
Object-Oriented Programming
Lesson 04: Some More Concepts in OOP
Capgemini

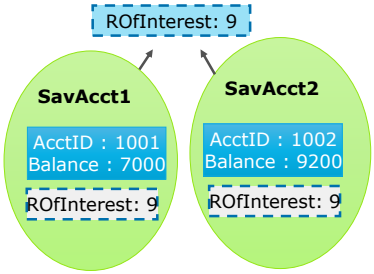
Lesson Objectives

- In this lesson, you will learn about:
  - Static Members
  - Abstract Classes
  - Interfaces
    - Interface versus Abstract Classes
  - Packages

4.1: Static Members

Introduction

- Each object has its own copy of data (instance variable)
- Use Static (Class variable) if data members are to be shared amongst all object instances of the given type
  - Example: Rate of Interest for Savings Account will be the same in the bank for all Accounts, so common copy for data is sufficient. Not so for Account Balances!



Getting into Details – Static Members:

When an object is instantiated against a given class, memory is set aside for storing each of the attributes defined for the class. So each object will have its own copy of the data.
For example: If you were to create three instances of Savings Account, then each Savings Account object maintains a copy of the Balance field assuming Balance is an attribute of the Account Class. What about something like Rate of Interest on the savings account or Minimum balance to be maintained in the account? These values must be the same for all objects of the Account Class since the Bank will have same business rules for all accounts. In such cases, it is not necessary for all objects to store their own copy of this data; infact it could lead to maintenance issues!

Here we need a mechanism whereby data stored once can be shared by all objects of a given class. Static or Class Variables provide this mechanism. Static data is allocated once and shared among all object instances of the same type. So as in the example above, Rate of Interest need not be shared within objects, it can be a static data member which is available for all

Savings Account Objects.

4.1: Static Members

Introduction

- Static Member Functions can be invoked without an object instance.
  - For example: Counting the number of Customer Objects created in the Banking System – this is not specific to one object!
- Example

```
class Customer{
   static int customerCount;
   Customer(){
         customerCount++;
   }
   static int countCustomers(){
         return customerCount;
   }
}
```

Invoking above static function would be like

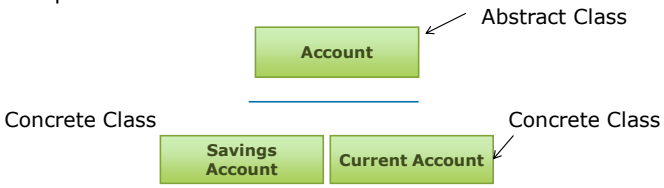
```
Customer.countCustomers();
```

Getting into Details – Static Members:
Static member functions are operations defined within the scope of a class. However, they can be invoked without using an instance. This means that a static function is not invoked on an “object”. However, it is instead invoked on a “class”.
A static member function has access to the private data of a class. It can access “static member variables”, or given a pointer or reference to an object of the class as an argument it can access the “private instance variables” of any object passed as an argument.

4.2: Abstract Class

Concept of Abstract Class

- Abstract class is a Special Type of Base Class.
  - Implementation details are undefined for one or more operations, they are implemented by derived classes.
  - Objects cannot be instantiated against such classes.
- Example: Account class



Abstract class:
Abstract classes are special type of base classes. In addition to normal class members, they have abstract class members. These abstract class members are methods that are declared without an implementation. All classes derived directly from abstract classes must implement all of these abstract methods. Abstract classes can never be instantiated, since the members have no implementations.
For example: One does not really have just an Account, but one has an account of specific type, say an account of savings account type or current account type.

4.2: Abstract Class

Concept of Abstract Class

- Why Abstract Classes?
  - Establish Structure to Class Hierarchies:
    - They capture the commonality amongst hierarchy of classes, at the same time "changing" implementations can be left to derived classes.

Design Heuristic: Base of a class hierarchy should be an abstract class

Abstract class:
Abstract classes sit toward the top of a class hierarchy. They establish structure and meaning to code. They make frameworks easier to build. This is possible because abstract classes have information and behavior common to all derived classes in a framework. The aspects that are specific to different derived classes can be left open for implementation in the derived classes.
For example: One does not really have just an Account, but one has an account of specific type, say an account of savings account type or current account type.

4.2: Abstract Class

Lab

- Abstract Classes and Polymorphism
  - Lab 4.1

4.3: Interface

Concept of Interface

- Interface is a specification of a set of operations that indicates service provided by a class or component.
  - Interfaces have only specifications, no implementations.
  - Implementations are provided by classes that implement the interfaces.
- Why Interfaces?
  - They allow polymorphism – without being restrictive about class hierarchies.
  - They enable Plug and Play architecture

Interface:
Interfaces only specify operations, not implementation of the operations. In that sense, they are like a contract specification, and to fulfill the contract, the corresponding class needs to provide implementations for all operations specified within interface.
Polymorphism and interfaces go hand-in-hand. Interfaces formalize polymorphism. If two objects use same methods, to achieve different, but more or less similar results, then they are polymorphic in nature.
Plug and Play architecture – allowing for easy maintainability and extensibility – are benefits of using interfaces.

4.3: Interface

Abstract Class versus Interface

Abstract Class	Interface
No instantiation	No instantiation
Can have implementations for some operations	Purely specifications, No implementations
Can encapsulate a common behaviour for use by related classes	Can encapsulate a common behaviour for use by unrelated classes
Class derived from Abstract classes can provide implementations for only some operations if needed	Class or component implementing an Interface is bound to provide implementation for ALL operations specified in interface
Widely supported by OO Programming Languages	No support for this concept in some of the earlier OO Programming Languages

Interface versus Abstract Class:
Abstract classes allow you to partially implement your class, whereas Interfaces contain no implementation for any members.
Interfaces are used to define the peripheral abilities of a class. An Abstract Class defines the core identity of a class and there it is used for objects of the same type.
If we add a new method to an Interface, then we have to track down all the implementations of the interface and define implementation for the new method. If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly.

4.4: Packages

Concept of Packages

- A package is a logical grouping unit of related classes and interfaces.
  - Unique identifiers for classes and interfaces are needed in same package.
  - A package maps to directory structure for application development.
- Why Packages?
  - It facilitates maintenance of large systems by partitioning the set of classes into **manageable chunks**.

Packages:
Large complex software systems contain hundreds of classes and interfaces. Managing these classes is a challenging issue. Maintenance becomes easier by placing these model elements (classes and the interfaces) into logical groups called packages.
Classes having same name can belong to different packages. To access them, the class needs to be qualified with the corresponding package name.

4.4: Packages

Lab

- Consolidated Exercise
  - Lab 5.1

Summary

- In this lesson, you have learnt that:
  - Static Members allow sharing of data across different objects of the same class
  - Abstract classes establish structure and meaning to code.
  - Interfaces formalize polymorphism.
  - Packages are a logical grouping mechanism

Answers to Review Question

Question 1: Option 2,3,4

Question 2: True

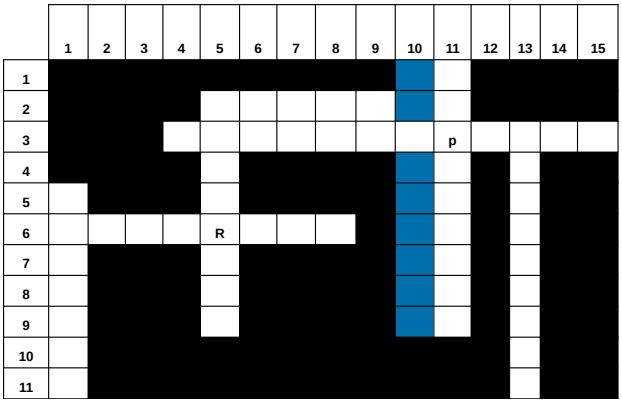
Question 3: Package

Review Question

- Question 1: Choose right options. Abstract classes
  - Option 1: Members have implementations
  - Option 2: Sit toward the top of a class hierarchy
  - Option 3: Establish structure and meaning to code
  - Option 4: Have information and behavior common to all derived classes
  - Option 5: Can be instantiated

- Question 2: Abstract classes allow you to partially implement your class.
  - True / False
- Question 3: ___ provides mechanism to logically group classes.

Review Question: Crossword



Clues. 11 rows (1-11) , 15 Cols (1-15), (Row,Col) Combination

Across:
2-5 : Packages are logical grouping of ____ and interface (5)
3-4: Interfaces formalize this (12)
6-1: _____ Members have no implementations (8)

Below:
1-11 : Derived class must _____ all abstract methods or declare itself abstract (9)
3-13: Collection of operations that are used to specify a service of a class or a component (9)
2-5: “Opposite” of abstract: Classes from which instances can be instantiated (8)
5-1: These help in clubbing together related classes and interfaces (7)