Fundamental Concepts of OOP

Encapsulation

OOP's innovation that made the paradigm a solution to the issues of state is its mechanism to construct boundaries wherever you want.

Encapsulation

Encapsulation, when done correctly, makes your system approach a more accurate simulation of the real world.

Encapsulation

The more you encapsulate related data and methods, the more you'll create cohesive classes that have definite and indivisible purpose.

Encapsulation

You should **encapsulate what varies**, meaning, things that always change should be encapsulated deep into the structure of your code.

Which implementation is better?

Consider a system that calculates tax by multiplying amounts with a tax rate, which is the best implementation for tax calculation

- A global variable called `tax_rate`
- Hard code the tax value for every instance of use.
- Implement some function called `calculateTax()` which calculates tax

Encapsulation

Encapsulating what varies helps with maintainability since the changing a isolated data or behavior will have less effect.

Encapsulation

Encapsulation means both attributes and behaviors.

Concrete objects should be given the responsibility of implementing their own behavior.

Inheritance

Inheritance is the concept in which the definition of a class is derived from another class.

An existing class, called the **super class** (also called the **base class** or the **parent class**) passes all visible attributes and methods to a **sub class** (also called the **derived class** or the **child class**).

Inheritance

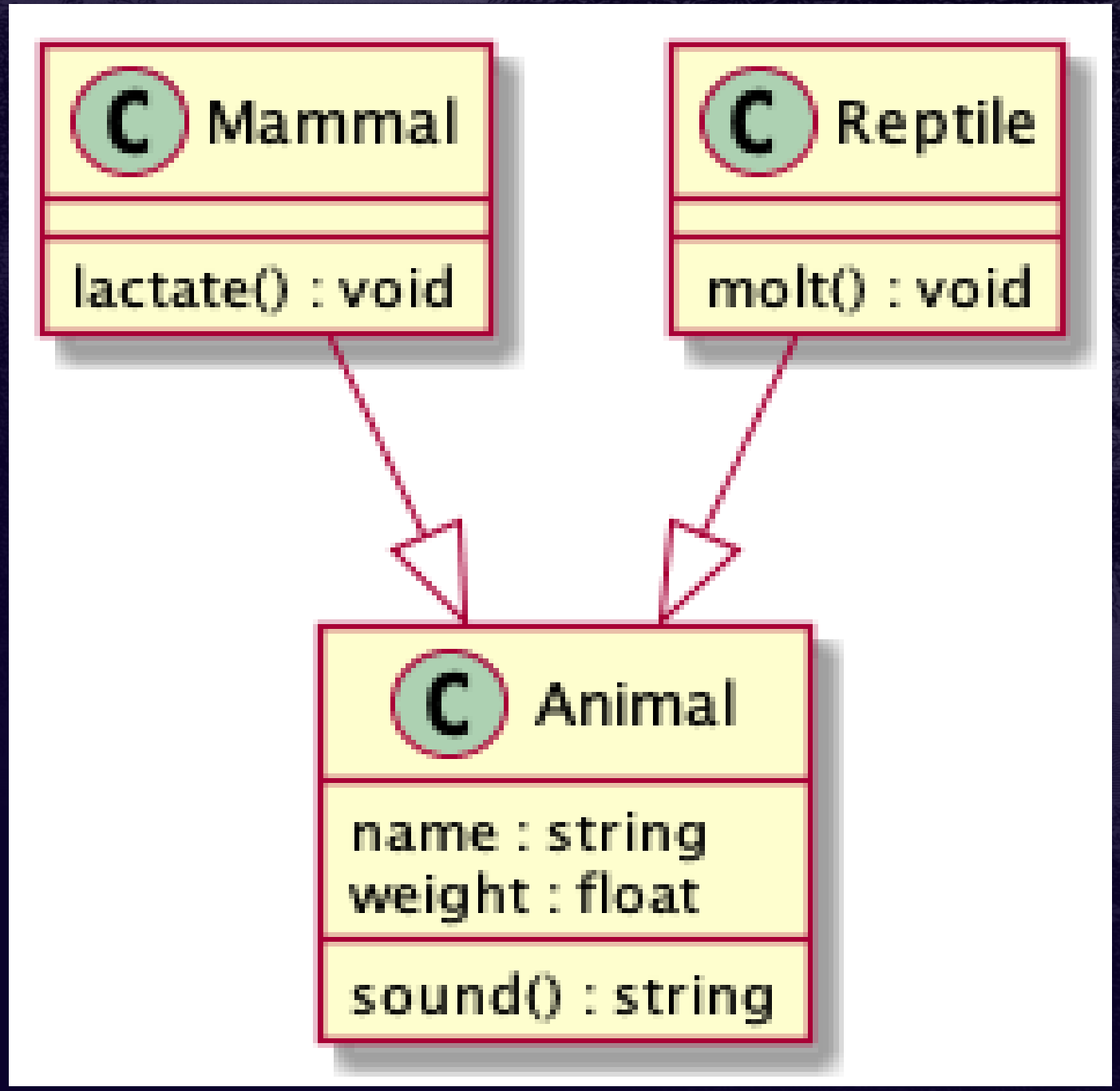
Encapsulation means both attributes and behaviors.

Concrete objects should be given the responsibility of implementing their own behavior.

Inheritance

The concept of inheritance is also a representation of the real world. You use inheritance to represent generalizations and specializations.

A super class is a generalization of a sub class and a sub class is a specialization of a super class.


Mammal
lactate() : void
Reptile
molt() : void
Animal
name : string
weight : float
sound() : string

Inheritance

In this example the supertype animal is a generalization of the subtype mammal.

Inheritance

Although it isn’t shown, Mammal will also have the attributes name and weight and the method sound() since it inherits these from the parent class.

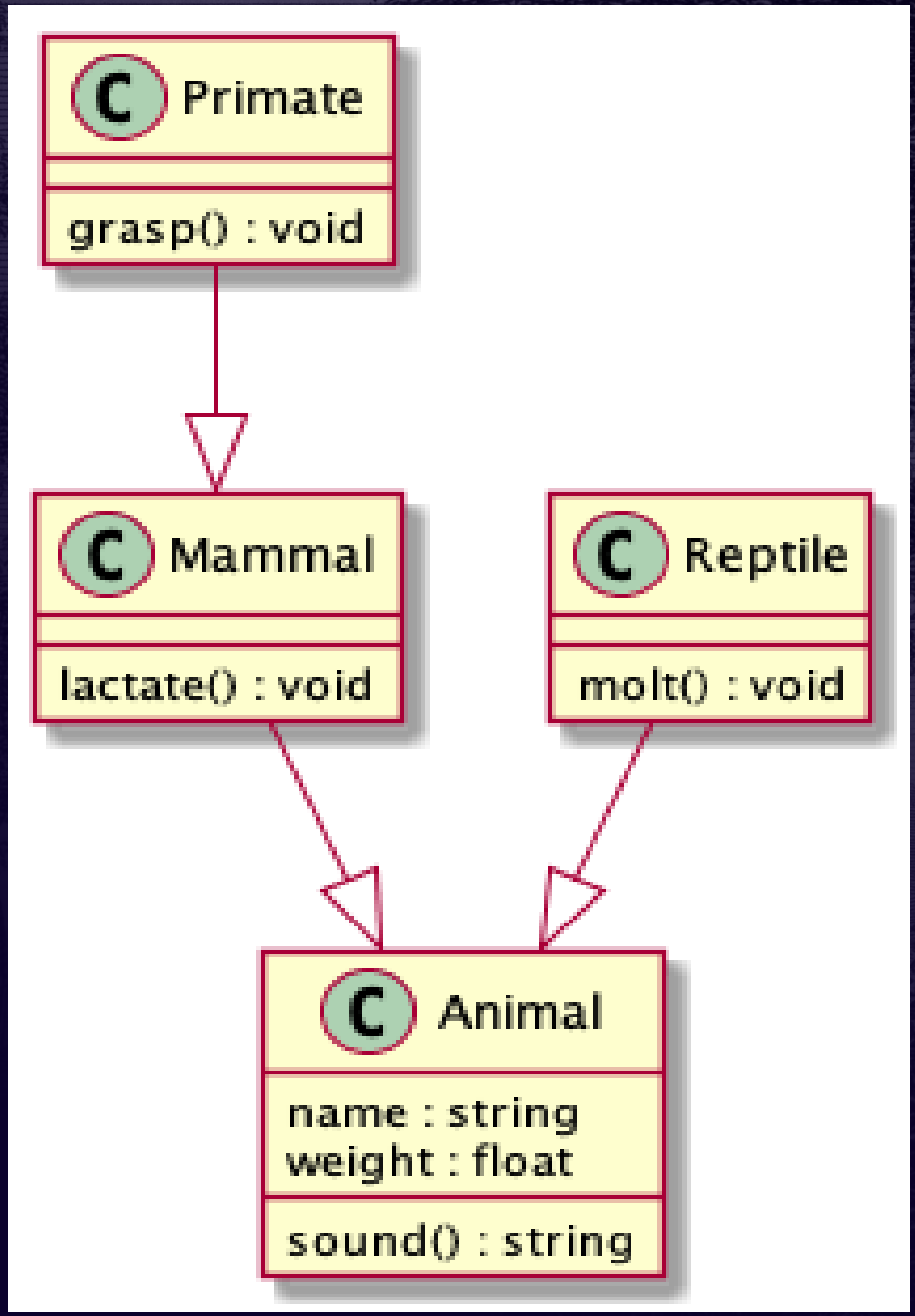
Inheritance

Mammal has a method of its own called lactate() which it doesn share with animal.

Inheritance

A subclass can also be a super class for another class.

This is used to represent specializations of specializations.


Primate
grasp() : void
Mammal
lactate() : void
Reptile
molt() : void
Animal
name : string
weight : float
sound() : string

Polymorphism

Polymorphism literally means *multiple forms*. One of the core philosophy of OOP allows object instances to exist in multiple forms.

What this means code-wise is that the types of object instances can be decided during runtime.

Inheritance

Mammal has a method of its own called lactate() which it doesn’t share with animal.

Compile-time polymorphism

There's also another type of polymorphism that is not necessarily shared by all OOP languages, compile-time polymorphism.

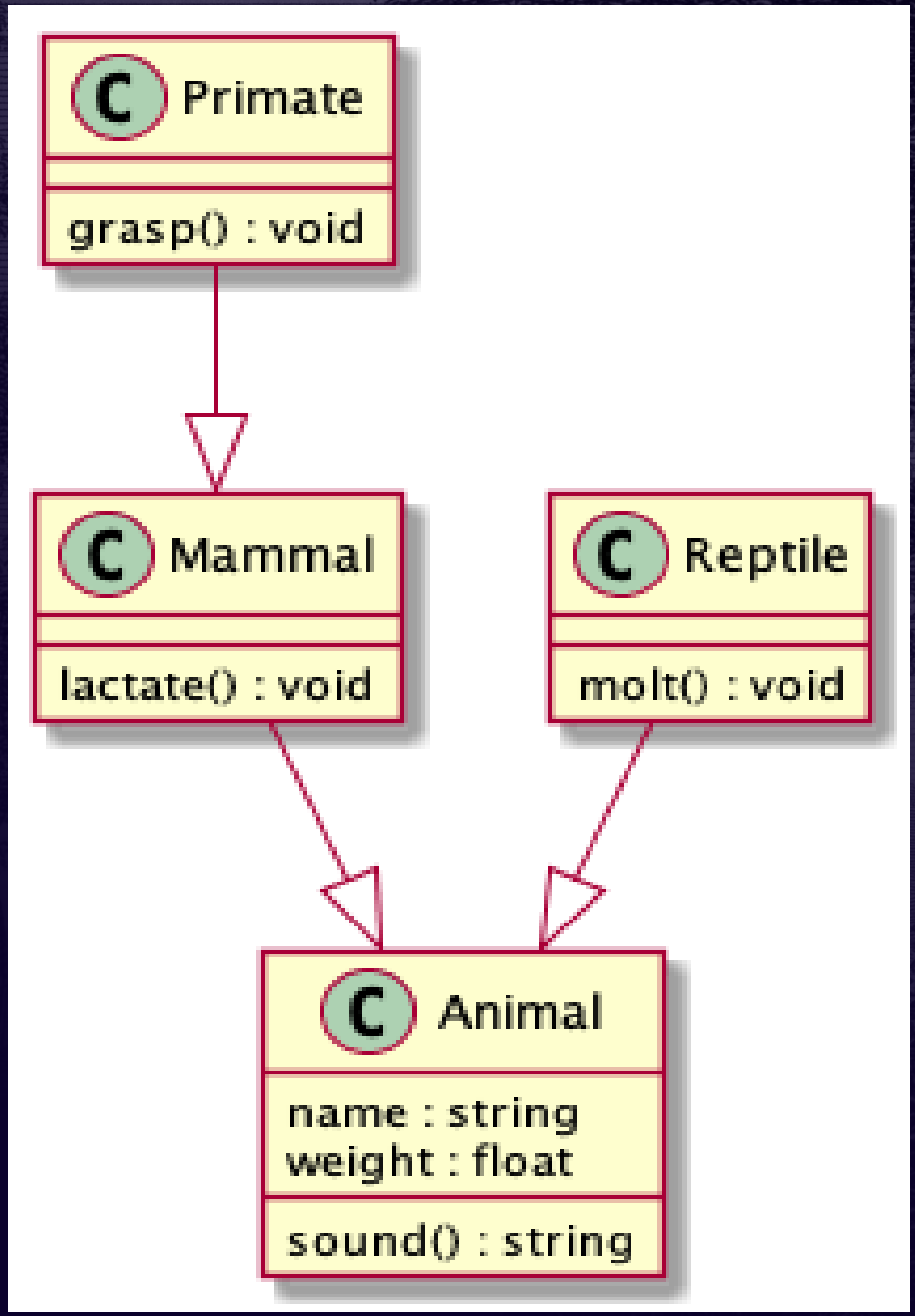
Compile-time polymorphism

This is basically the feature where multiple functions can have the same name as long as they have different parameter type signatures.

This is also known as method overloading.

Run-time polymorphism

Run-time polymorphism is basically achieved using specialization and realization relationships between objects.


Primate
grasp() : void
Mammal
lactate() : void
Reptile
molt() : void
Animal
name : string
weight : float
sound() : string

Run-time polymorphism

An object instantiated to be of type `Primate` is also an instance of an `Animal` because a primate is just a specialization of an animal.

Run-time polymorphism

This reflects how the real world works because a primate is indeed an animal

Run-time polymorphism

On realization relationships like a `Book` and a `BorrowableItem`, the same is also true, because a book is also something that can be borrowed.

Run-time polymorphism

Realization and specialization relationships guarantee that you can interact with a sub type as its super type and you can interact with concrete class as its abstraction.