COMP 110/L Lecture 20

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;
  }
}
```

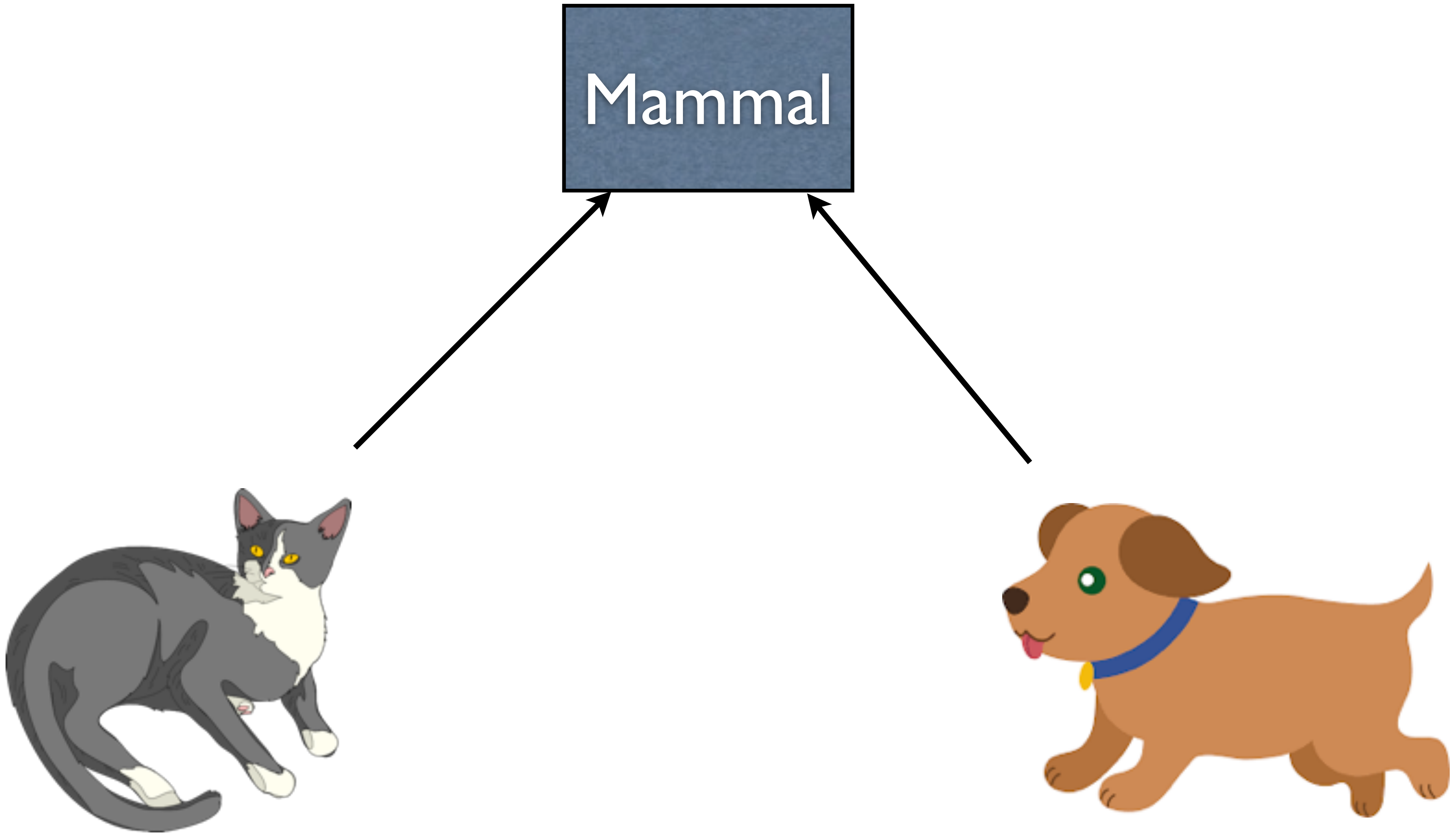
Returns 17

Example

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

`abstract` Classes and Methods

Recap - A Problem



-The example from last time stated that we had Mammal objects, Cat objects, and Dog objects
-Cat and Dog objects were both Mammal objects because of inheritance
-Having just a Mammal object (which isn't a Cat, Dog, or some other actual animal) is strange

`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) { ... }
}
```

–Before we defined this code...

`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")
```

-And we could create instances of this class directly

`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) { ... }
}
```

–If, however, we declare Mammal as an abstract 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) { ... }
}
    new Mammal("some string")
```

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

~~`new Mammal("some string")`~~

Does not compile

–If, however, we declare Mammal as an abstract class...

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();
}
```

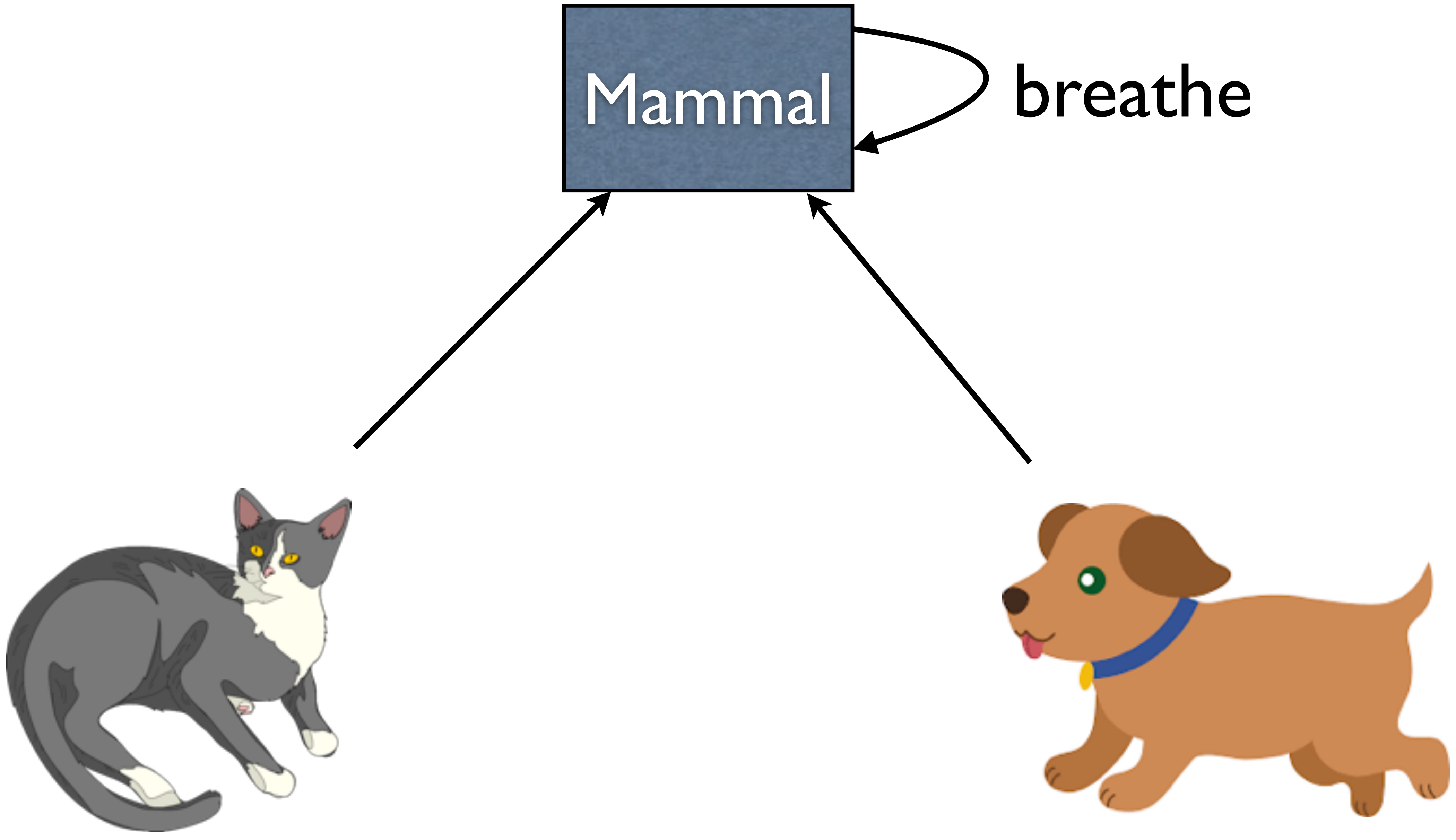
```
public class Sub extends Abstract {
  public int getValue() { return 5; }
}
```

Example

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

Polymorphism

Revisit



-From last time: mammals breathe, so transitively cats and dogs breathe, too
-Phrased another way, all mammals breathe, so if I have any mammal I can ask it to breathe

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

-Snippet of code from the last time: have variables which explicitly track that they point to Cat and Dog objects, and we ask them both to 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
```

-The above code produced the output that each Mammal took 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();
```

-Alternative version: we only track that the Cat and the Dog are Mammals

```
  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
```

-Output does not change at all. m1 knows it's really a Cat and m2 knows it's really a dog

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();
}
```

-I don't need to know if m is a Dog or a Cat in order to write the above code, only that m is a Mammal so I can call the breathe() method
-Key point: breathe() can do different things

Example

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

Example

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