Classes and Objects

Shirley B. Chu

June 22, 2020

De La Salle University
College of Computer Studies

Procedural vs Object-Oriented Paradigm

Procedural	Object-Oriented
identify the steps for solving the problem	identify the entities in the problem

Procedural vs Object-Oriented Paradigm

Procedural	Object-Oriented
identify the steps for solving the problem	identify the entities in the problem
program the steps in sequential order	model each identity as an object and make them interact like they would in the real world

Procedural vs Object-Oriented Paradigm

Procedural	Object-Oriented
identify the steps for solving the problem	identify the entities in the problem
program the steps in sequential order	model each identity as an object and make them interact like they would in the real world
organized around data and logic	organized around objects and data

What is an **Object**?

- a.k.a. *instance*

What is an Object?

- a.k.a. *instance*
- basic run-time entity of an object-oriented system

What is an Object?

- a.k.a. *instance*
- basic run-time entity of an object-oriented system
- software bundle of related state and behavior

What is an Object?

- a.k.a. *instance*
- basic run-time entity of an object-oriented system
- software bundle of related state and behavior
- e.g. `String` and `Scanner` objects

Where do objects come from?

- Objects are build from blueprints called *classes*.

Where do objects come from?

- Objects are build from blueprints called *classes*.
- A class defines the *attributes* and *behavior* of an object.

What is a class?

- a blueprint that tells how an object is built

What is a class?

- a blueprint that tells how an object is built
- has two components

What is a class?

- a blueprint that tells how an object is built
- has two components
  1. *attributes*
  2. *methods*

What is a class?

- a blueprint that tells how an object is built
- has two components
  1. *attributes*
     - a.k.a. fields, states, properties, characteristics, member variable
  2. *methods*

What is a class?

- a blueprint that tells how an object is built
- has two components
  1. *attributes*
     - a.k.a. fields, states, properties, characteristics, member variable
     - properties the object has or should have
  2. *methods*

What is a class?

- a blueprint that tells how an object is built
- has two components
  1. *attributes*
     - a.k.a. fields, states, properties, characteristics, member variable
     - properties the object has or should have
     - data that should be stored by the object
  2. *methods*

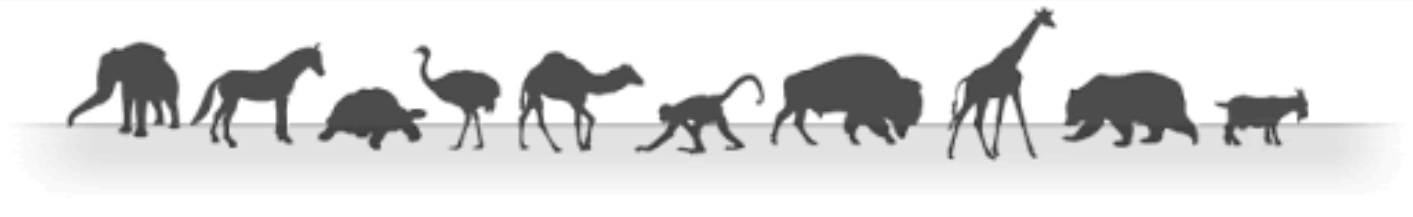
What is a class?

- a blueprint that tells how an object is built
- has two components
  1. *attributes*
     - a.k.a. fields, states, properties, characteristics, member variable
     - properties the object has or should have
     - data that should be stored by the object
  2. *methods*
     - a.k.a actions, behavior

What is a class?

- a blueprint that tells how an object is built
- has two components
  1. *attributes*
     - a.k.a. fields, states, properties, characteristics, member variable
     - properties the object has or should have
     - data that should be stored by the object
  2. *methods*
     - a.k.a actions, behavior
     - actions or behavior that the object is capable of

Try this



Create a class for your favorite animal.

Identify at least two properties and at least one method.

Class Diagram

A *class diagram* in the Unified Modeling Language (UML) is a type of static structure diagram.

It describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

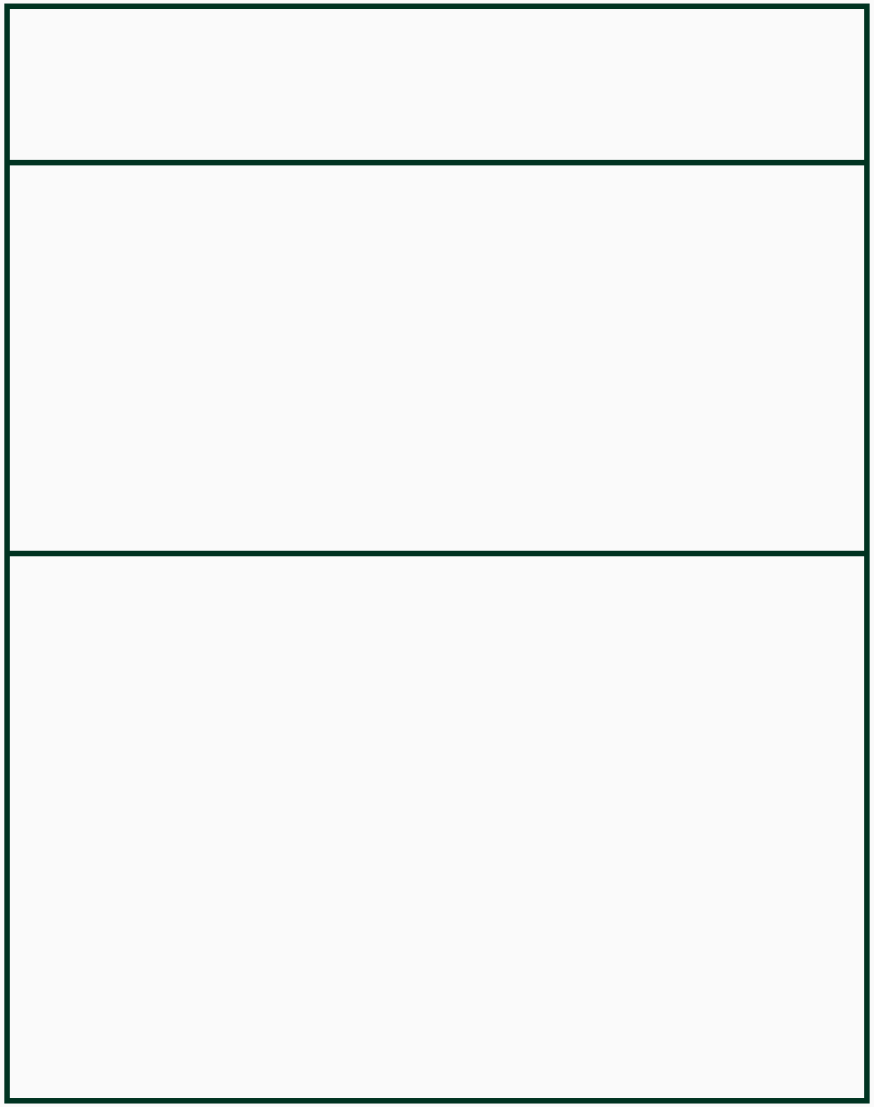
Class Diagram

A *class diagram* in the Unified Modeling Language (UML) is a type of static structure diagram.

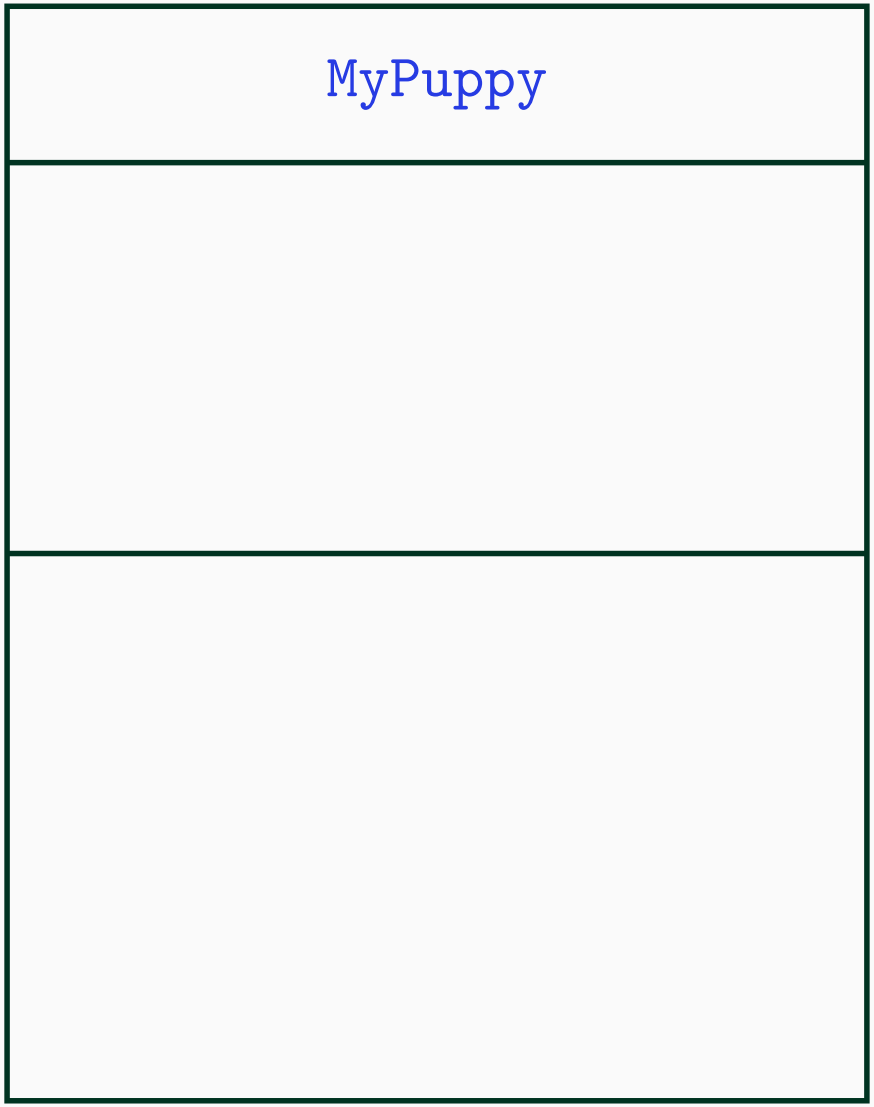
It describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

By looking at the UML class diagram, a person should have an idea on what different classes are available and how they are structured to work with one another.

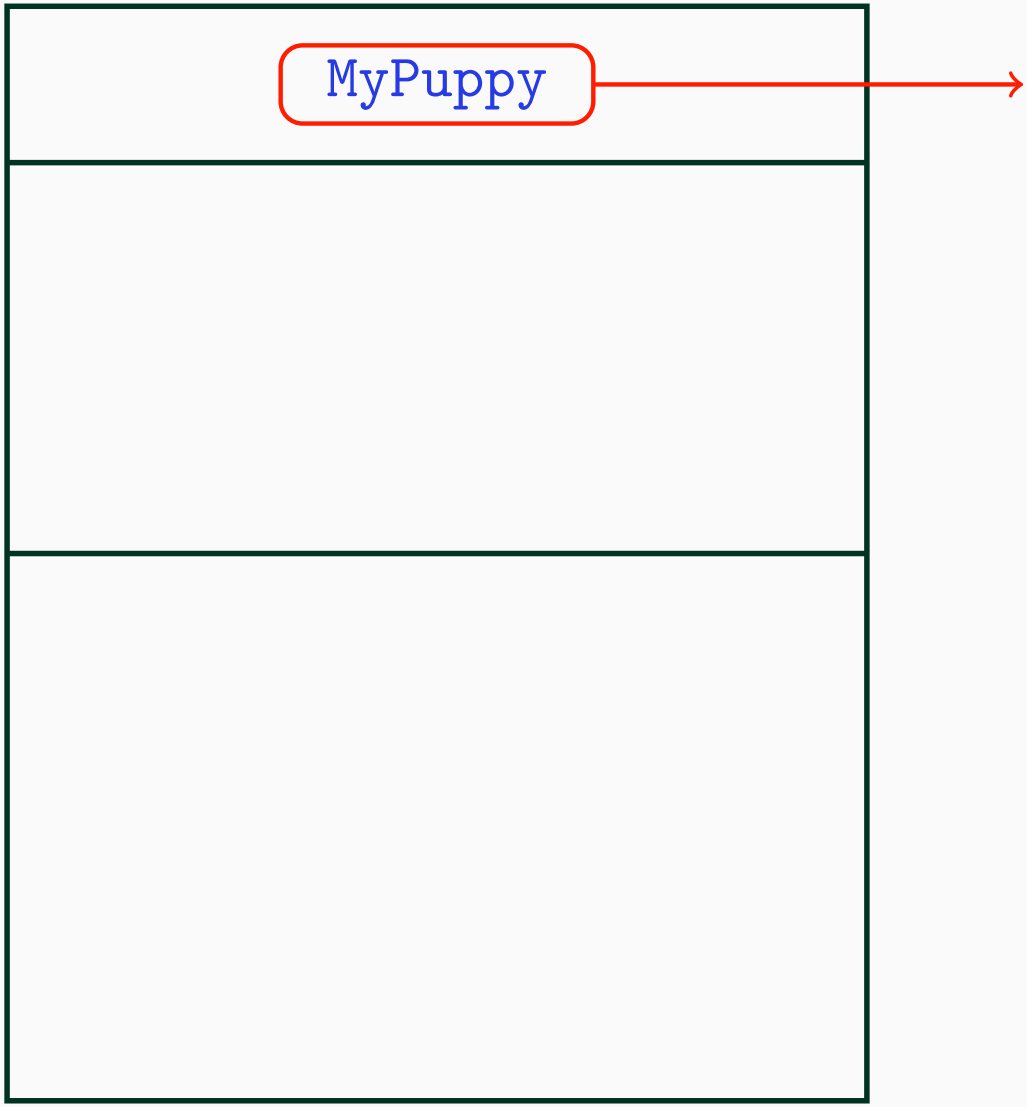
Class Diagram



Class Diagram

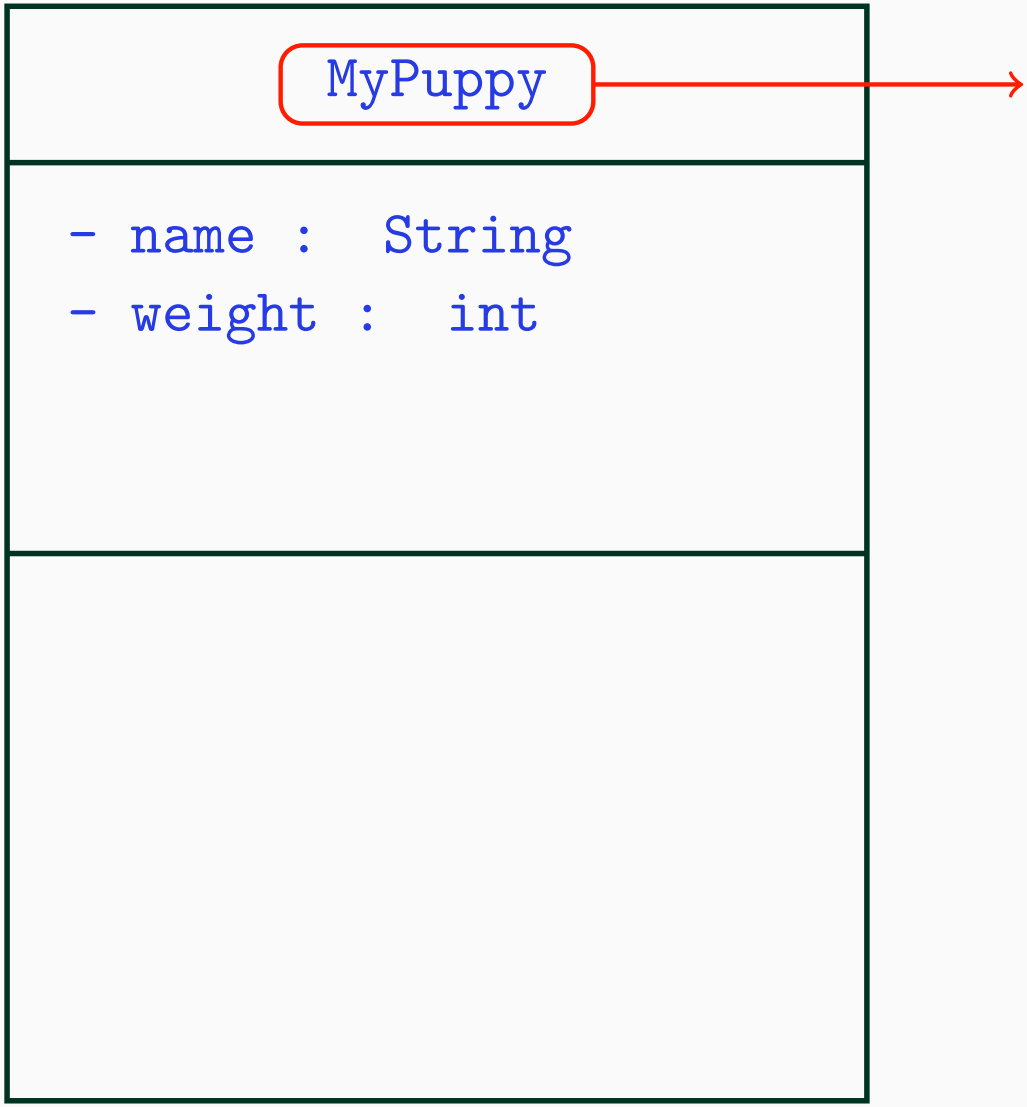


Class Diagram



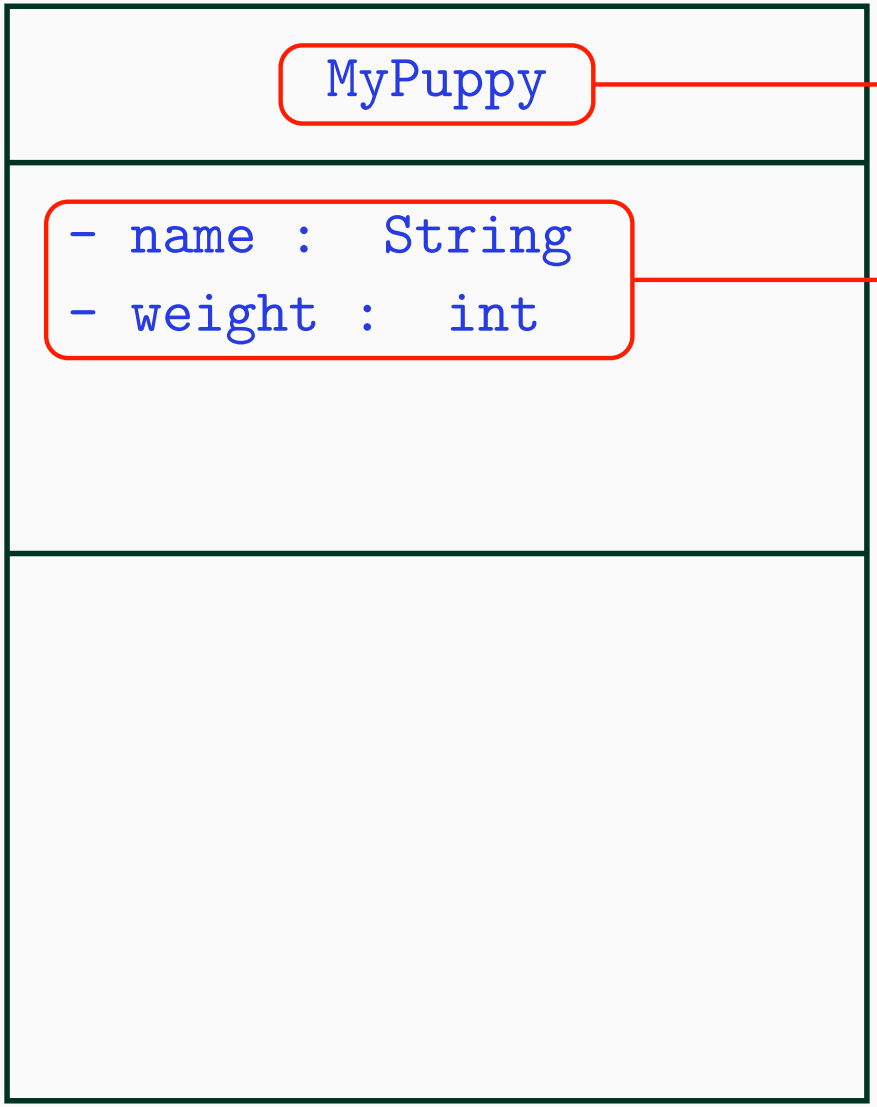
Class name: starts with an uppercase, camel-cased, no space

Class Diagram



Class name: starts with an uppercase, camel-cased, no space

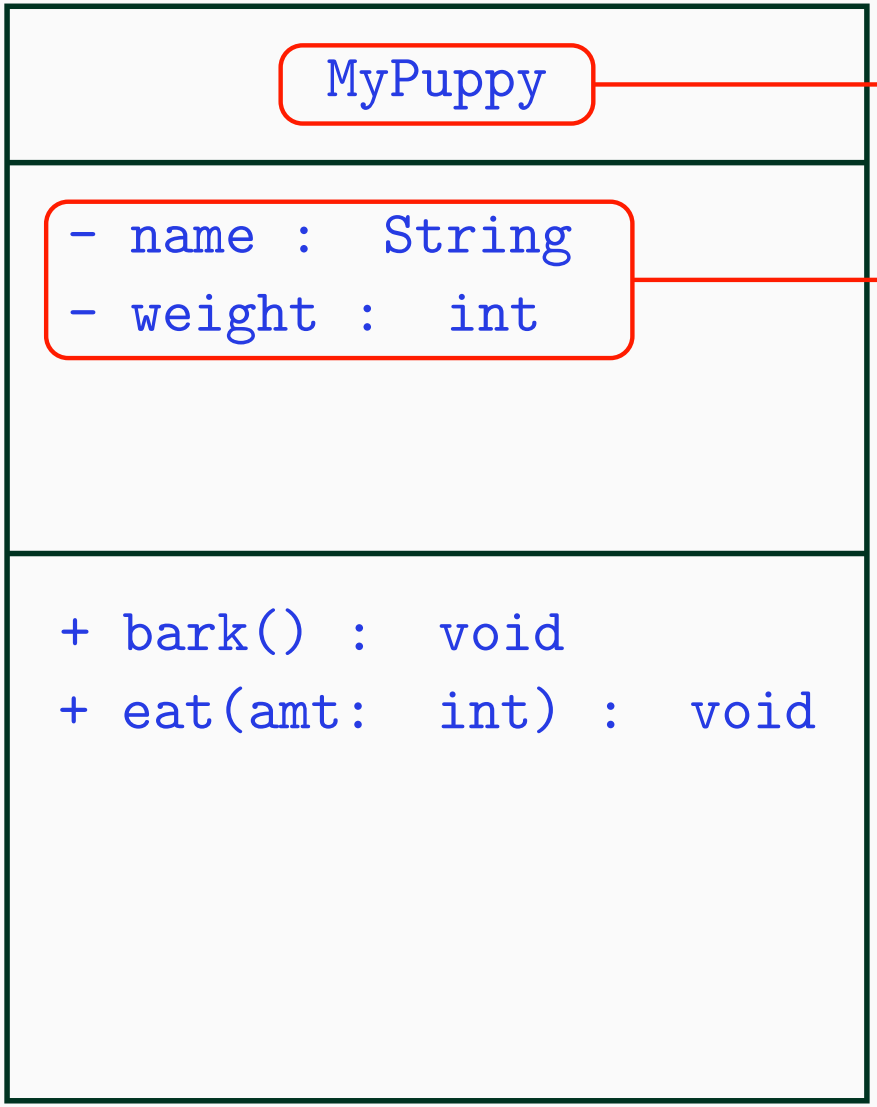
Class Diagram



Class name: starts with an uppercase, camel-cased, no space

Attributes: indicate access modifier, attribute name, type

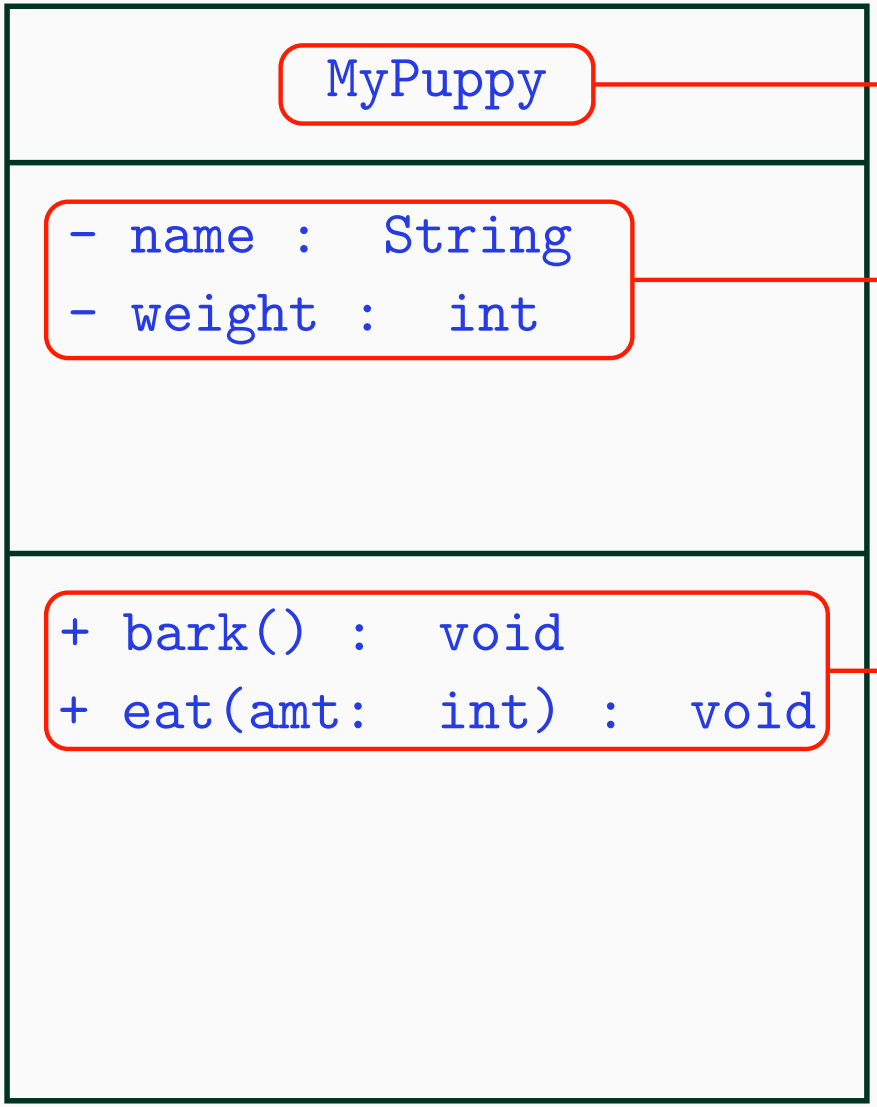
Class Diagram



Class name: starts with an uppercase, camel-cased, no space

Attributes: indicate access modifier, attribute name, type

Class Diagram



Class name: starts with an uppercase, camel-cased, no space

Attributes: indicate access modifier, attribute name, type

Behavior: indicate access modifier, method name, parameter list, return type

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

- Class

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

- Class
  - `public` (`+`) : the class is visible to all classes everywhere

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

- Class
  - `public` (+) : the class is visible to all classes everywhere
  - *no modifier* (∼) : a.k.a. package-private; the class is visible only within its own package

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

- Class
  - `public` (+) : the class is visible to all classes everywhere
  - *no modifier* (∼) : a.k.a. package-private; the class is visible only within its own package
- Members

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

- Class
  - `public` (+) : the class is visible to all classes everywhere
  - *no modifier* (∼) : a.k.a. package-private; the class is visible only within its own package
- Members
  - `public` (+) : the member is visible to all classes everywhere

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

- Class
  - `public` (+) : the class is visible to all classes everywhere
  - *no modifier* (∼) : a.k.a. package-private; the class is visible only within its own package
- Members
  - `public` (+) : the member is visible to all classes everywhere
  - *no modifier* (∼) : a.k.a. package-private; the member is visible only within its own package

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

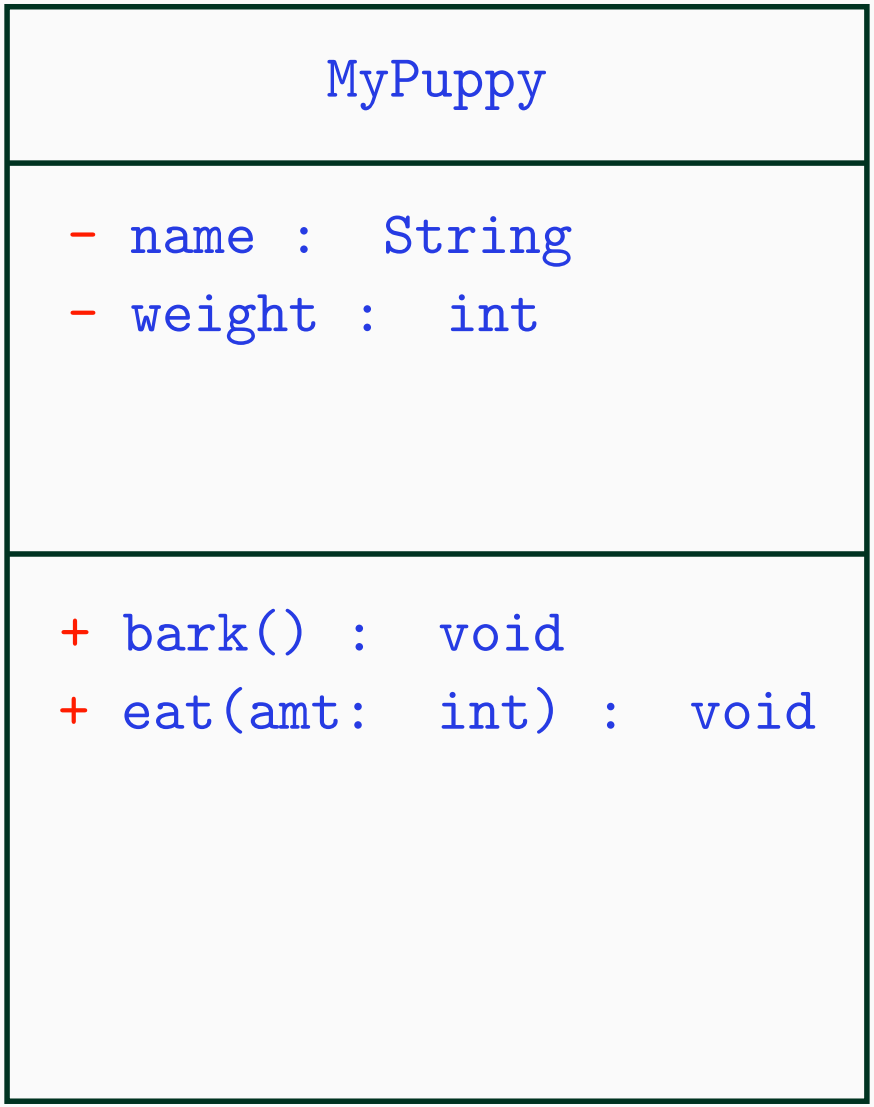
- Class
  - `public` (`+`) : the class is visible to all classes everywhere
  - *no modifier* (~) : a.k.a. package-private; the class is visible only within its own package
- Members
  - `public` (`+`) : the member is visible to all classes everywhere
  - *no modifier* (~) : a.k.a. package-private; the member is visible only within its own package
  - `private` (`-`) : the member can only be accessed within its own class.

Access Modifier

Access modifiers are used to control the *visibility* of a class, or an attribute, or a method.

- Class
  - `public` (+) : the class is visible to all classes everywhere
  - *no modifier* (∼) : a.k.a. package-private; the class is visible only within its own package
- Members
  - `public` (+) : the member is visible to all classes everywhere
  - *no modifier* (∼) : a.k.a. package-private; the member is visible only within its own package
  - `private` (-) : the member can only be accessed within its own class.
  - `protected` (#) : the member can be accessed within its own package, by a subclass of its class.

Class Diagram



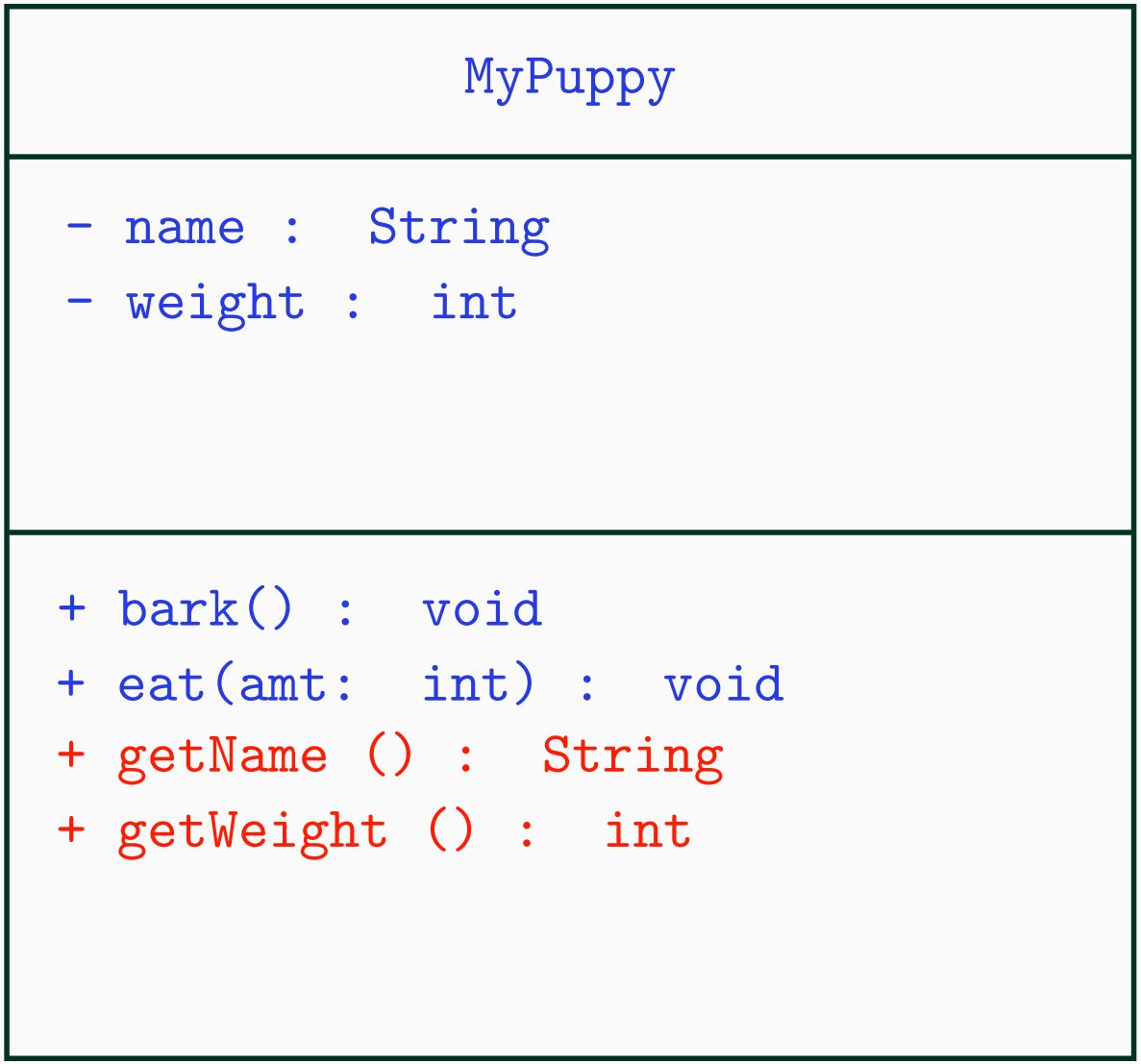
Getters and Setters

- Methods that are used for retrieving (getting) and changing (setting) the values of private attributes.

Getters and Setters

- Methods that are used for retrieving (getting) and changing (setting) the values of private attributes.
- Allows you to restrict which properties could be accessed / changed, and how these are done.

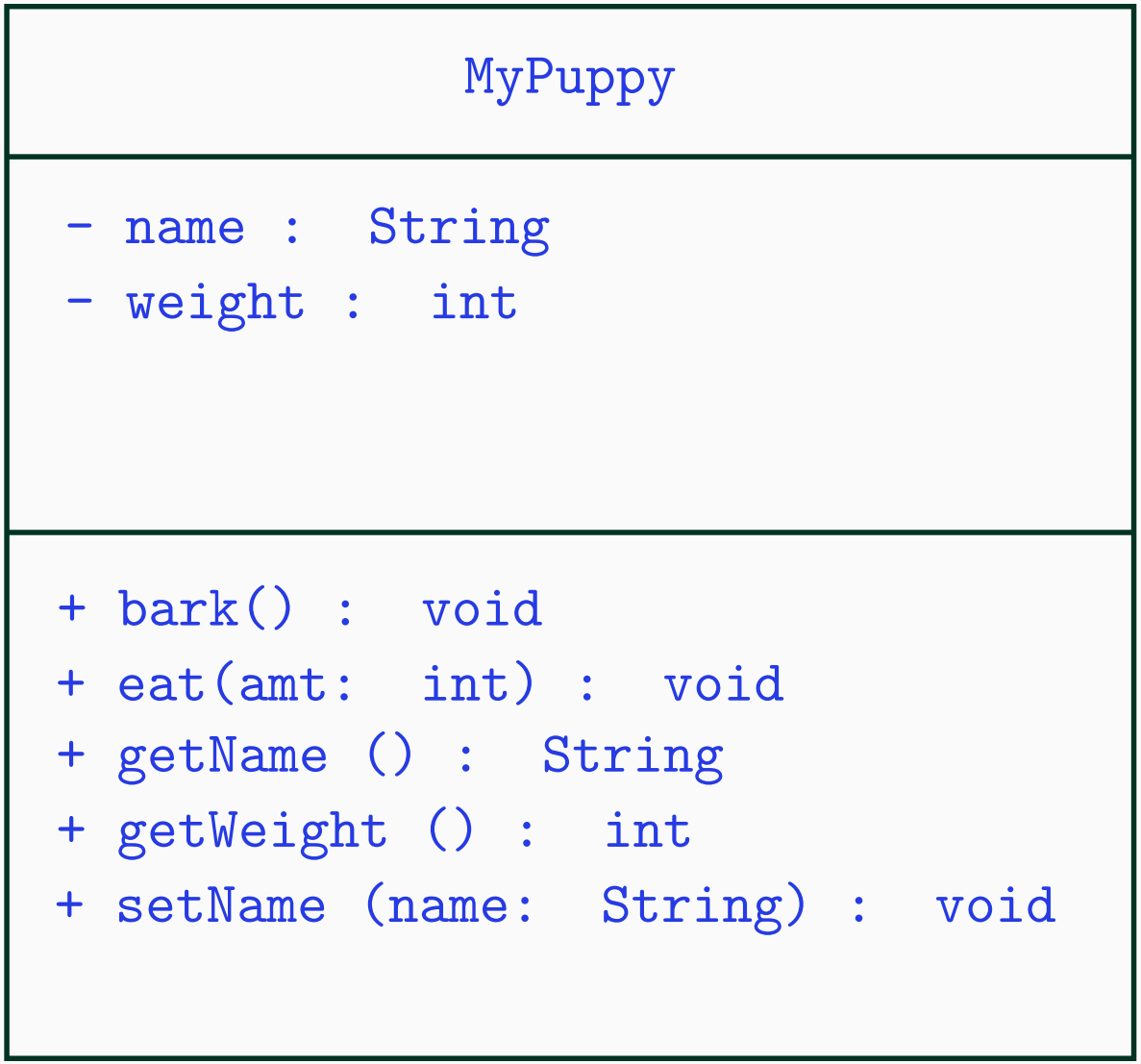
Class Diagram



Class Diagram

MyPuppy
- name : String - weight : int
+ bark() : void + eat(amt: int) : void + getName () : String + getWeight () : int + setName (name: String) : void + setWeight (wt: int): void

Class Diagram



Constructor

- A component of the class that tells how an object should be built.

Constructor

- A component of the class that tells how an object should be built.
- Typically used to initialize the values of an object once it is instantiated.

Constructor

- A component of the class that tells how an object should be built.
- Typically used to initialize the values of an object once it is instantiated.
- A special method with the same name as the class name, and has no return type.

Constructor

- A component of the class that tells how an object should be built.
- Typically used to initialize the values of an object once it is instantiated.
- A special method with the same name as the class name, and has no return type.
- Declared `public`, most of the time. .

Constructors

MyPuppy
- name : String - weight : int
+ MyPuppy(n : String, w: int) + bark() : void + eat(amt: int) : void + getName () : String + getWeight () : int + setName (name: String) : void

Constructors

- The name of the constructor must be the same as the class name.

Constructors

- The name of the constructor must be the same as the class name.
- Constructors have no explicit return type.

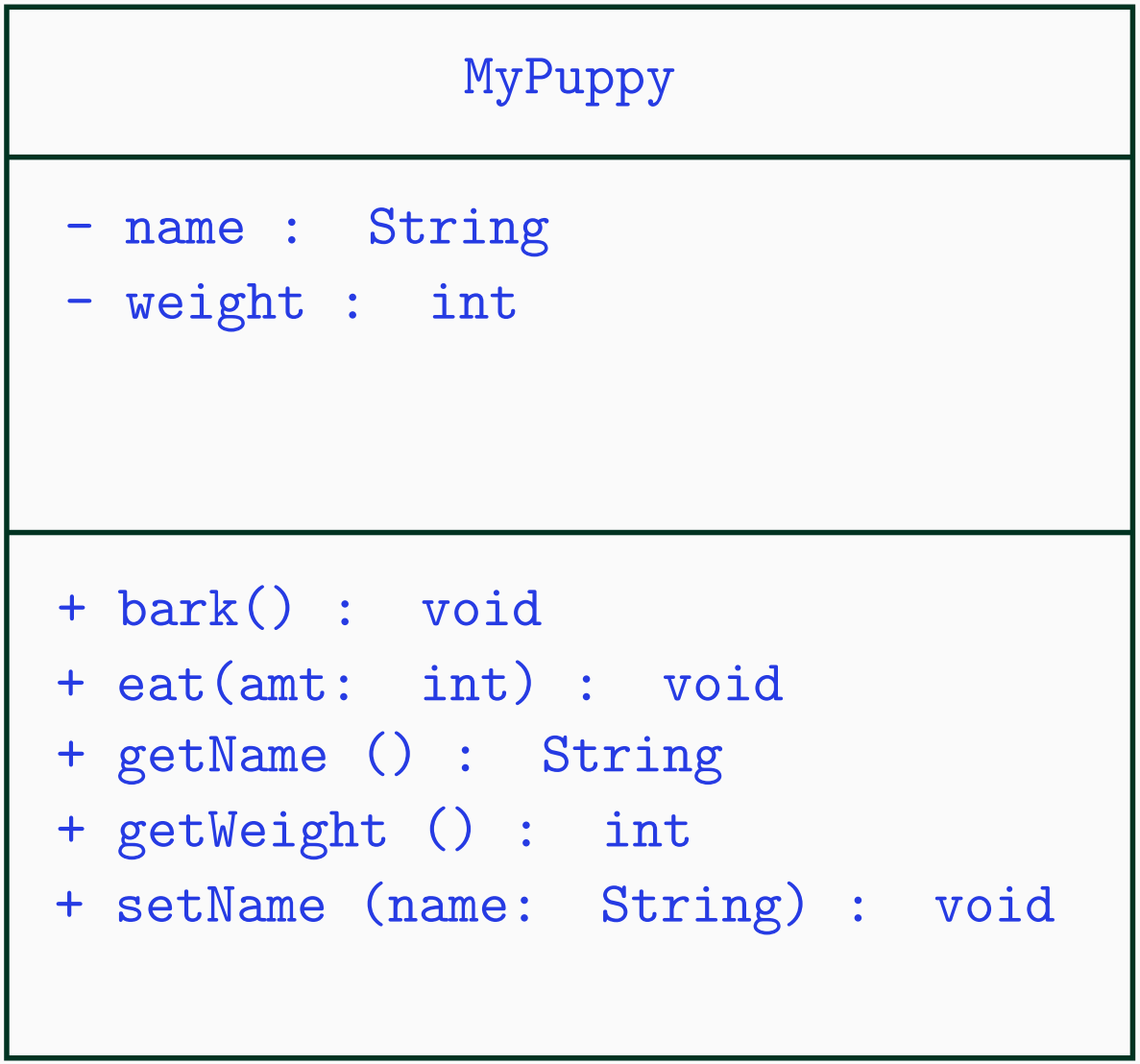
Constructors

- The name of the constructor must be the same as the class name.
- Constructors have no explicit return type.
- Constructors can only be invoked by the `new` operator.

Constructors

- The name of the constructor must be the same as the class name.
- Constructors have no explicit return type.
- Constructors can only be invoked by the `new` operator.
- Every class defines its own constructor/s.

Default Constructor



Default constructor is a constructor with no parameter.

It initializes the member variables to their default values.

This is automatically generated by the compiler if no constructors have been defined.

Constructor Overloading

- provides different ways of constructing the object

Constructor Overloading

- provides different ways of constructing the object
- differentiated by their *parameter lists*, in terms of number, order and type

Constructor Overloading

MyPuppy
- name : String - weight : int
+ MyPuppy() + MyPuppy(n : String, w: int) + MyPuppy(name : String) + bark() : void + eat(amt: int) : void + getName () : String + getWeight () : int + setName (name: String) : void

Design the Date class

A date is composed of integers representing month, day, and year. When a `Date` object is created, its default date is set to July 1, 2020. However, a `Date` object can also be created given a specified date, or given month and day.

When updating the date value of the object, the new date value must be a valid date.

Do include a method to display the date.

OO Concepts

encapsulation refers to the bundling of data with the methods that operate on these data into a single unit.

information hiding refers to hiding the internal representation or state of an object

abstraction refers to hiding certain details (representation, implementation details) and showing only essential information to the user.

☺ **Thank you!** ☺