COMP 110/L Lecture 19

Maryam Jalali

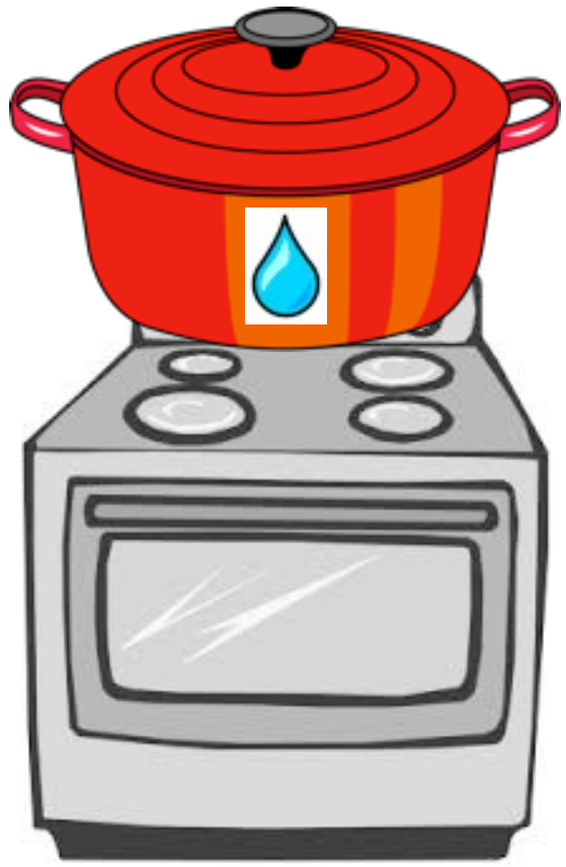
Slides adapted from Dr. Kyle Dewey

Outline

- Inheritance
  - `extends`
  - `super`
- Method overriding
- Automatically-generated constructors

Inheritance

Recap



Mammal

Mammal

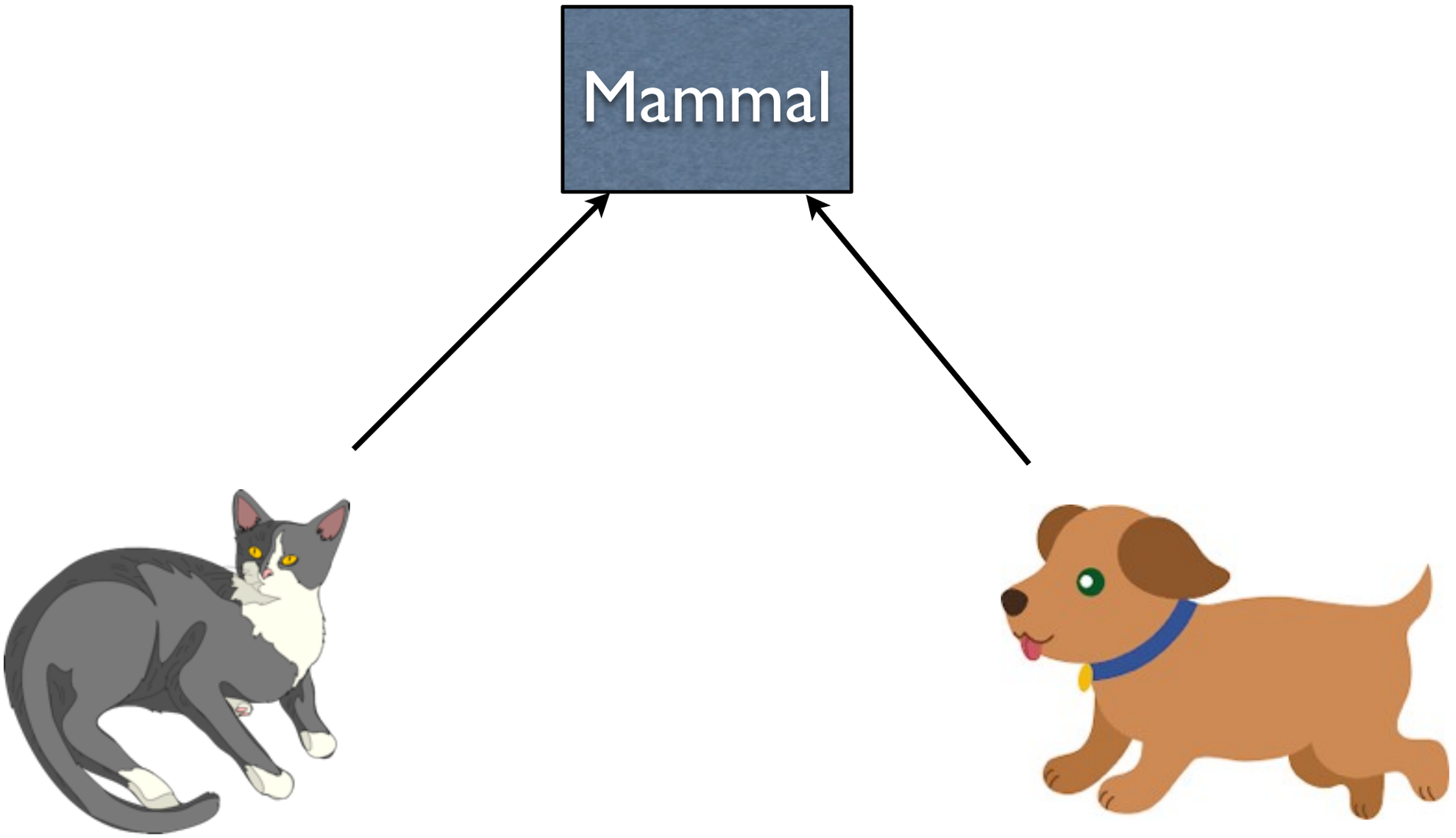
Mammal

Mammal

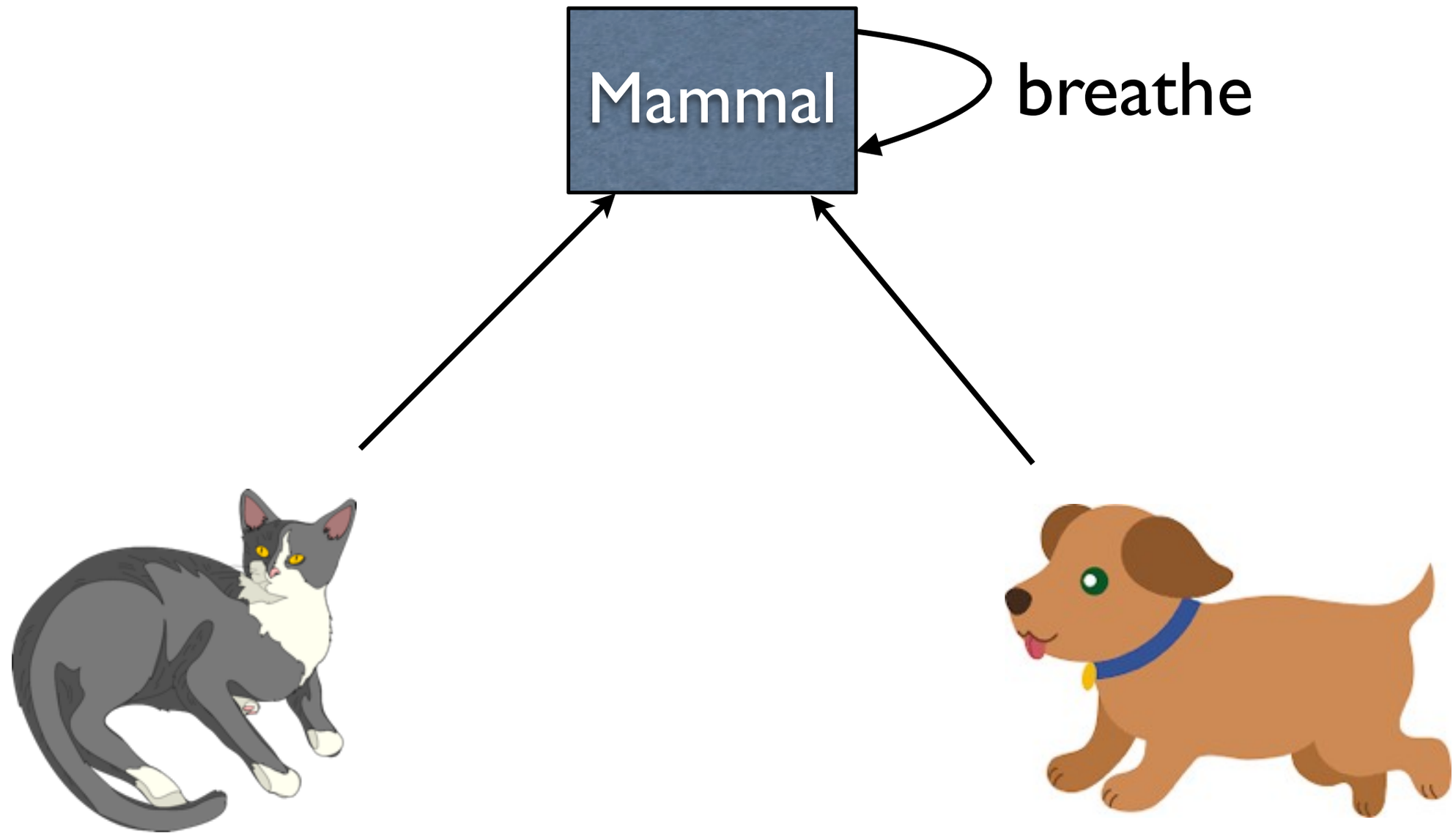
breathe

breathe

Inheritance



Inheritance



extends

`extends`

States that a *subclass* inherits from a *parent* class

extends

States that a *subclass* inherits from a *parent* class

```
public class Mammal {
   ...
}
```

extends

States that a *subclass* inherits from a *parent* class

```
public class Mammal {
   ...
}
```

```
public class Cat extends Mammal {
   ...
}
```

extends

States that a *subclass* inherits from a *parent* class

```
public class Mammal {
   ...
}
```

```
public class Cat extends Mammal {
   ...
}
```

```
public class Dog extends Mammal {
   ...
}
```

super

`super`

Used to invoke the constructor of the parent class.
Another name for the parent class is the *superclass*.

super

Used to invoke the constructor of the parent class.
Another name for the parent class is the *superclass*.

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

super

Used to invoke the constructor of the parent class.
Another name for the parent class is the *superclass*.

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

```
public class Child extends BaseClass {
   public Child(String s) {
      super(s);
   }
}
```

Example

- `Mammal.java`
- `Cat.java`
- `Dog.java`
- `MammalMain.java`

Method Overriding

`toString()` Revisit

`toString()` Revisit

```
public String toString() {
   ...
}
```

`toString()` Revisit

```
public String toString() {
   ...
}
```

```
Rectangle(3, 4)
```

`toString()` **Revisit**

```
public String toString() {
   ...
}
```

```
Rectangle(3, 4)
```

```
Rectangle@302b09c9
```

`toString()` **Revisit**

```
public String toString() {
   ...
}
```

```
Rectangle(3, 4)
```

```
Rectangle@302b09c9
```

Key point: even without `toString()` defined, a `String` was still produced.

Base `toString()` Origin

- All classes inherit from `Object`, **even if** you don't explicitly say so
- `Object` defines its own `toString()` that produces `Rectangle@302b09c9`

Base `toString()` Origin

- All classes inherit from `Object`, **even if** you don't explicitly say so
- `Object` **defines its own** `toString()` **that produces** `Rectangle@302b09c9`

```
public class Object {
  public String toString() { ... }
}
```

Base `toString()` Origin

- All classes inherit from `Object`, **even if** you don't explicitly say so
- `Object` **defines its own** `toString()` **that produces** `Rectangle@302b09c9`

```
public class Object {
  public String toString() { ... }
}
```

```
public class Rectangle { ... }
```

Base `toString()` Origin

- All classes inherit from `Object`, **even if** you don't explicitly say so
- `Object` **defines its own** `toString()` **that produces** `Rectangle@302b09c9`

```
public class Object {
  public String toString() { ... }
}
```

```
public class Rectangle { ... }
```

```
public class Rectangle extends Object {
  ...
}
```

Overriding Methods

- You can *override* a method definition in a base class by defining a method with the same signature in a subclass
- The method in the subclass will execute *instead of* the method in the parent class

Overriding Methods

- You can *override* a method definition in a base class by defining a method with the same signature in a subclass
- The method in the subclasswill execute *instead of* the method in the parent class

```
public class Rectangle {
    public String toString()
{
        ...
    }
}
```

Overriding Methods

- You can *override* a method definition in a base class by defining a method with the same signature in a subclass
- The method in the subclass will execute *instead of* the method in the parent class

```
public class Rectangle extends Object {
   public String toString() {
      ...
   }
}
```

Example

- `OverrideBase.java`
- `OverrideSub.java`
- `OverrideMain.java`

Automatically-Generated Constructors

Automatic Constructors

If you don't define any constructors,
Java will define one for you which takes no arguments.

Automatic Constructors

If you don't define any constructors,
Java will define one for you which takes no arguments.

```
public class MyClass {
}
```

Automatic Constructors

If you don't define any constructors,
Java will define one for you which takes no arguments.

```
public class MyClass {
}
```

```
public class MyClass {
   public MyClass() {}
}
```

Example:

`AutomaticConstructor.java`

Automatic Constructors

This also applies to subclasses,
as long as the base class has a no-argument constructor

Automatic Constructors

This also applies to subclasses,
as long as the base class has a no-argument constructor

```
public class MyBase {}
public class MySub extends MyBase {}
```

Automatic Constructors

This also applies to subclasses,
as long as the base class has a no-argument constructor

```
public class MyBase {}
public class MySub extends MyBase {}
```

```
public class MyBase {
   public MyBase() {}
}

public class MySub extends MyBase {
   public MySub() { super(); }
}
```

Automatic Constructors

This also applies to subclasses,

as long as the base class has a no-argument constructor

Automatic Constructors

This also applies to subclasses,

as long as the base class has a no-argument constructor

```
public class MyBase {
  // explicit non-no-arg constructor
  // defined - no automatically
  // generated constructors
  public MyBase(int x) {}
}
public class MySub extends MyBase {}
```

Automatic Constructors

This also applies to subclasses,

as long as the base class has a no-argument constructor

```
public class MyBase {
  // explicit non-no-arg constructor
  // defined - no automatically
  // generated constructors
  public MyBase(int x) {}
}
public class MySub extends MyBase {
  public MySub() { super(); }
}
```

Automatic Constructors

This also applies to subclasses,
as long as the base class has a no-argument constructor

```
public class MyBase {
  // explicit non-no-arg constructor
  // defined - no automatically
  // generated constructors
  public MyBase(int x) {}
}
public class MySub extends MyBase {
  public MySub() { super(); }
}
```

Does not exist - code will not compile