CSE 1322
Module 3 – Part 1

Inheritance

KENNESAW STATE UNIVERSITY

Relations

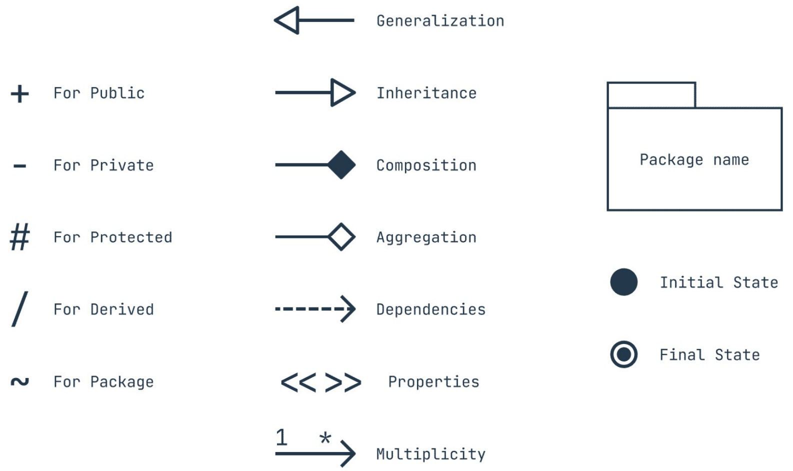
- We can define certain relations between different classes.
- Previously we have done “Composition” or “Aggregation” type relations.
  - For example, whenever a class contains another class as an attribute.
- In this module, we will introduce the inheritance relation.

Unified Modeling Language

- Whenever we work with complex OOP programs, we are going to end up dealing with lots of different types of classes.
- These classes will interact and relate to one another with different types of relations.
- The Unified Modeling Language allows us to draw and visualize these relations.

Unified Modeling Language

- Furthermore, with UML we can also represent the different component and details of a class such as attributes and behaviors and each of their access modifiers.



+ For Public

- For Private

# For Protected

/ For Derived

~ For Package

Generalization

Inheritance

Composition

Aggregation

Dependencies

<< >> Properties

1 * Multiplicity

Package name

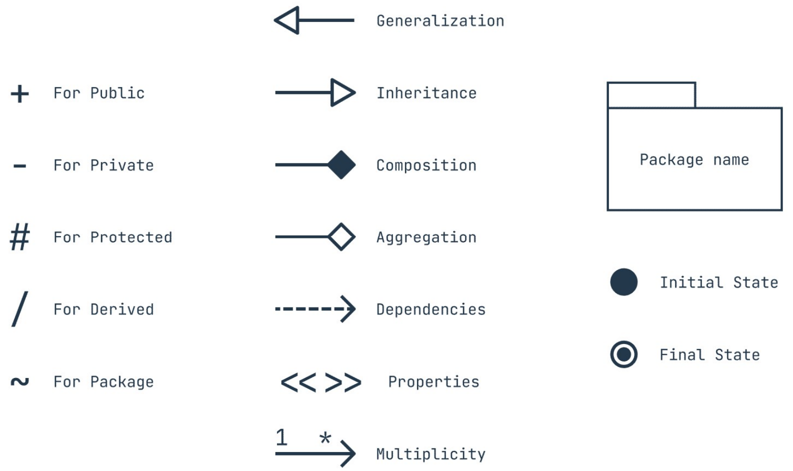
Initial State

Final State

KENNESAW STATE UNIVERSITY

Unified Modeling Language

- Each class is drawn as a rectangle, with the class name at the top, followed by all the attributes, then all the methods.
- We also specify their types or their return types.
- Constructors are often not mentioned.



+ For Public

- For Private

# For Protected

/ For Derived

~ For Package

Generalization

Inheritance

Composition

Aggregation

Dependencies

<< >> Properties

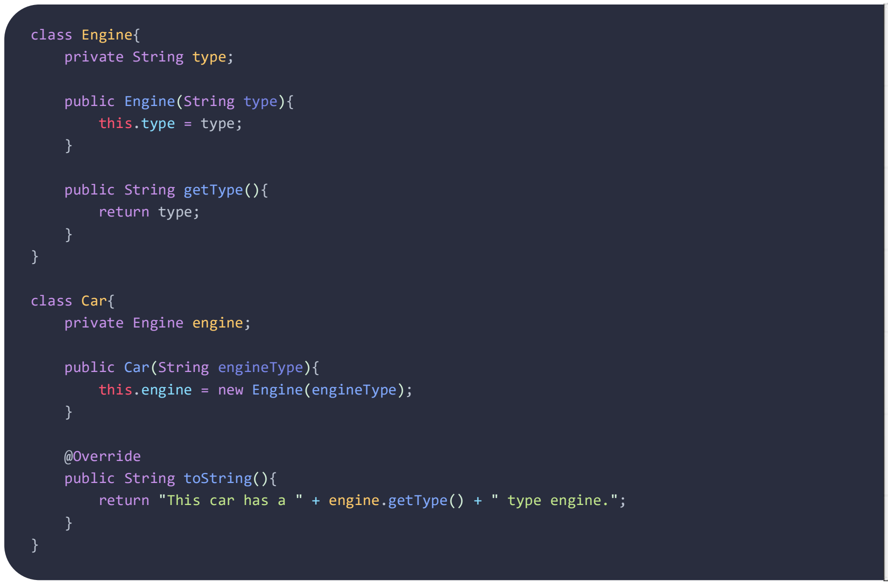
1 * Multiplicity

Package name

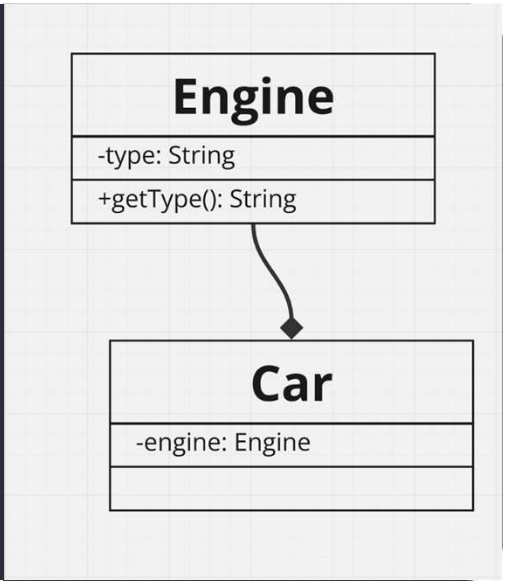
Initial State

Final State

Unified Modeling Language



```java
class Engine{
    private String type;

    public Engine(String type){
        this.type = type;
    }

    public String getType(){
        return type;
    }
}

class Car{
    private Engine engine;

    public Car(String engineType){
        this.engine = new Engine(engineType);
    }

    @Override
    public String toString(){
        return "This car has a " + engine.getType() + " type engine.";
    }
}
```



Engine

-type: String

+getType(): String

Car

-engine: Engine

Week-5/Composition.java

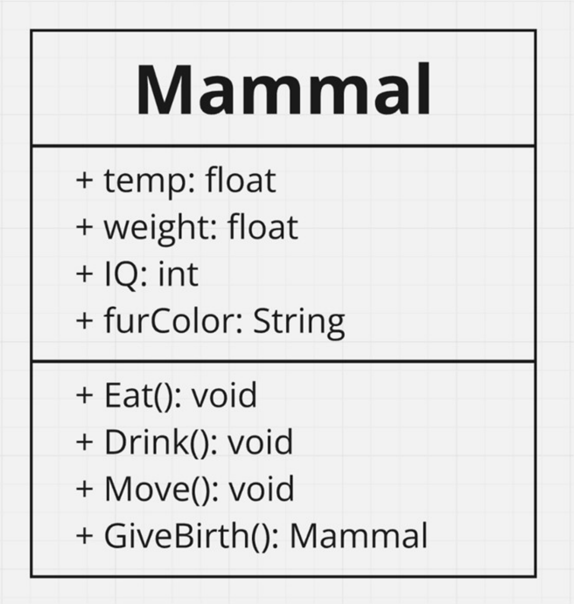
KENNESAW STATE UNIVERSITY

Unified Modeling Language

- We are not going to cover in-depth this topic, but you are going to have to create or generate a UML diagram for you Lab Assignments and Labs.
- Make sure you attend your Lab to learn mode details into this.

An Example – Mammal Class

- Attributes
    - Temperature
    - Weight
    - Intelligence Level
    - Fur Color
- Behaviors
    - Eat
    - Drink
    - Move
    - Give Birth



Mammal

+ temp: float
+ weight: float
+ IQ: int
+ furColor: String

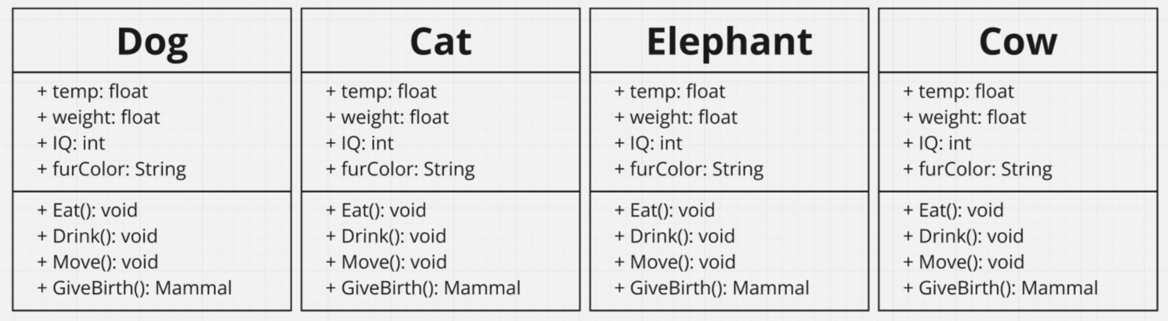
+ Eat(): void
+ Drink(): void
+ Move(): void
+ GiveBirth(): Mammal

Inheritance

- Inheritance allows a new class (child) to “inherit” an existing class’s (parent) members (attributes and behaviors).
  - Private members do get inherited at the object level but are not directly accessible in the **child** class, so make sure to use getters and setter functions from the **parent** class.
- Inheritance allows us to re-use code!

Inheritance

- Let’s say that we want to expand our program and add Dogs, Cats, Elephants, Cows.
- We could manually implement each:

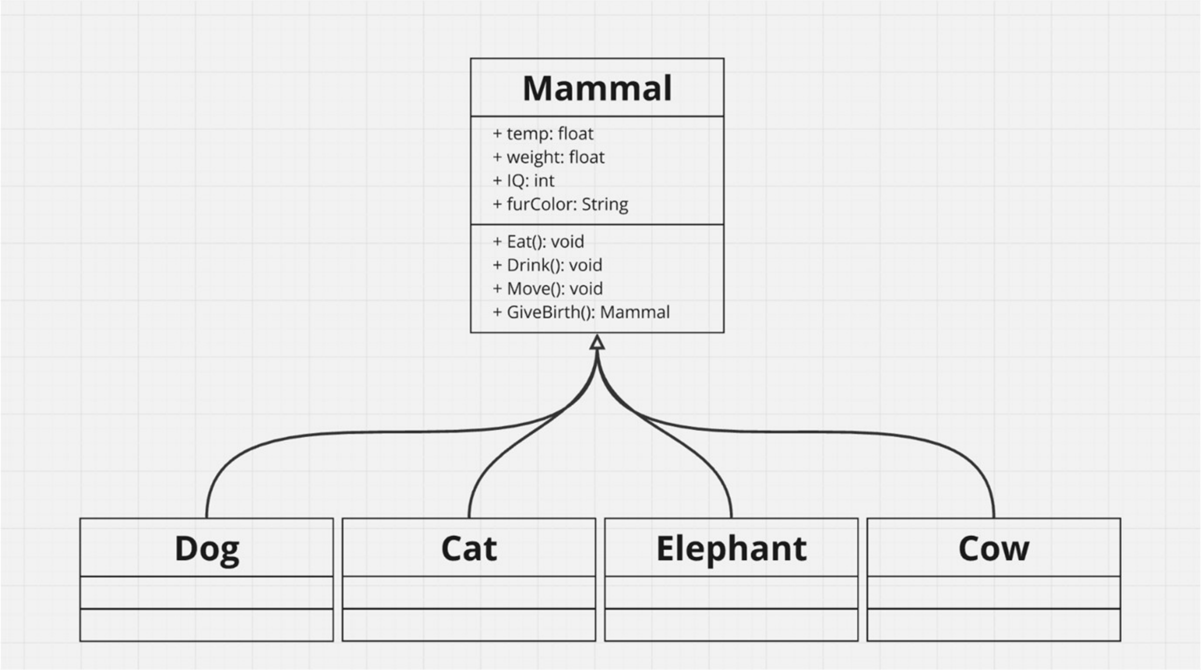


Dog	Cat	Elephant	Cow
+ temp: float + weight: float + IQ: int + furColor: String	+ temp: float + weight: float + IQ: int + furColor: String	+ temp: float + weight: float + IQ: int + furColor: String	+ temp: float + weight: float + IQ: int + furColor: String
+ Eat(): void + Drink(): void + Move(): void + GiveBirth(): Mammal	+ Eat(): void + Drink(): void + Move(): void + GiveBirth(): Mammal	+ Eat(): void + Drink(): void + Move(): void + GiveBirth(): Mammal	+ Eat(): void + Drink(): void + Move(): void + GiveBirth(): Mammal

Inheritance

- Or these animals could inherit the properties of Mammal:

- We use the “white” arrow to show inheritance.



Inheritance

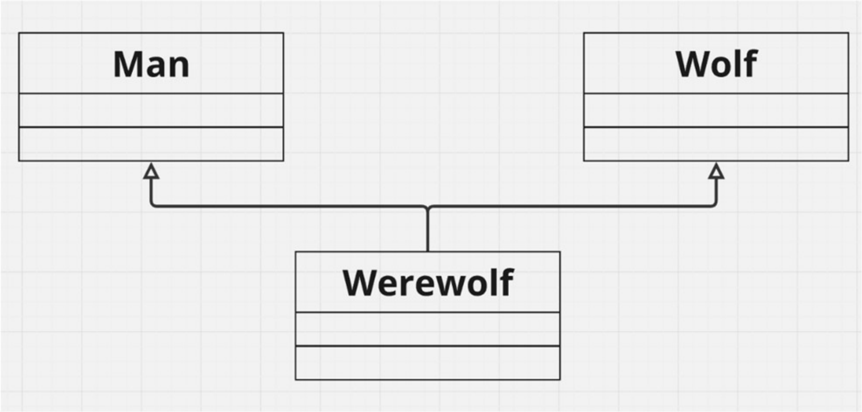
- Through Inheritance, we can make all those different animal classes share a similar set of attributes and behaviors.
- This means that we do not need to declare the weight attribute for the **Dog** class since it inherits this attribute from **Mammal**.

Inheritance

- Now that the Dog, Cat, Elephant, and Cow classes share the same set of attributes, we can now develop each class with their own unique set of attributes and behaviors.
- For example, Dogs could have a **barkVolume** integer attribute, Cats can have a **likesToClimb** Boolean attribute, Elephants could have a **hasTusks** Boolean attribute, and Cows could have a **milkProduction** integer attribute.

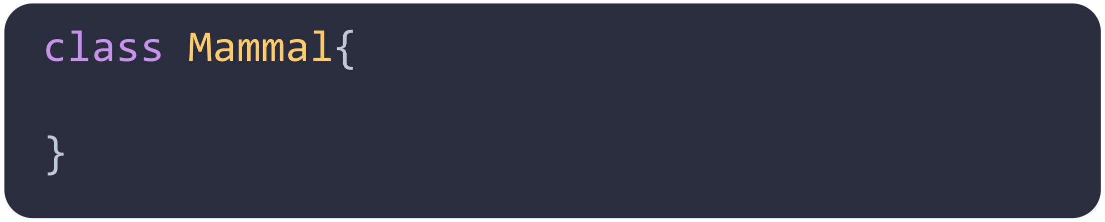
Inheritance – Multi-inheritance

- One constraint with inheritance is that a **child** class can only inherit **one parent** class.
- This is not valid:



Inheritance – Syntax

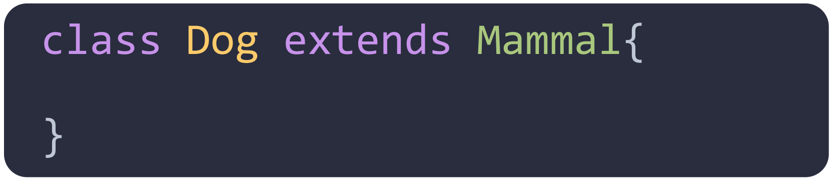
If a parent class is defined as:



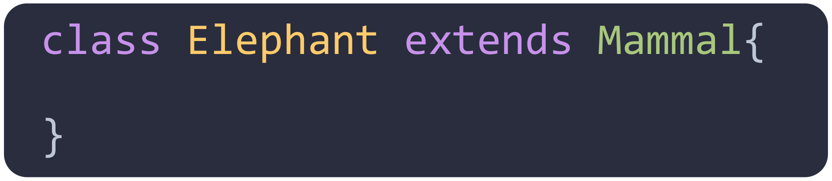
```
class Mammal{

}
```

Inheritance – Syntax

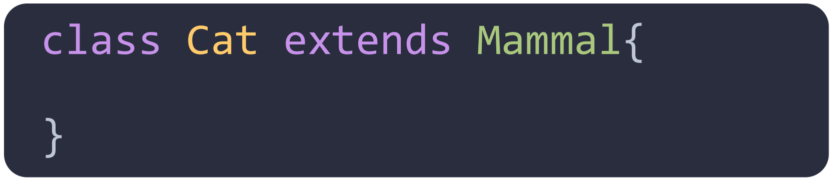
Then the child classes can be defined as:



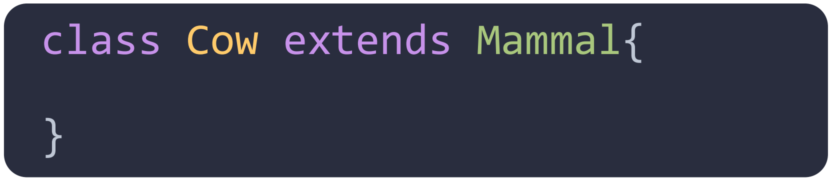
```
class Dog extends Mammal{

}
```



```
class Elephant extends Mammal{

}
```



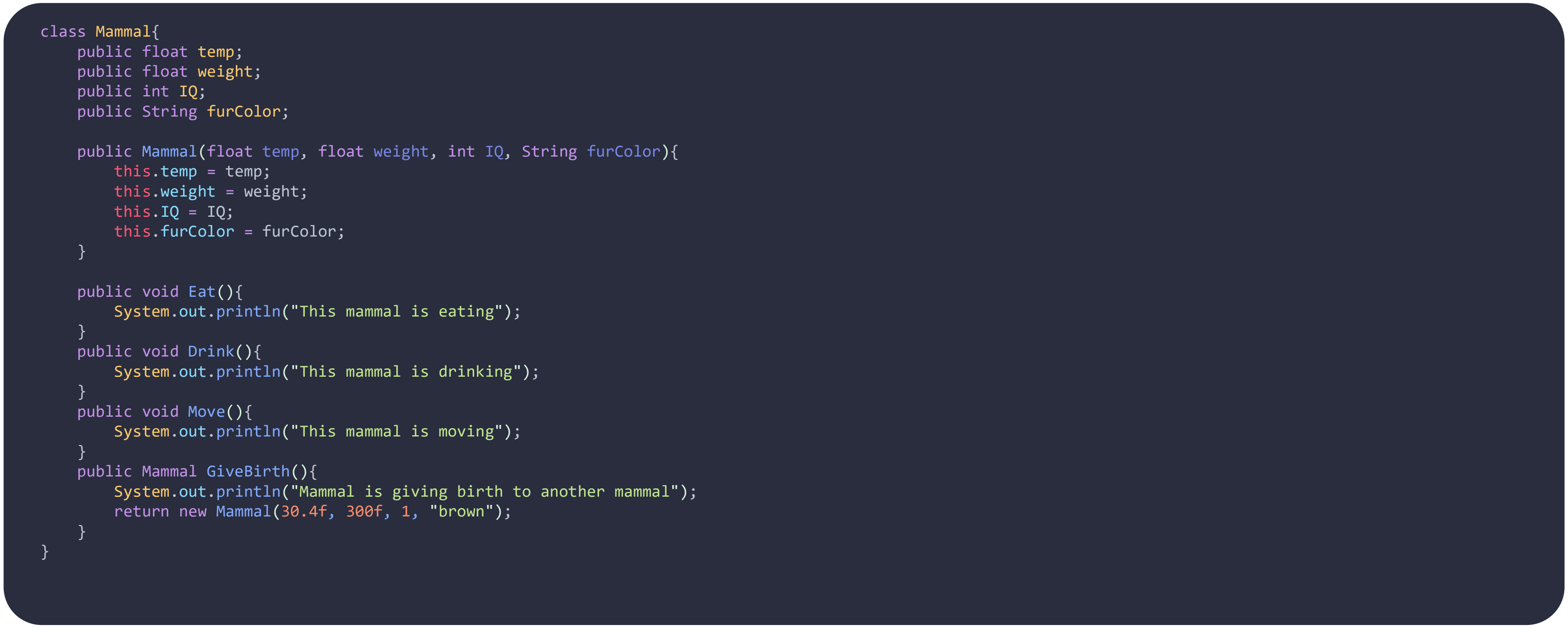
```
class Cat extends Mammal{

}
```



```
class Cow extends Mammal{

}
```

In short, we use the **extends** keyword.

Inheritance – Mammal Class

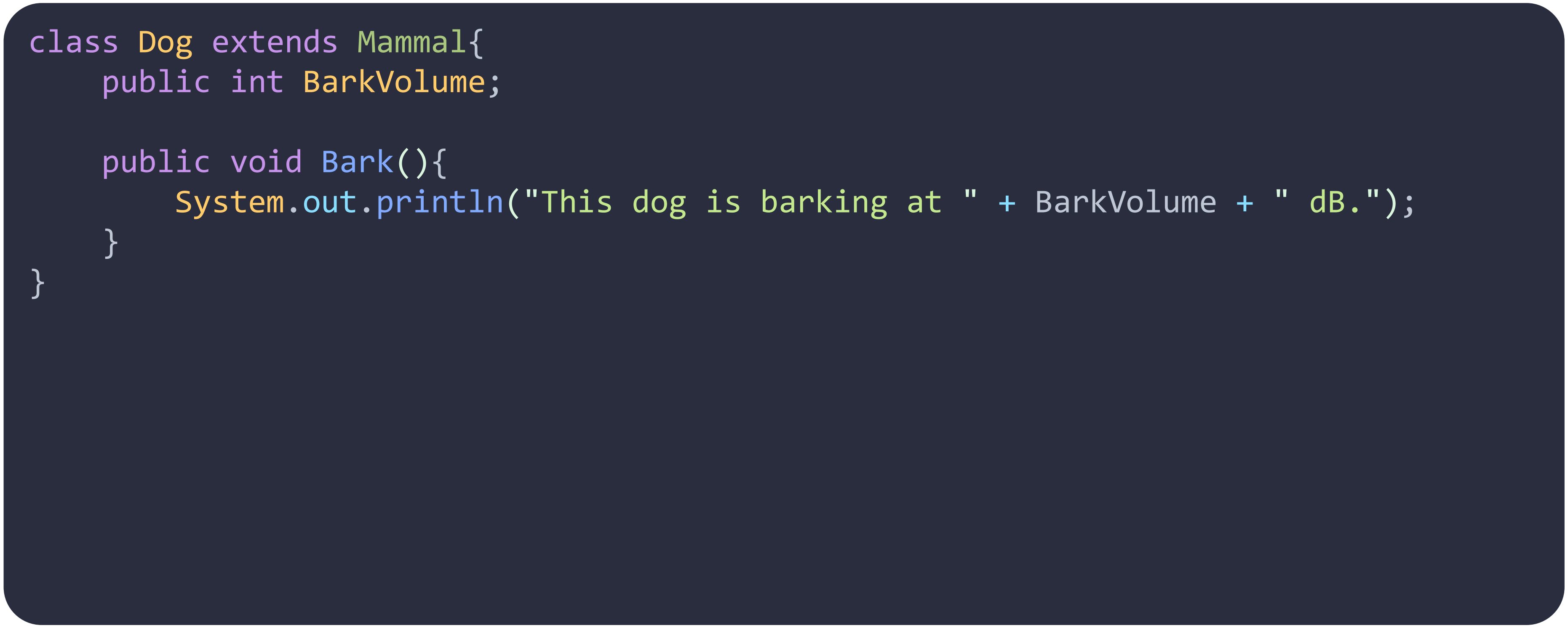


```java
class Mammal{
    public float temp;
    public float weight;
    public int IQ;
    public String furColor;

    public Mammal(float temp, float weight, int IQ, String furColor){
        this.temp = temp;
        this.weight = weight;
        this.IQ = IQ;
        this.furColor = furColor;
    }

    public void Eat(){
        System.out.println("This mammal is eating");
    }
    public void Drink(){
        System.out.println("This mammal is drinking");
    }
    public void Move(){
        System.out.println("This mammal is moving");
    }
    public Mammal GiveBirth(){
        System.out.println("Mammal is giving birth to another mammal");
        return new Mammal(30.4f, 300f, 1, "brown");
    }
}
```

Week-5/Mammal/Driver.java

KENNESAW STATE UNIVERSITY

Inheritance – Dog Class



```java
class Dog extends Mammal{
    public int BarkVolume;

    public void Bark(){
        System.out.println("This dog is barking at " + BarkVolume + " dB.");
    }
}
```

Week-5/Mammal/Driver.java

KENNESAW STATE
UNIVERSITY

Inheritance – Dog Class

- At this point, you may notice that your IDE is giving you an error:

  ```
  Implicit super constructor Mammal() is undefined for
  default constructor. Must define an explicit constructor
  ```

- This is because **Mammal** has an overloaded constructor, and since **Dog** inherits from it, we need to call the **Mammal** constructor inside the **Dog** class constructor.

Inheritance – super

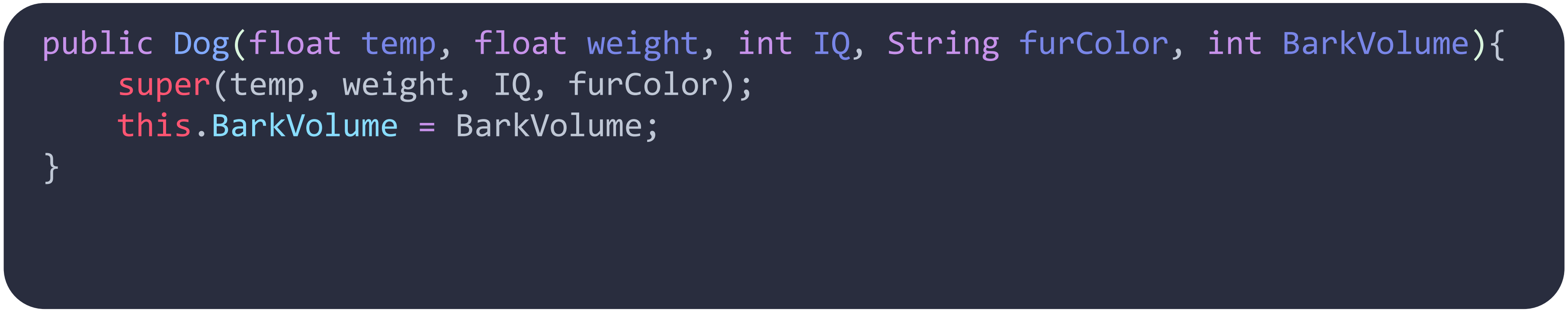
- Remember that we have a keyword to explicitly mention or reference the current object instance inside of the class
  - this() -> to reference the default constructor.
  - this.BarkVolume -> to reference the objects's BarkVolume attribute.

Inheritance – super

- We also have a way to explicitly mention or reference the current object’s **parent** class with the **super** keyword:
  - super() -> reference the parent’s default constructor
  - super.furColor -> reference the furColor attribute.

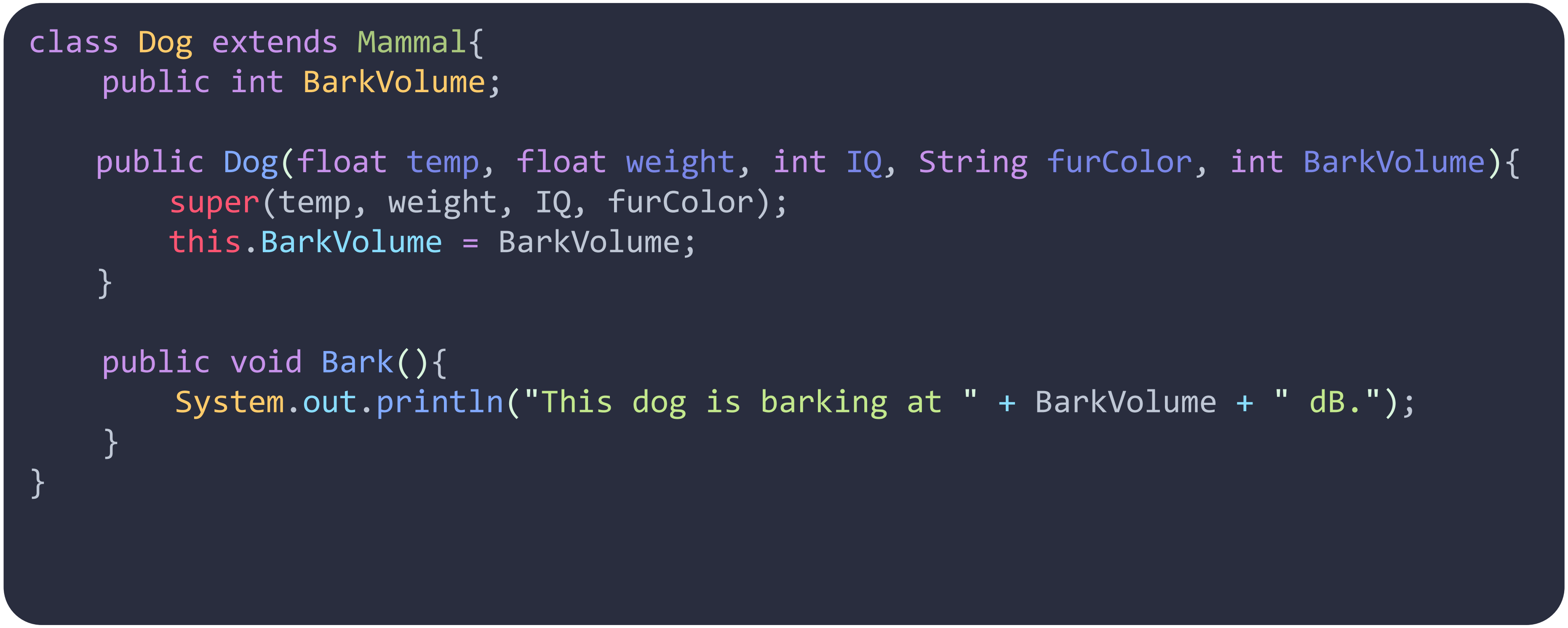
Inheritance – super

- Going back to our issue, the error message mentioned that we must explicitly define the parent constructor:



```
public Dog(float temp, float weight, int IQ, String furColor, int BarkVolume){
    super(temp, weight, IQ, furColor);
    this.BarkVolume = BarkVolume;
}
```

Inheritance – Dog Class



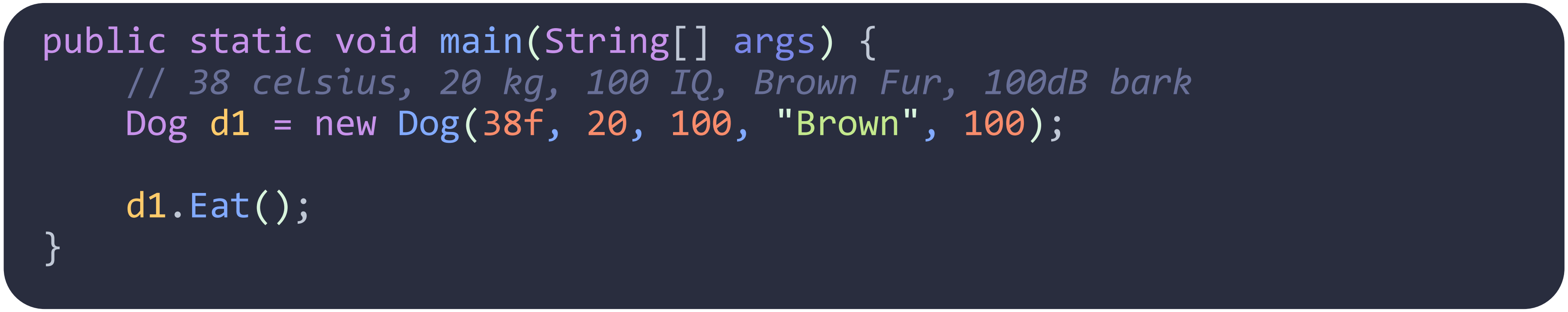
```java
class Dog extends Mammal{
    public int BarkVolume;

    public Dog(float temp, float weight, int IQ, String furColor, int BarkVolume){
        super(temp, weight, IQ, furColor);
        this.BarkVolume = BarkVolume;
    }

    public void Bark(){
        System.out.println("This dog is barking at " + BarkVolume + " dB.");
    }
}
```

Week-5/Mammal/Driver.java

KENNESAW STATE UNIVERSITY

Inheritance – Dog Class

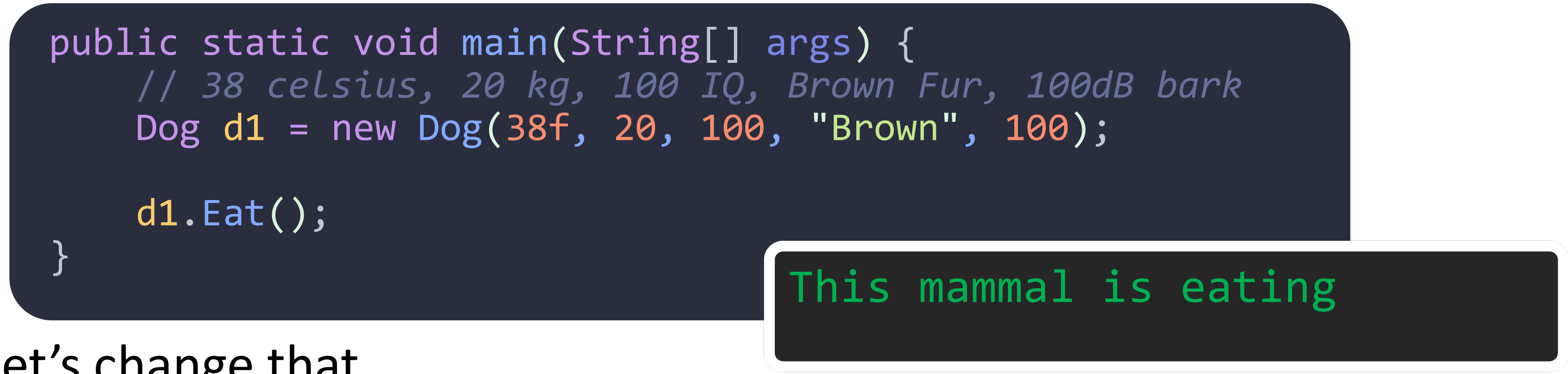
- Now we should be ready to create a **Dog** object and use it in our program.



```
public static void main(String[] args) {
    // 38 celsius, 20 kg, 100 IQ, Brown Fur, 100dB bark
    Dog d1 = new Dog(38f, 20, 100, "Brown", 100);

    d1.Eat();
}
```

Inheritance – Dog Class

- The output seems a bit wrong:



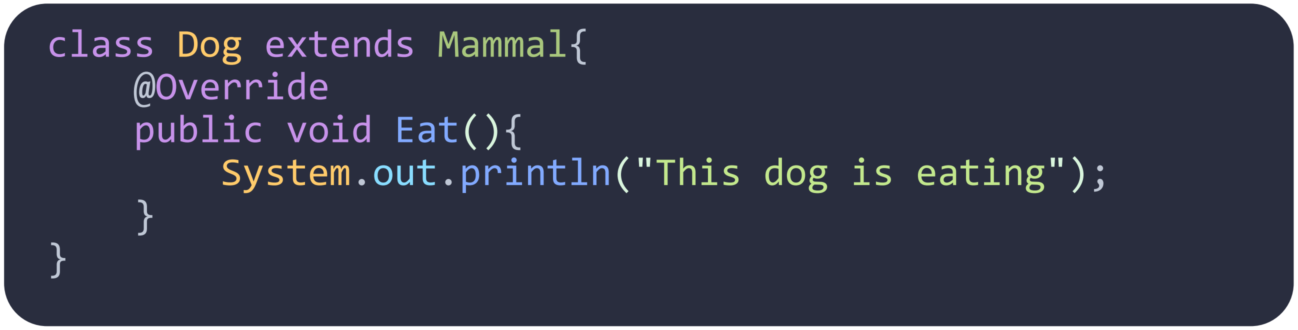
```java
public static void main(String[] args) {
    // 38 celsius, 20 kg, 100 IQ, Brown Fur, 100dB bark
    Dog d1 = new Dog(38f, 20, 100, "Brown", 100);

    d1.Eat();
}
```

```
This mammal is eating
```

- Let’s change that

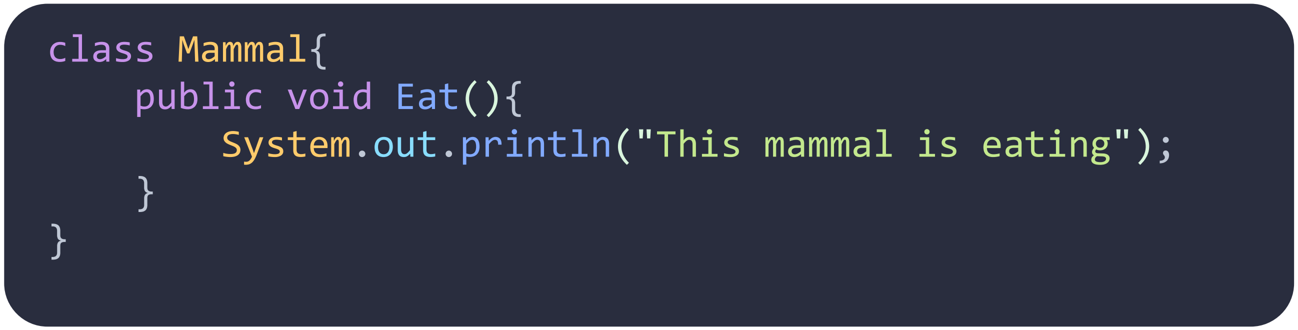
Inheritance – Overriding

- As discussed previously, we can override functions or behaviors inherited.
- To override a function, we must declare the same **function header**.

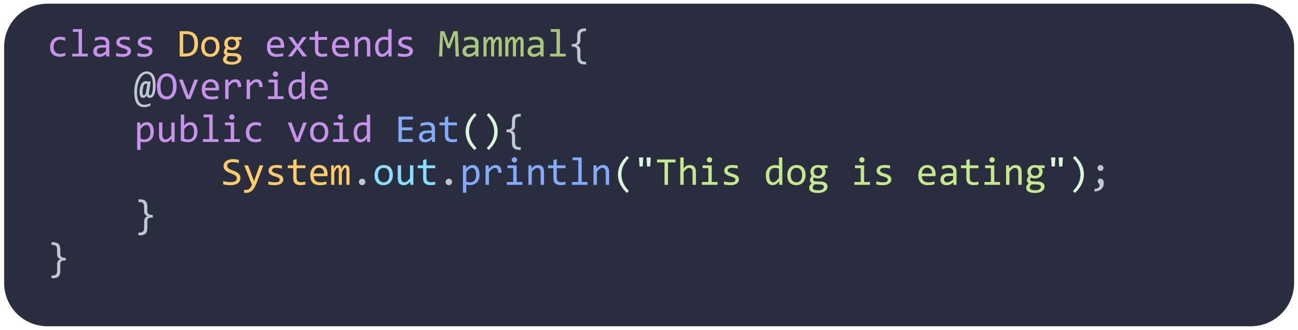


```java
class Dog extends Mammal{
    @Override
    public void Eat(){
        System.out.println("This dog is eating");
    }
}
```

Inheritance – Overriding



```java
class Mammal{
    public void Eat(){
        System.out.println("This mammal is eating");
    }
}
```

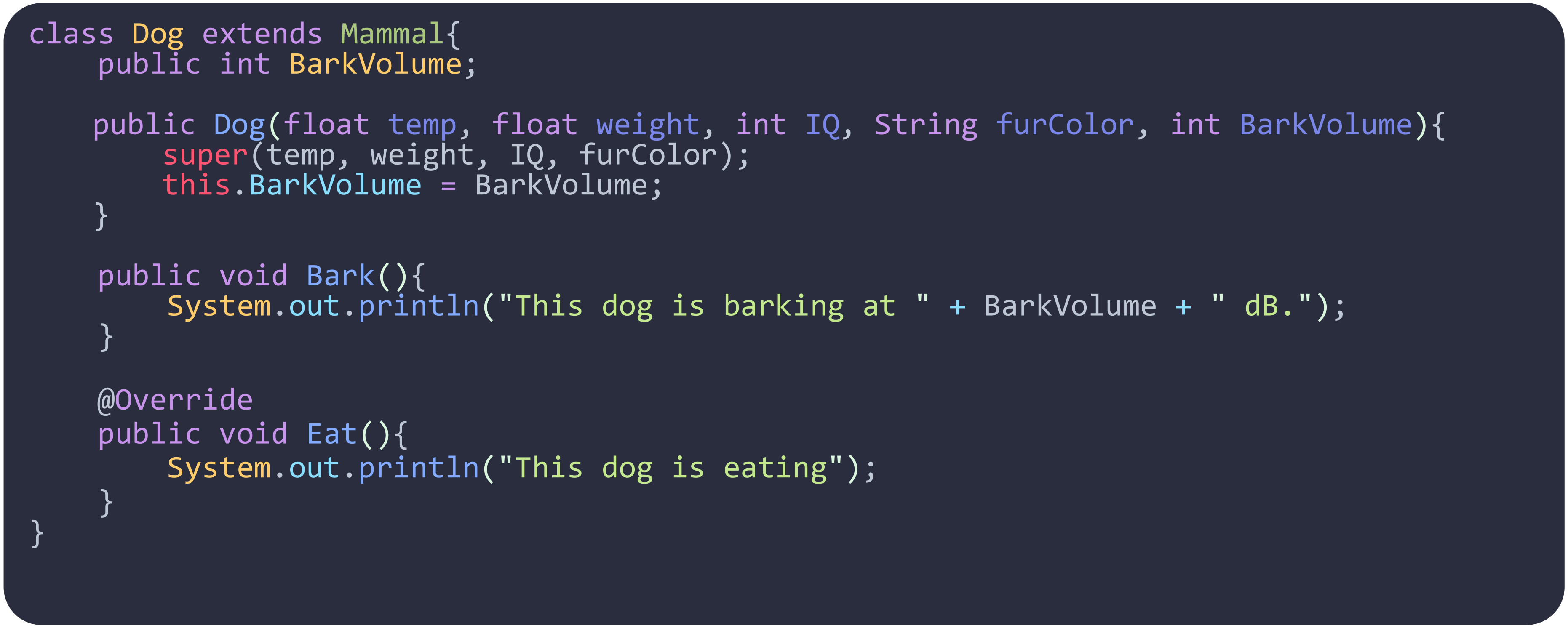


```java
class Dog extends Mammal{
    @Override
    public void Eat(){
        System.out.println("This dog is eating");
    }
}
```

Inheritance – Overriding

- Remember that you can add the Override annotation **@Override** at the top of the overriding function.
- This will help you ensure that you are overriding a function that you are inheriting
- It also help ensure that you have defined the same **function header** to the function you are overriding.

Inheritance – Dog Class

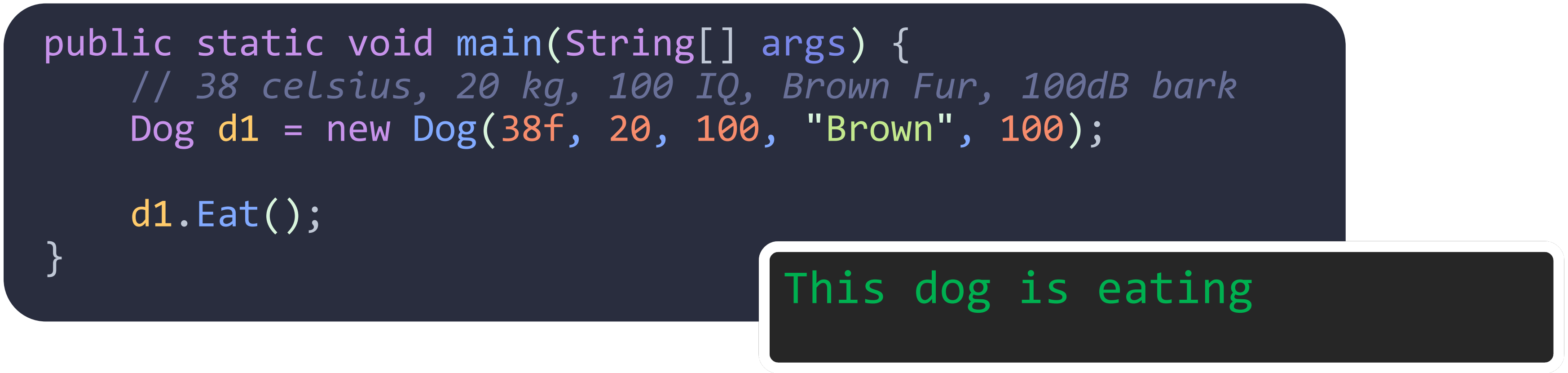


```java
class Dog extends Mammal{
    public int BarkVolume;

    public Dog(float temp, float weight, int IQ, String furColor, int BarkVolume){
        super(temp, weight, IQ, furColor);
        this.BarkVolume = BarkVolume;
    }

    public void Bark(){
        System.out.println("This dog is barking at " + BarkVolume + " dB.");
    }

    @Override
    public void Eat(){
        System.out.println("This dog is eating");
    }
}
```

Week-5/Mammal/Driver.java

Inheritance – Dog Class

- Now the output will be more fitting:

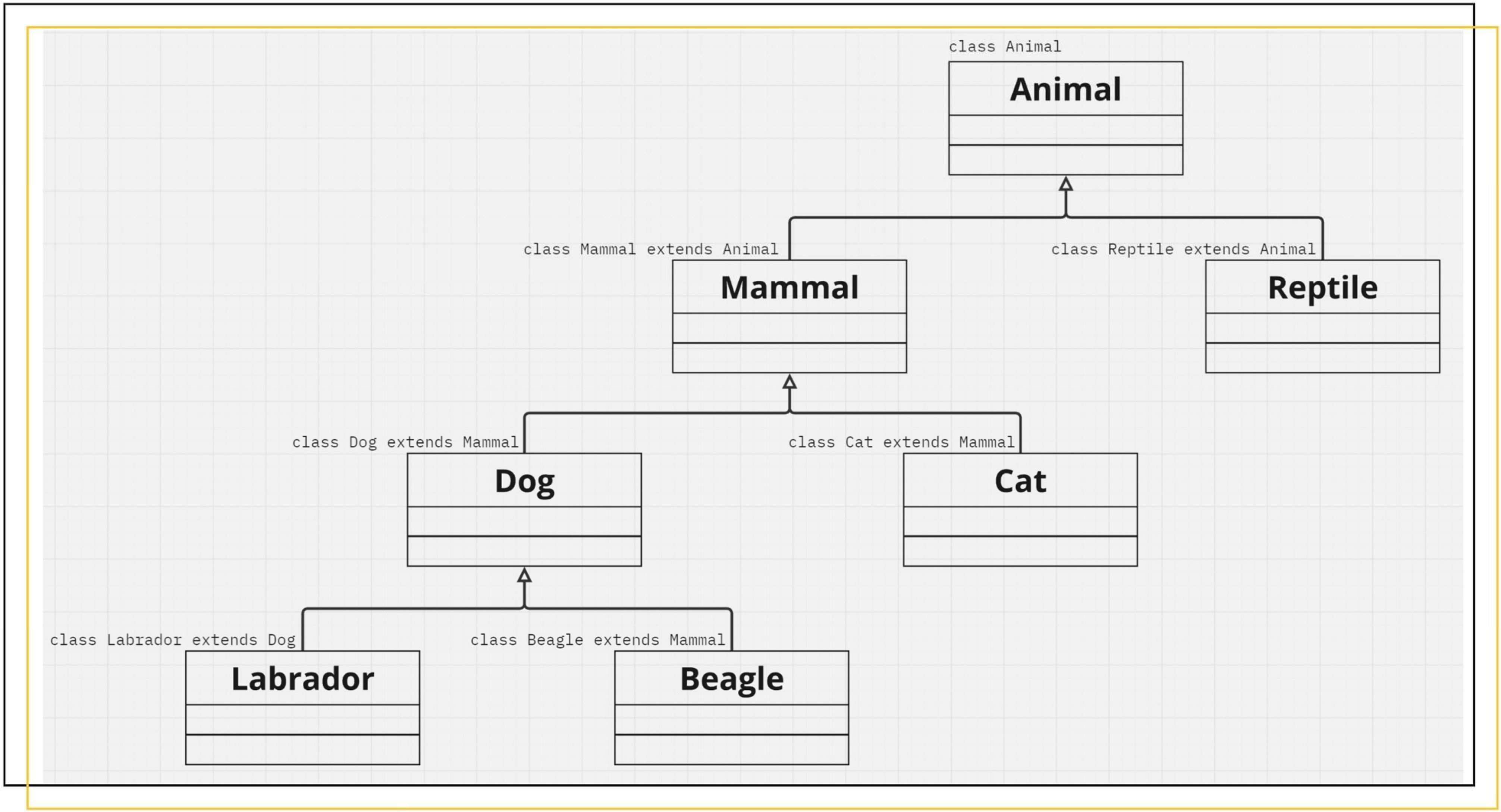


```java
public static void main(String[] args) {
    // 38 celsius, 20 kg, 100 IQ, Brown Fur, 100dB bark
    Dog d1 = new Dog(38f, 20, 100, "Brown", 100);

    d1.Eat();
}
```

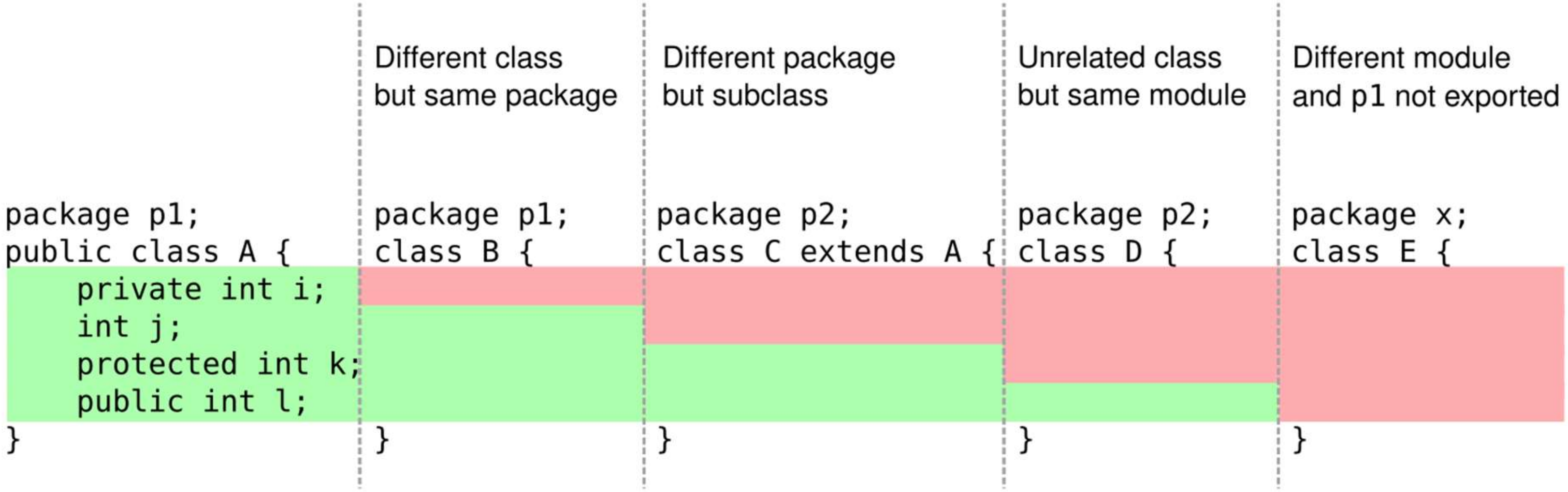
```
This dog is eating
```

Inheritance

- We can also have more complex inheritance.
- We could expand on what we have and add a **parent class** to the Mammal class, the Animal class.
- We could also add a **child class** to Dog, the Labrador and Beagle Class.
- We do not need to explicitly define “grand-parents”, this will be implied.


class Animal
Animal
class Mammal extends Animal
Mammal
class Reptile extends Animal
Reptile
class Dog extends Mammal
Dog
class Cat extends Mammal
Cat
class Labrador extends Dog
Labrador
class Beagle extends Mammal
Beagle

Access Modifiers



```
package p1;
public class A {
    private int i;
    int j;
    protected int k;
    public int l;
}
```

Different class but same package

```
package p1;
class B {
}
```

Different package but subclass

```
package p2;
class C extends A {
}
```

Unrelated class but same module

```
package p2;
class D {
}
```

Different module and p1 not exported

```
package x;
class E {
}
```

🟩 Accessible 🟥 Inaccessible

Inheritance – Object Class

- All classes inherits from the **Object Class** (it is an actual built-in class called **Object**).
- The Object Class has defined some useful functions such as the **toString()** function we discussed previously.

Inheritance – Object Class

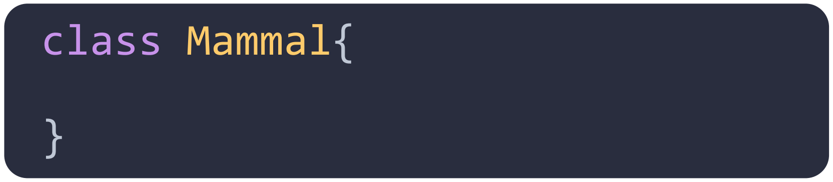
Method	Description
clone()	Creates and returns a copy of this object.
equals(Object obj)	Indicates whether some other object is "equal to" this one.
getClass()	Returns the runtime class of this Object.
hashCode()	Returns a hash code value for the object.
notify()	Wakes up a single thread that is waiting on this object's monitor.
notifyAll()	Wakes up all threads that are waiting on this object's monitor.
toString()	Returns a string representation of the object.
wait()	Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
wait(long timeout)	Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
wait(long timeout, int nanos)	Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Inheritance – Object Class

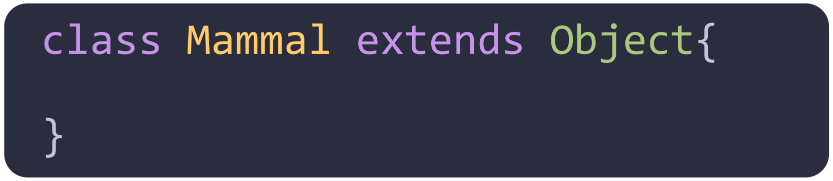
- We discussed that we cannot have multi-inheritance — a child class with more than one parent class.
- Then how can the **Dog** class inherit from the **Object** class?

Inheritance – Object Class

- Whenever we do not declare explicitly inheritance (using the **extends** keyword), Java by default **extends** the class to the **Object** class.
- This is done "invisibly":



```
class Mammal{

}
```

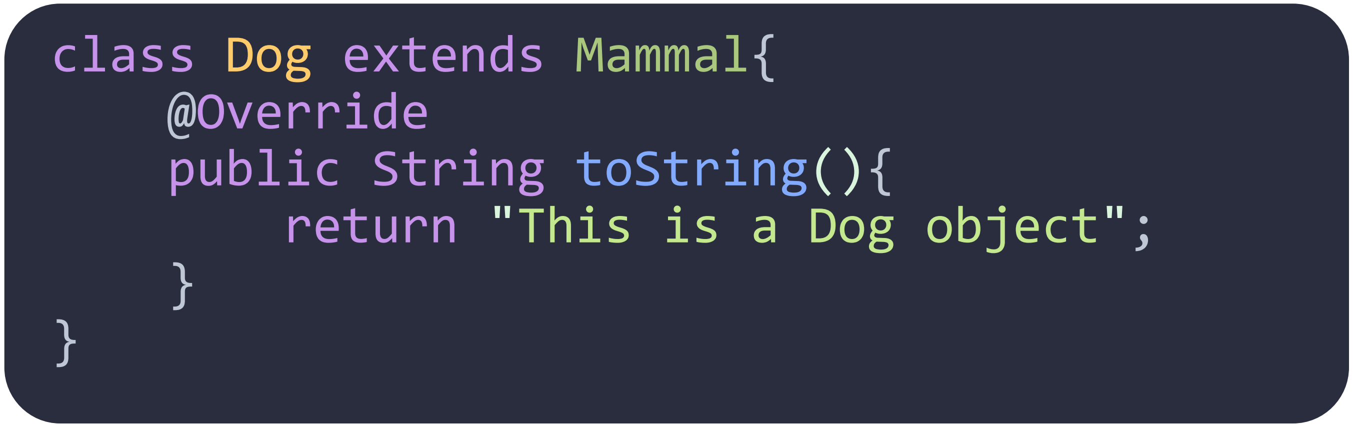


```
class Mammal extends Object{

}
```

Even though we did not declare that **Mammal** inherits from **Object**, Java will implicitly do that for us, so we do not have to do this every single time.

Inheritance – Object Class

- Now, since **Dog** inherits or extends the **Mammal** class then the **Dog** class will also inherit the functions from the **Object** class.



```java
class Dog extends Mammal{
    @Override
    public String toString(){
        return "This is a Dog object";
    }
}
```