractSub.java`
- `AbstractMain.java`

abstract Methods

- Methods of abstract classes can also be defined abstract
 - To be overridden later
- abstract methods have no bodies

abstract Methods

- **Methods of abstract classes can also be defined abstract**
 - **To be overridden later**
- **abstract methods have no bodies**

```
public abstract class Abstract {  
    public abstract int getValue();  
}
```

abstract Methods

- **Methods of abstract classes can also be defined abstract**
 - **To be overridden later**
- **abstract methods have no bodies**

```
public abstract class Abstract {  
    public abstract int getValue();  
}
```

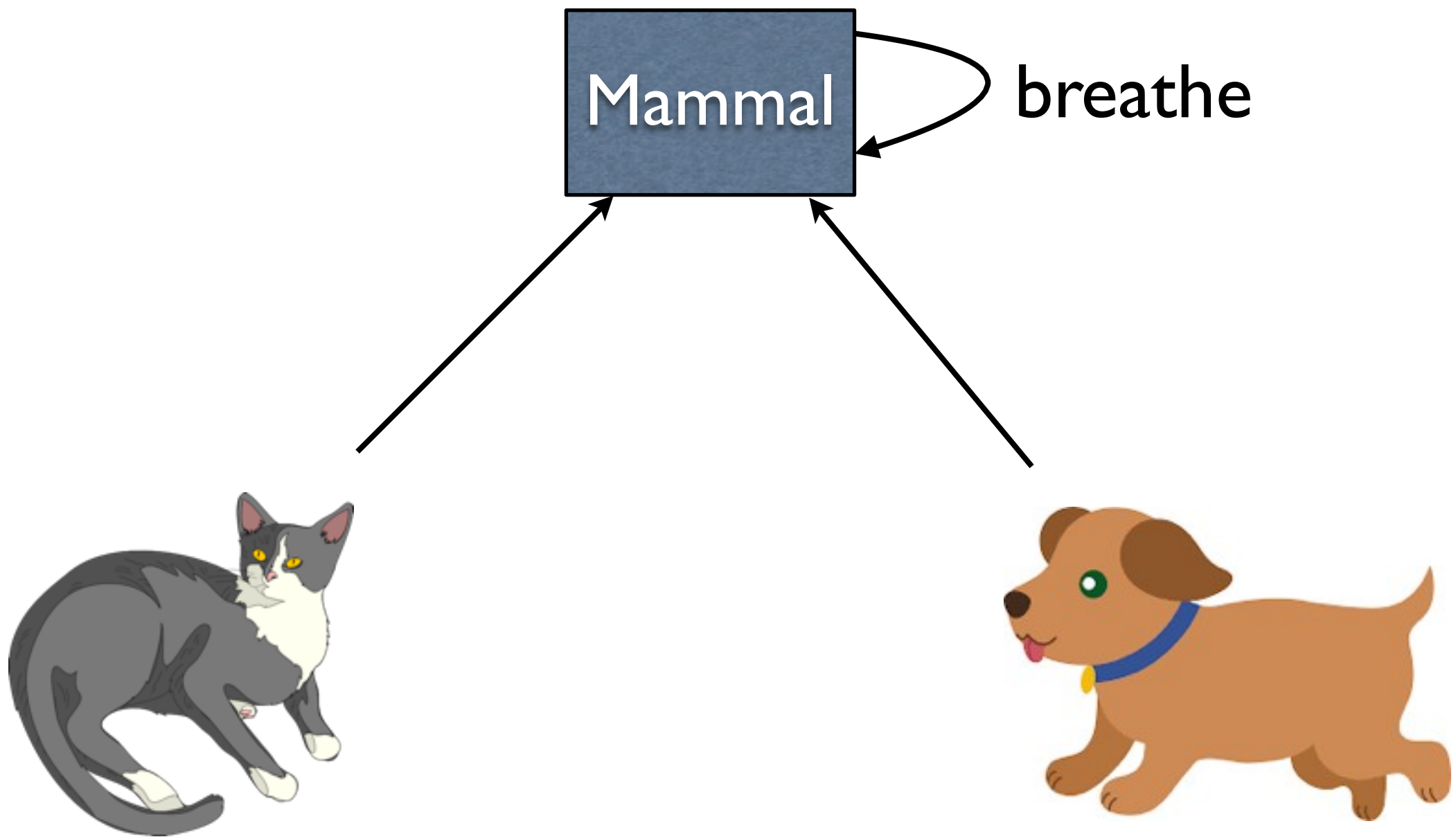
```
public class Sub extends Abstract {  
    public int getValue() { return 5; }  
}
```

Example

- `ArithmeticOperation.java`
- `Add.java`
- `Subtract.java`

Polymorphism

Revisit



```
Cat cat = new Cat ("Tom") ;  
Dog dog = new Dog ("Rover") ;  
cat.breathe() ;  
dog.breathe() ;
```

```
Cat cat = new Cat ("Tom");  
Dog dog = new Dog ("Rover");  
cat.breathe();  
dog.breathe();
```

Tom the mammal takes a breath
Rover the mammal takes a breath

```
Cat cat = new Cat("Tom");  
Dog dog = new Dog("Rover");  
cat.breathe();  
dog.breathe();
```

Tom the mammal takes a breath
Rover the mammal takes a breath

```
Mammal m1 = new Cat("Tom");  
Mammal m2 = new Dog("Rover");  
m1.breathe();  
m2.breathe();
```

```
Cat cat = new Cat("Tom");  
Dog dog = new Dog("Rover");  
cat.breathe();  
dog.breathe();
```

Tom the mammal takes a breath
Rover the mammal takes a breath

```
Mammal m1 = new Cat("Tom");  
Mammal m2 = new Dog("Rover");  
m1.breathe();  
m2.breathe();
```

Tom the mammal takes a breath
Rover the mammal takes a breath

Polymorphism

- “many-forms”
- A Mammal could be a Cat or a Dog
- Specific use in Java: a variable with a superclass type can hold an instance of any subclass, too

Polymorphism

- “many-forms”
- A Mammal could be a Cat or a Dog
- Specific use in Java: a variable with a superclass type can hold an instance of any subclass, too

```
Mammal m1 = new Cat ("Tom");  
Mammal m2 = new Dog ("Rover");
```

Polymorphism

Significance

Can write code without knowing exactly which implementation is used.

Polymorphism

Significance

Can write code without knowing exactly which implementation is used.

```
public static void method(Mammal m) {  
    m.breathe();  
}
```

Example

- `Car.java`
- `SportsCar.java`
- `SemiTruck.java`
- `CarMain.java`

Example

- `MammalRevisited.java`
- `CatRevisited.java`
- `DogRevisited.java`
- `MammalMainRevisited.java`

Polymorphism

1. **Static** binding/Compile-Time binding/Early binding/Method **overloading**.(in same class)
2. **Dynamic** binding/Run-Time binding/Late binding/Method **overriding**.(in different classes)

Polymorphism

Static binding/Compile-Time binding/Early binding/Method overloading.(in same class)

Method overloading example:

```
class Calculation {  
    public void sum(int a, int b) {  
        System.out.println(a + b);  
    }  
    public void sum(int a, int b, int c) {  
        System.out.println(a + b + c);  
    }  
  
    public static void main(String args[]) {  
        Calculation obj = new Calculation();  
        obj.sum(10, 10, 10);    // 30  
        obj.sum(20, 20);        //40  
    }  
}
```

Polymorphism

Dynamic binding/Run-Time binding/Late binding/Method overriding.(in different classes)

Method overriding example:

```
class Animal {
    public void move() {
        System.out.println("Animals can move");
    }
}

class Dog extends Animal {
    public void move() {
        System.out.println("Dogs can walk and run");
    }
}

public class TestDog {
    public static void main(String args[]) {
        Animal a = new Animal();    // Animal reference and object
        Animal b = new Dog();        // Animal reference but Dog
object
        a.move();    //output: Animals can move
        b.move();    //output: Dogs can walk and run
    }
}
```