

CS1020: DATA STRUCTURES AND ALGORITHMS I

Tutorial 2 – OO Design and Programming

(Week 4, starting 1 February 2016)

1. Class Design

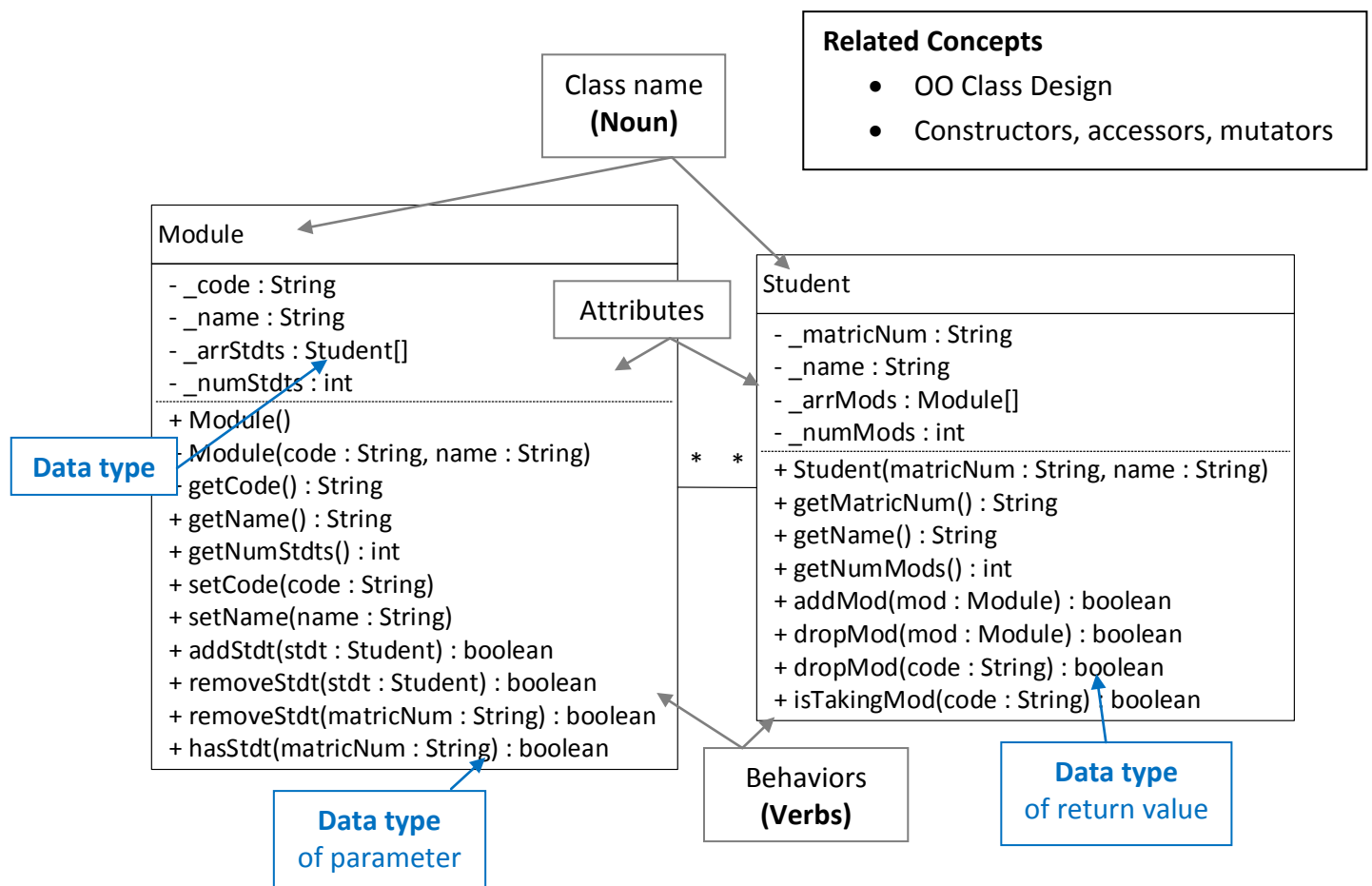
Design a **simple** module registration system for SoC to manage students joining and leaving modules. List two **classes** you think are necessary for this system. For each class, specify its **attributes** and **behaviors**, and provide at least one **constructor**, one **accessor** and one **mutator**. Refer to the lecture notes for these concepts if you are not familiar with them. As this is a design question, implementation is not required.

The following webpage provides a brief introduction to UML, a design language suited for the OO Model. The UML Class Diagram will prove useful for you to express your answer in a drawing that many others understand.

<https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>

Possible answer

First, **identify** the *classes* involved, their *attributes* and *behaviors*. Attributes usually translate into member variables (fields), behaviors into methods. Also think of what *constructors*, *mutators* and *accessors* are necessary. Providing more mutators/accessors may speed up development, but expose data unnecessarily.



2. Object-Oriented Programming

You might have heard the nursery rhyme “Old McDonald”. Its lyrics teach kids about the sounds animals make. You may recall the part of the lyrics about the Cow:

Old McDonald had a farm, E-I-E-I-O
And on his farm he had a Cow, E-I-E-I-O
With a **Moo Moo** here and a **Moo Moo** there...

The lyrics can be generalized for different animals, each having a different *name* and **sound**. With knowledge of object-oriented programming, you want to demonstrate that it is possible to write a program that *sings* “Old McDonald”. To show that your program works, add the following (*animal*, **sound**) to the list: (Cow, Moo), (Dog, Woof), and (Duck, Quack).

Use the following template below to solve your problem:

```
public class OldMcDonald {
    private Animal[] _farm; // Old McDonald had a farm (still has now)
    private static final int SIZE = 3; // Fixed farm size

    public OldMcDonald() {
        /* TODO: Create your farm, an Animal array */
    }

    public void sing() {
        for (int i = 0; i < SIZE; i++)
            System.out.println("Old McDonald had a farm, E-I-E-I-O"+
                /* TODO: Add the rest of the lyrics here... */
            );
    }

    public static void main(String[] args) {
        (new OldMcDonald()).sing();
    }
}

class Animal {
    /* TODO: Implement data and functionality of an Animal here */
}
```

Answers

Animal has 2 attributes: name and sound. These are modelled as String instance variables (fields) – EACH Animal object has references to two Strings, containing the name and sound. The data is **encapsulated** by declaring them with the **private** access modifier. This means, outside the animal class, we *cannot directly access or modify* any Animal object’s fields.

getName() and getSound() are **instance methods** serving as accessors. As we do not need to modify an animal’s attributes, there are no mutators here. The method getName() returns the value of the `_name` instance variable belonging to the Animal object pointed to by `this`. Therefore, invoking getName() on different Animal objects may return references to different names.

```

class Animal {
    private String _name; // e.g. Cow
    private String _sound; // e.g. Moo
    public Animal(String name, String sound) {
        _name = name;
        _sound = sound;
    }
    public String getName() {
        return _name;
    }
    public String getSound() {
        return _sound;
    }
}

```

Related Concepts

- Class vs object
- Access modifiers, static, final
- Member variables & encapsulation

A constant has the keywords `static final`: (1) `static` means there is only 1 copy of `SIZE` belonging to the `OldMcDonald` class rather than a different copy for each object; (2) `final` means the variable must be initialized, and cannot be subsequently assigned a value.

```

public class OldMcDonald {
    private Animal[] _farm; // Old McDonald had a farm (still has now)
    private static final int SIZE = 3; // Fixed farm size

    public OldMcDonald() {
        _farm = new Animal [SIZE];
        _farm[0] = new Animal("Cow", "Moo");
        _farm[1] = new Animal("Dog", "Woof");
        _farm[2] = new Animal("Duck", "Quack");
    }

    public void sing() {
        for (int i = 0; i < SIZE; i++)
            System.out.println("Old McDonald had a farm, E-I-E-I-O\n" +
                "And on his farm he had a " + _farm[i].getName() +
                ", E-I-E-I-O\n" + "With a " + _farm[i].getSound() + " " +
                _farm[i].getSound() + " here and a " + _farm[i].getSound() +
                " " + _farm[i].getSound() + " there...\n"
            );
    }
}
...

```

`_farm[2].getName()` is equivalent to `this._farm[2].getName()`. First, `this._farm` refers to the current `OldMcDonald` object's `_farm` field, which points to an `Animal` array of size 3. Therefore, `_farm[2]` moves to the array, two elements away from the start. That element is another reference, to an `Animal` object. (`_farm[2].`) moves to that `Animal` object (Duck).

We then read the name of the `Animal` object (Duck) using the `getName()` instance method. Remember, that method does not return a `String` object itself, but a reference to the `String` "Duck".

As we have mentioned, **the first 100 times** you encounter some OOP code, **draw out what the code does in memory...**! Don't view code as mere text.

3. Members

There are 4 types of members: instance variables, class variables, instance methods, class methods.

(a) Identify where each of them appears in the following code fragment.

```
class TutorialGroup {
    private String _groupNum;
    private static int _totalGroups;

    public TutorialGroup(String groupNum) {
        _groupNum = groupNum;
        _totalGroups++;
    }

    public String toString() {
        return "Tutorial " + _groupNum + " out of " + _totalGroups
            + " groups";
    }

    public static int getTotalGroups() {
        return _totalGroups;
    }
}
```

(b) Is it possible, within (1), to **directly access** (2) **without** specifying any instance?

(1)	(2)	Yes and what is accessed / No and why not
Instance method	Instance variable	
Instance method	Class variable	
Class method	Instance variable	
Class method	Class variable	

(c) In (a), `_totalGroups` was not explicitly initialized. If not explicitly done so, are class variables initialized? How about instance variables, local variables, array elements?

Answers

Note that “instance variable” here refers to the “instance attribute”, and “class variable” here refers to the “class attribute”. Attributes are also known as “data members” and “fields” (on the API documentation).

(a)

Type	Example
Instance variable	<code>private String _groupNum</code>
Class variable	<code>private static int _totalGroups</code>
Instance method	<code>public String toString()</code>
Class method	<code>public static int getTotalGroups()</code>

Related Concepts

- Only 1 copy of class members
- `this` points to current instance
- Default initialization

(b)

(1)	(2)	Yes & what is accessed / No & why not
Instance method	Instance variable	Yes, refers to instance variable of <code>this</code> object. e.g. within <code>t1.toString()</code> , <code>t1._groupNum</code> instance variable is accessed.
Instance method	Class variable	Yes, refers to the only copy of the variable, belonging to the class. e.g. within <code>t1.toString()</code> , <code>TutorialGroup._totalGroups</code> class variable is accessed.
Class method	Instance variable	No , not directly (without specifying a particular instance). Which instance are you referring to?
Class method	Class variable	Yes, refers to the only copy of the variable, belonging to the class. e.g. within <code>TutorialGroup.getTotalGroups()</code> , <code>TutorialGroup._totalGroups</code> class variable is accessed.

(c)

Type	Initialized automatically?
Instance variable	Yes. These are initialized to their default values: false for boolean, 0 for char and numbers, null for reference types (including references to arrays).
Class variable	
Local variable	No. We don't know what value a primitive (or address, for a reference-typed) local variable holds when just declared. A compilation error may result, as the variable is not initialized.
Array elements	Yes. (We are referring to elements in an array, not the reference to an array.) After an array object is created, each element is initialized to the default value.

- Practise consistently 😊 -

Explore the Java API
Test out its functions
Code incrementally