CSC207H Lecture 2

Sadia Sharmin

Sep 13, 2016

UML

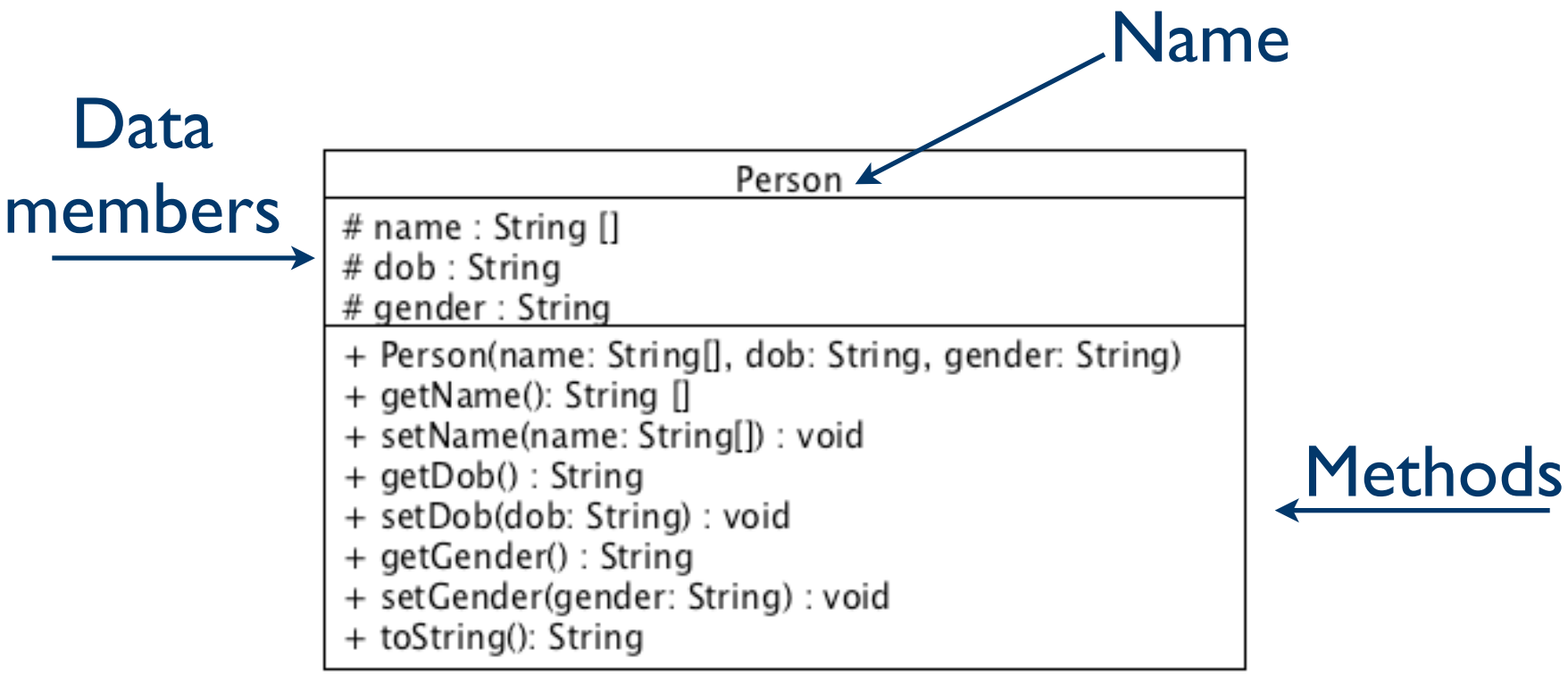
Unified Modeling Language (UML) allows us to express the design of a program before writing any code.

It is language-independent.

An extremely expressive language.

We'll use only a small part of the language, Class Diagrams, to represent basic OO design.

Example: Class Person



Notation

Data members:

```
name: type
```

Methods:

```
methodName(param1: type1, param2: type2,... ): returnType
```

Visibility:

- \- private
- \+ public
- \# protected
- ~ package

Static: <u>underline</u>

Constructors

A constructor has:

- the same name as the class
- no return type (not even `void`)

A class can have multiple constructors, as long as their signatures are different.

If you define no constructors, the compiler supplies one with no parameters and no body.

If you define any constructor for a class, the compiler will no longer supply the default constructor.

`this`

`this` is an instance variable that you get without declaring it.

It's like `self` in Python.

Its value is the address of the object whose method has been called.

Defining methods

- A method must have a return type declared. Use `void` if nothing is returned.
- The form of a return statement:
  `return` *expression;*

  If the expression is omitted or if the end of the method is reached without executing a return statement, nothing is returned.
- Must specify the accessibility. For now:
  `public` – callable from anywhere
  `private` – callable only from this class
- Variables declared in a method are local to that method.

Parameters

When passing an argument to a method, you pass what's in the variable's box:

- For class types, you are passing a reference. (Like in Python.)
- For primitive types, you are passing a value. (Python can't do anything like this.)

This has important implications!

You must be aware of whether you are passing a primitive or object.

Instance Variables and Accessibility

If an instance variable is private, how can client code use it?

Why not make everything public — so much easier!

Encapsulation

Think of your class as providing an abstraction, or a service.

- We provide access to information through a well-defined interface: the public methods of the class.
- We hide the implementation details.

What is the advantage of this “encapsulation”?

Conventions

Make all non-final instance variables either:

- *private*: accessible only within the class, or
- *protected*: accessible only within the package.

When desired, give outside access using "getter" and "setter" methods.

[A *final* variable cannot change value; it is a constant.]

Access Modifiers

Classes can be declared public or package-private.

Members of classes can be declared public, protected, package-protected, or private.

Modifier	Class	Package	Subclass	World
`public`	Yes	Yes	Yes	Yes
`protected`	Yes	Yes	Yes	No
`default (package private)`	Yes	Yes	No	No
`private`	Yes	No	No	No