2.1 Objects and Classes

Object-oriented programming (OOP) – Objects and Classes

Introduction

Object-oriented programming (OOP) models a given problem/scenario into a set of interacting objects. These objects have some data, also known as properties, and exhibit some behaviors. In Object-Oriented terms, we refer to this data as attributes, and behaviors are referred to as methods. Methods refer to what an object can do and what can be done on an object. The benefit of OOP is that it effectively represents real-world entities. For example, if we take a company, it has employees. If we implement a Human Resource management system using an OOP language, we can consider each employee as an object. Each employee object handles the processing and data management related to that employee. There are multiple employee objects - each corresponding to an individual employee.

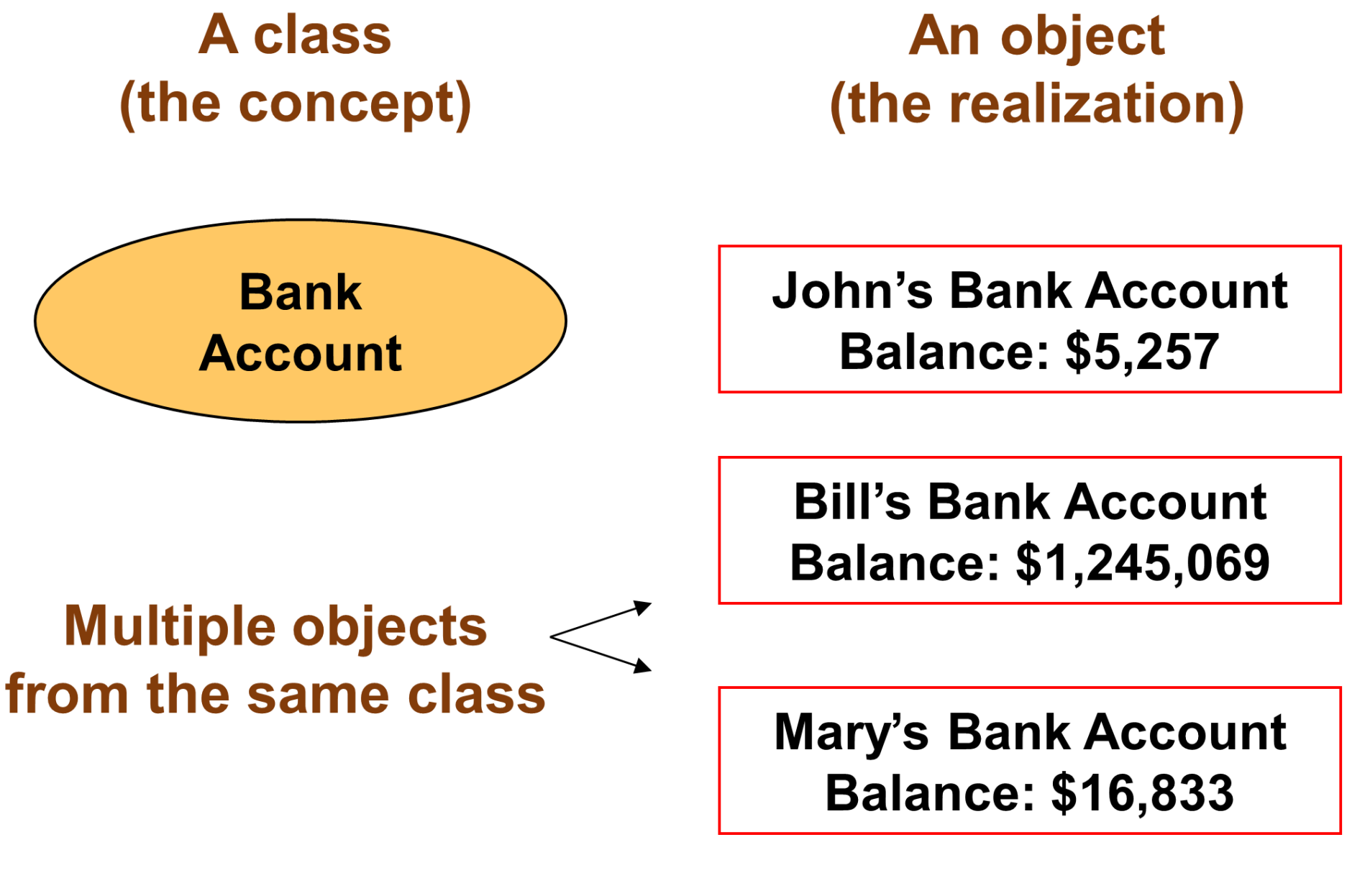
The set of attributes of an object collectively defines its characteristics. The values of these attributes at a given point in time define the object's state. As an example, consider an object referring to a bank account. It has a unique number, the account holder's name, and the current balance. The account number or the account holder's name may remain the same—however, the current balance changes. If we consider the state of a bank account object with these three attributes, the object may be in different states at different time points if any of these attributes change their values during the object's lifespan. For example, if the current balance of an account object changes, that object is said to be in a new state.

In the case of a bank account object, depositing and withdrawing money can be taken as its behavior. Note that the behavior of an object might change its state. For example, when we withdraw money from the bank account, it changes the current balance of the bank account, which indicates a different state of the bank account object.

Classes

Objects of a given type (e.g., bank account) have the same attributes. What differs across objects is the values of these attributes. For example, Peter's bank account may have $500, while Ann's account has $1000. To create different objects of the same type, we use a blueprint called a class. A class specifies what attributes and methods its objects can have. When creating objects from a class, we provide actual values for the attributes defined by the class. That way, we can create any number of objects from the same class. See Figure 1.

To identify the set of classes that we should implement for a given problem, first, we need to look for the nouns in the given problem description. However, even class attributes are nouns. For example, when we say, 'employee has a name,' both employee and name are nouns. However, only the employee should be an object. Thus, we need to have an "Employee" class. Thus, after identifying nouns, we need to see which of these nouns interact with other nouns. For example, consider the statement, 'an employee works in a division'. This shows an interaction between the division and the employee. Thus, both employee and division can be considered as objects.



Constructor

To create objects from a class, we have to call the 'constructor' of the class. The constructor can be considered a special method in the class, called ONLY when an object has to be created. A constructor must have its name as '__init__,' and the first parameter should be the 'self' keyword.

Attributes

A class in Python can have different types of attributes. Attributes defined outside any method but within the class are called the class attributes. In contrast, values for instance attributes should be set for each object created. These attributes are declared inside the constructor and follow the 'self' keyword (e.g., self.age). In a Python class, a variable name with a leading underscore indicates that the attribute should be treated as private to that class. Note that it is always possible to access such attributes from outside the class they have been defined in. However, this is not a good programming practice.

Methods

In Python, methods in a class can be of three types: instance methods, static methods, and class methods. Similar to functions, methods perform some tasks and may return a value or values. However, there are a few key differences between methods and functions. First and foremost, we define methods inside a class, but we can define functions without any class.

An instance method is associated with the objects of the class it belongs to. In other words, an instance method can be called only on an object created from that class. We identify an instance method by passing the 'self' keyword as the method's first argument. An instance method can operate on the arguments passed on to it and the attributes of the associated object. Class methods do not have to know about any object created from the class. However, they need to know about class-level information such as class attributes. We define a class method using the @classmethod decorator and pass the 'cls' keyword as the first argument. A class method can change the state of the class, which will affect all the instances of that class. If a method is inside a class but does not have to access any of the instance or class attributes, this class can be made a static class using the @staticmethod decorator. These methods are used as utility functions.

```python
from datetime import date

class Monster:

    # class attributes
    color = "black"

    def __init__(self, age, name):
        # instance attributes
        self.age = age
        self.name = name

        self._is_innocent = None

    # instance method
    def steal(self, warrior):
        warrior.lose_stick()

    @classmethod
    def create_monster_from_birth_year(cls, year, name):
        return cls(date.today().year - year, name)

    @staticmethod
    def is_adult_Monster(age):
        return age > 18
```

Class

Class attribute

Constructor

Instance attributes

Private attribute by convention

Instance method

Class method

Static method

GET IN TOUCH

University of Moratuwa
Centre for Open & Distance Learning
CODL

011 308 2787/8

011 265 0301 ext. 3850,3851

open@uom.lk