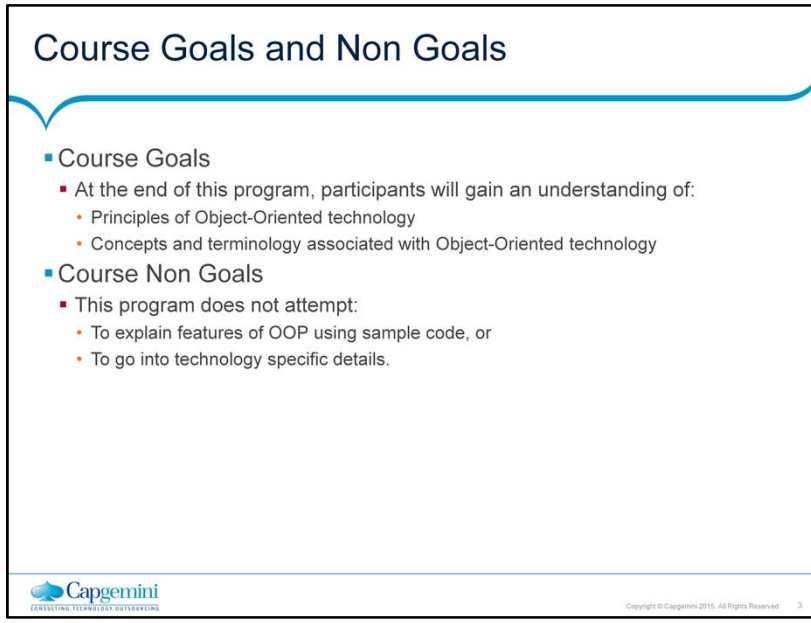
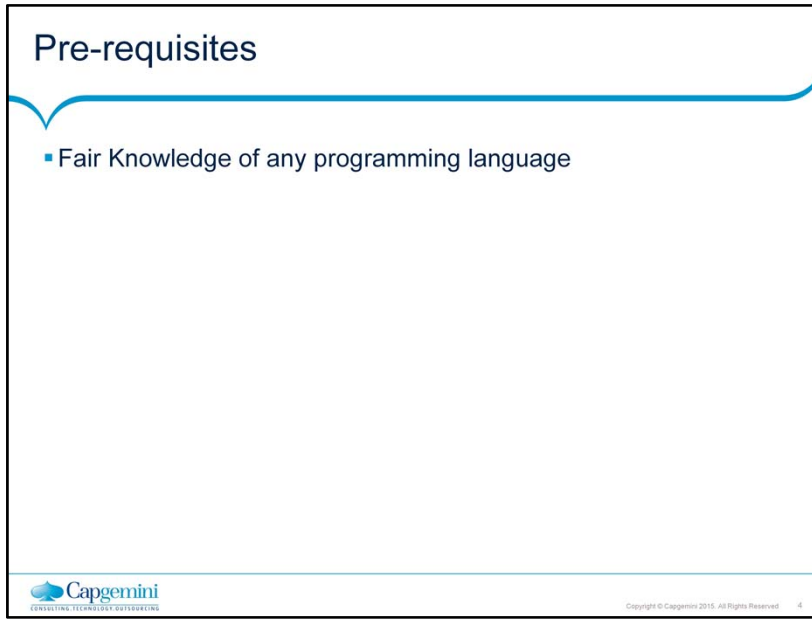
## Document History

| Date | Course Version No. | Software Version No. | Developer / SME | Change Record Remarks |
|---|---|---|---|---|
| 06-Oct-2008 | 0.1D | NA | Shrilata Tavargeri | Content creation. Inputs from existing material in MS word format and corresponding ppt. |
| Nov-2008 | | NA | Veena Deshpande / Rashmi Bharti | Review |
| 08-Dec-2008 | | NA | CLS team | Review |
| Jan-2009 | 1.0 | NA | Nilendra Nagwekar | Review |
| Jul-2009 | 2.0 | NA | Shrilata Tavargeri | Content revamp. Inputs from review team. |
| May-2011 | 2.1 | NA | Veena Deshpande | Refinements to include contents from WBT slides and review comments of Integration Exercise |
| March 2015 | 2.2 | NA | Kavita Arora | Made changes according to revised TOC |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

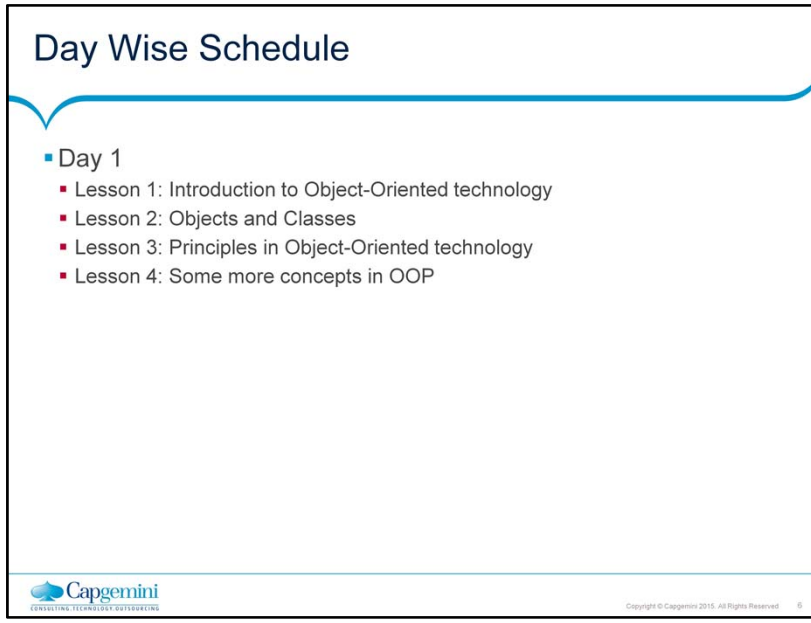
 2


Course Goals and Non Goals
Course Goals
At the end of this program, participants will gain an understanding of:
Principles of Object-Oriented technology
Concepts and terminology associated with Object-Oriented technology
Course Non Goals
This program does not attempt:
To explain features of OOP using sample code, or
To go into technology specific details.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved. 3


Pre-requisites
Fair Knowledge of any programming language
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 4

## Intended Audience

- Developers in Object-Oriented technology


Day Wise Schedule
Day 1
Lesson 1: Introduction to Object-Oriented technology
Lesson 2: Objects and Classes
Lesson 3: Principles in Object-Oriented technology
Lesson 4: Some more concepts in OOP
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 6

# Table of Contents



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

## Table of Contents (contd.)



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

## Table of Contents (contd.)



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

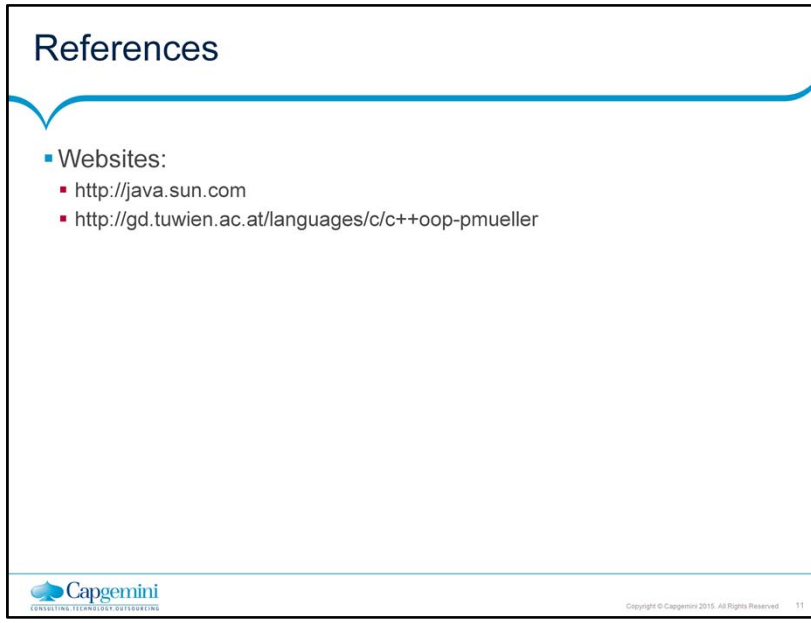
 

## References

- Books:
  - Sams Teach Yourself Object Oriented Programming in 21 Days; by Anthony Sintes (Sams Publishing)
  - Object-Oriented Software Construction; by Bertrand Meyer, (Prentice-Hall)
  - The Object-Oriented Thought Process; by Matt Weisfeld (Sams Publishing)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10


References
Websites:
http://java.sun.com
http://gd.tuwien.ac.at/languages/c/c++oop-pmueller
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
11

## Next Step Courses

- Programming with Object Oriented languages

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

# Object-Oriented Programming

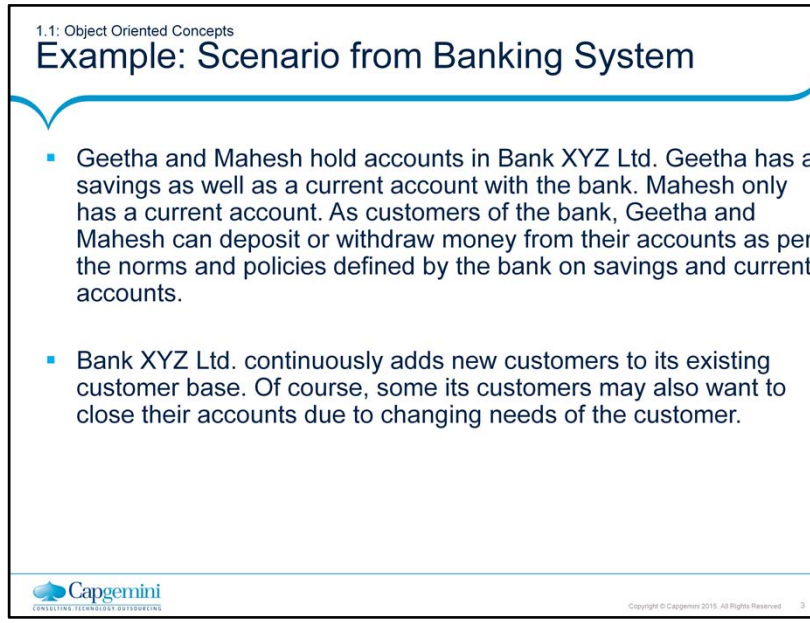
Lesson 01: Introduction to Object-Oriented Technology

## Lesson Objectives

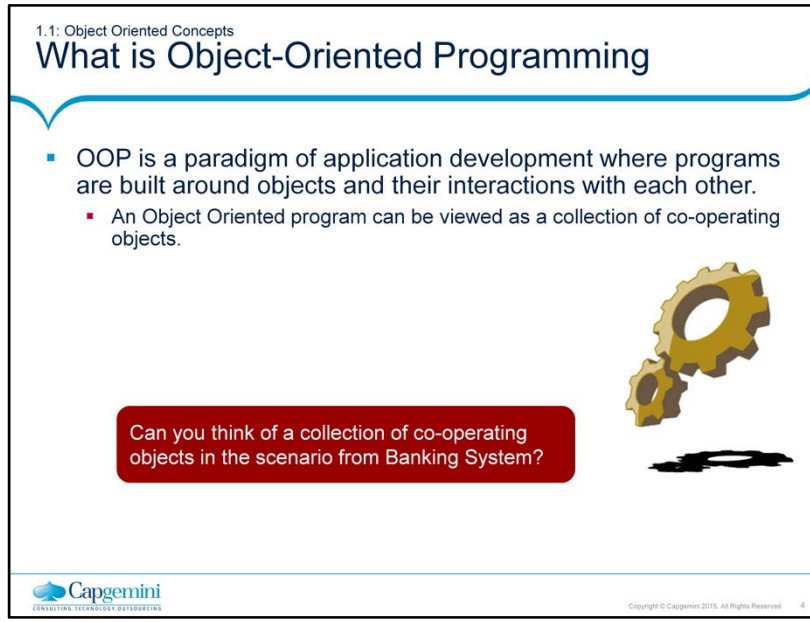
- In this lesson, you will learn:
  - What is Object-Oriented Programming?
  - Why Object-Oriented Programming?
  - Object-Oriented Programming versus traditional software development methodologies
  - Benefits of Object-Oriented technology

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

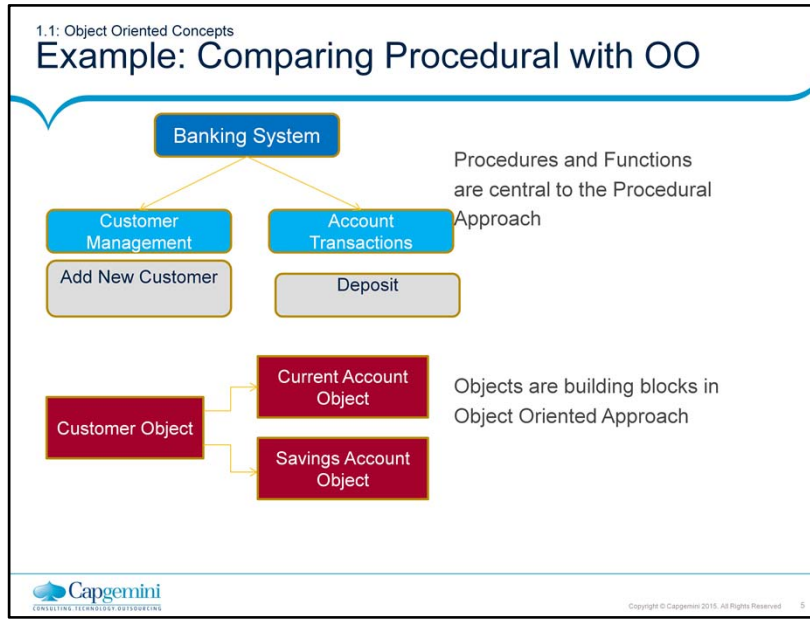
 2




If you consider this scenario for a Banking System, what services and features do you expect the bank to offer? Who are the customers mentioned for this bank? What operations can they perform on their accounts?



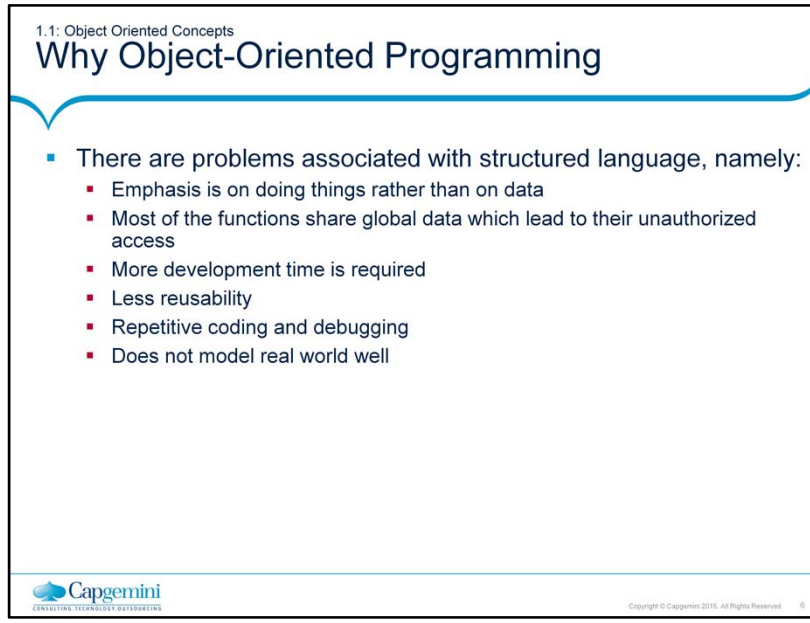

What is Object-Oriented Programming?
The object oriented approach is a fundamental shift from the procedural approach. Instead of functions and procedures being central to the program, in the OO world, we have objects that are the building blocks. An OO program is made up of several objects that interact with each other to make up the application.
For example: In a Banking System, there would be Customer objects pertaining to each customer. Each customer object would own its set of Account Objects, pertaining to the set of Savings and Current Accounts that the customer holds in the bank.
Today, most programming languages are object oriented.
For example: Java, C++, C#
Why do you think most of today's programming languages are object oriented? Are there any advantages of OO languages?




Consider the Banking System. In the procedural approach, we would try to find the top level modules of the application. Eg. A Module to maintain customers, another to maintain accounts and so on. In each of these modules, we would have procedures and functions to take care of different features. Eg. Procedure/Function to add customer or delete customer in the module for maintaining customer. Or procedures/ functions to deposit and withdraw in the module for maintaining accounts. So in the procedural approach, we identify modules, then identify procedures/functions – this is like a top down approach to system development.

Procedures and functions are the building blocks of the application in the procedural approach.

However, in the OO approach, it is the objects which are the building blocks. If we reconsider the same system, we would have objects for customer “Geetha” and customer “Mahesh”; we would also have objects for their respective savings and current accounts. These objects would interact with each other for us to achieve the desired features of the application.




Why Object-Oriented Programming?

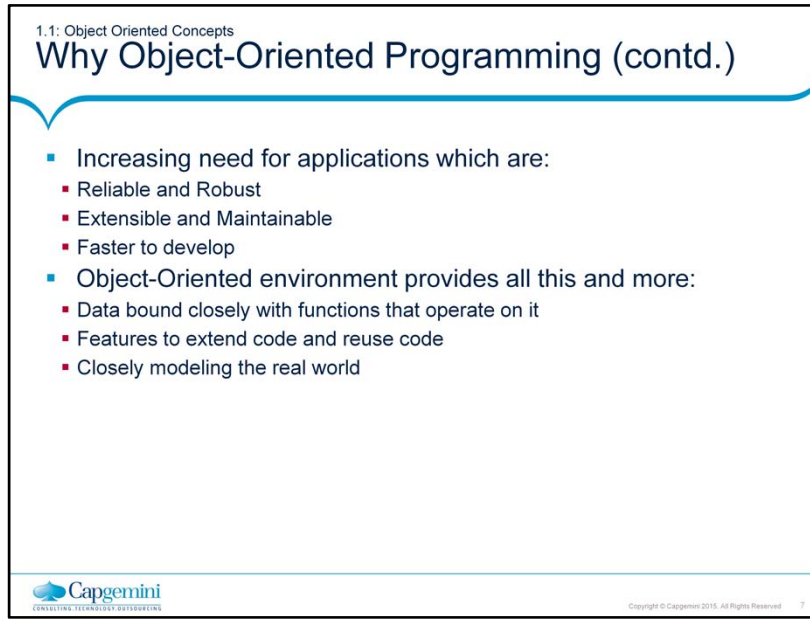
Before the advent of Object-Oriented technology, the primary software engineering methodology was structured or procedural programming. Some drawbacks of this approach are as follows:

Structured programming is based around data structures and subroutines. Data structures are simply containers for the information needed by subroutines. Thus, emphasis is almost entirely on algorithm required to solve a problem.

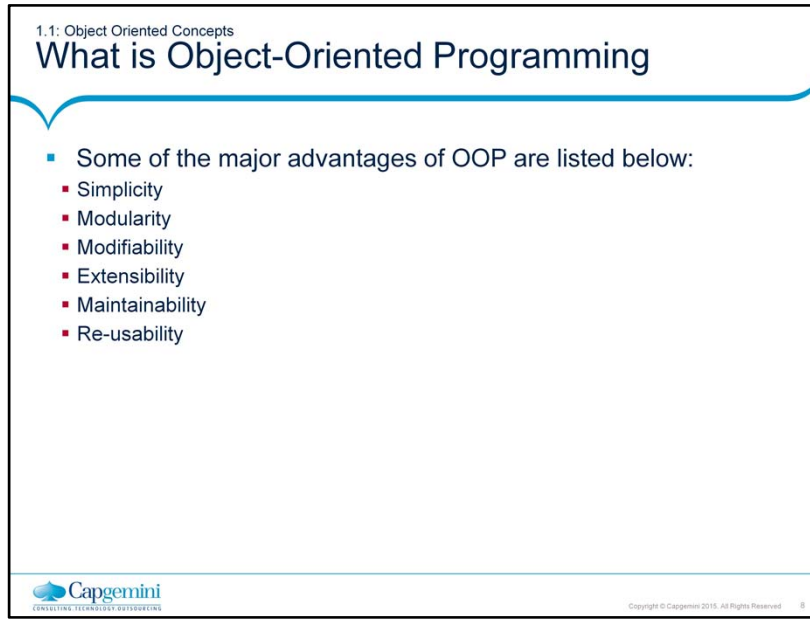
Data is openly accessible to other parts of the program, which is risky.

Structured programming tends to produce a design that is unique to that problem (thus non-reusable). Reusing code from another project usually involves a lot of effort and time. Moreover, since the emphasis is on functionality, functionality change might force entire code to be modified, thus increasing development time.

In structured programming, while analysis starts with a consideration of real-world problems, the real-world focus is lost as requirements are transformed into a series of data flow diagrams.



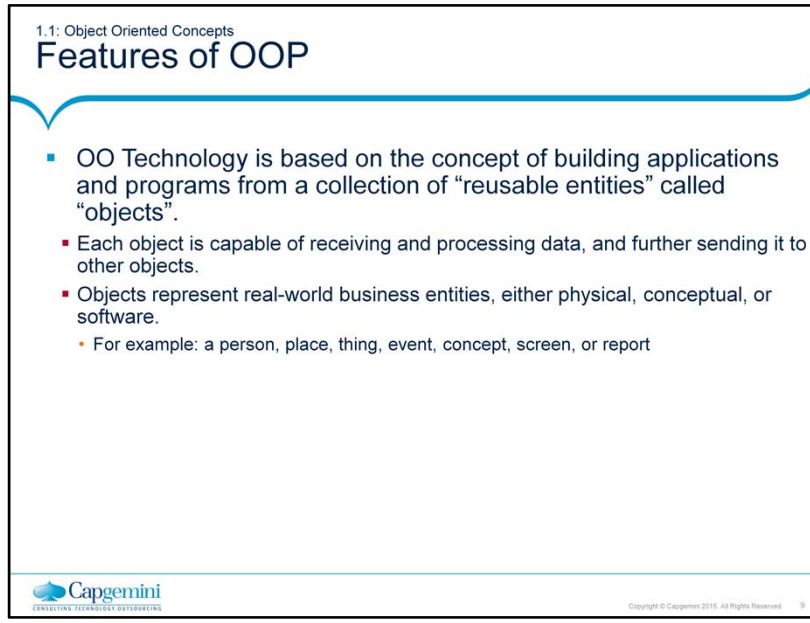

Why Object-Oriented Programming? (contd.)
With increasing complexity of software applications, some of the “must have” features are reliability, robustness, and maintainability. With increasing competition, high productivity which aids faster turn around times for application development and deployment is key.
Object-Oriented programs offer features which allow meeting the above goals. Binding of data and functions together means that data cannot be accessed unless designed for it. There is no possibility of mistakenly corrupting data. Decomposition in terms of objects allow for easily building new programs using existing objects and adding features to existing objects.
Moreover, the Object-Oriented world very closely models the real world, making it much more intuitive and faster to develop. The objects themselves often correspond to phenomena in the real world that the system is going to handle.
For example: An object can be an invoice in a business system or an employee in a payroll system. Thus, OO is natural way programming, i.e., “real life” objects are mapped as is in “programming” as classes / objects.




Advantages of OOP:

Simplicity: Software objects model the real world objects. Hence the complexity is reduced and the program structure is very clear.
Modularity: Each object forms a “separate entity” whose internal workings are decoupled from other parts of the system.
Modifiability: It is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only “public interface” that the external world has to a class is through the use of “methods”.
Extensibility: Adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.
Maintainability: Objects can be separately maintained, thus making locating and fixing problems easier.
Re-usability: Objects can be reused in different programs.

OOP is more than just learning a new language. It requires “a new way of thinking”. The idea is primarily not to concentrate on the cornerstones of procedural languages - data structures and algorithms, instead think in terms of “objects”.




Features of OOP:
Models built by using Object-Oriented technology can be smoothly implemented in any software, by using “Object-Oriented modeling language”. These models also easily adjust to changing requirements.
It is based on best practices. As a result, the systems developed by using Object-Oriented technology are stable with a baselined architecture. The systems are more reliable, scalable, and succinct. They are more easily maintained and adaptable to change.

## Summary

- In this lesson, you have learnt:
  - The Object-Oriented Programming approach for software development
  - How Object-Oriented technology is used to design and develop stable and dynamic systems
  - Advantages of Object-Oriented Programming


Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 10

## Review Question

- Question 1: Which of the following are features of Structured programming:
  - Option 1: Based on Data structure
  - Option 2: Emphasis on data
  - Option 3: Reusability
  - Option 4: Produces a design unique to the problem
- Question 2: Objects can be grouped together in different ways to form new programs.
  - True / False

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

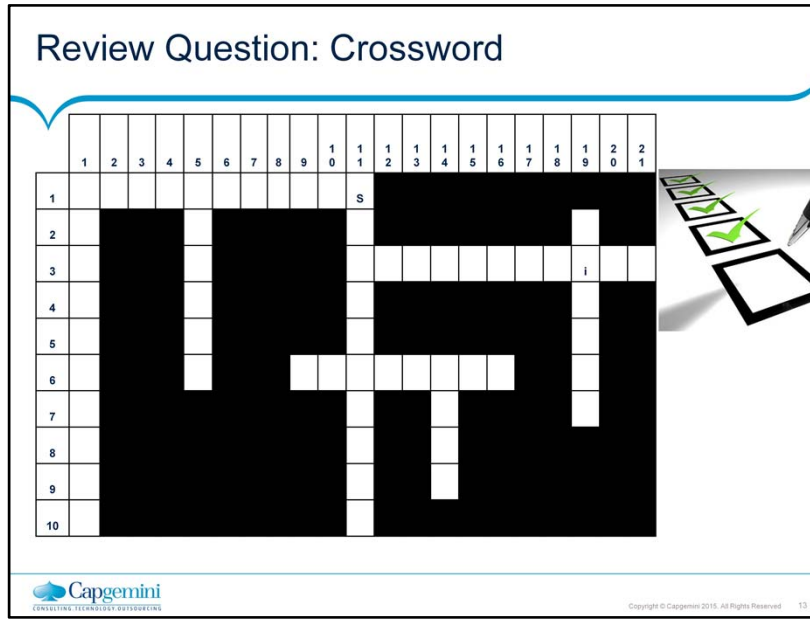
 11

## Review Question

- Question 3 : ___ methodology specifies the steps to be sequentially followed by a computer to execute a program.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12




Clues.: 10 rows (1-10) , 21 Cols (1-21), (Row,Col) Combination

Across:

1-1: Structured programming is based around this (11)

3-11: One of the benefits of Object-Oriented Programming (11)

6-9: Existing objects can be _____ (8)

Below:

1-1: One of the major advantages of OOP (10)

1-5: Data and functions are build around this (6)

1-11: This program data is openly accessible to other parts of the program (10)

2-19: This helps in securing the data (6)

6-14: OOP places emphasis on _____ rather than algorithm (4)

# Object-Oriented Programming

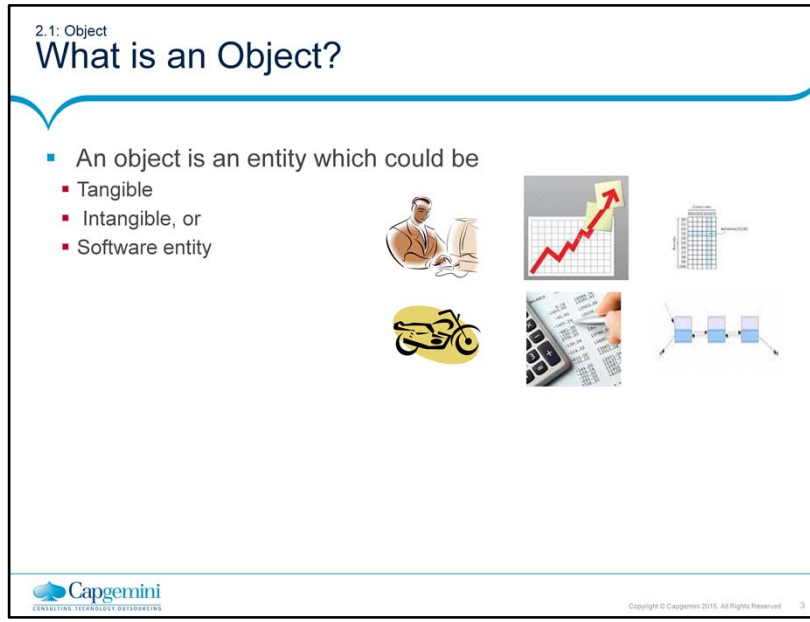
Lesson 2: Objects and Classes

# Lesson Objectives

- In this lesson, you will learn:
  - What is an Object?
    - State, Behavior, and Identity of an Object
  - What is a Class?
    - Attributes and Operations of a Class
    - Access modifiers and its types
    - Constructors and Destructors
    - Static members

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 2

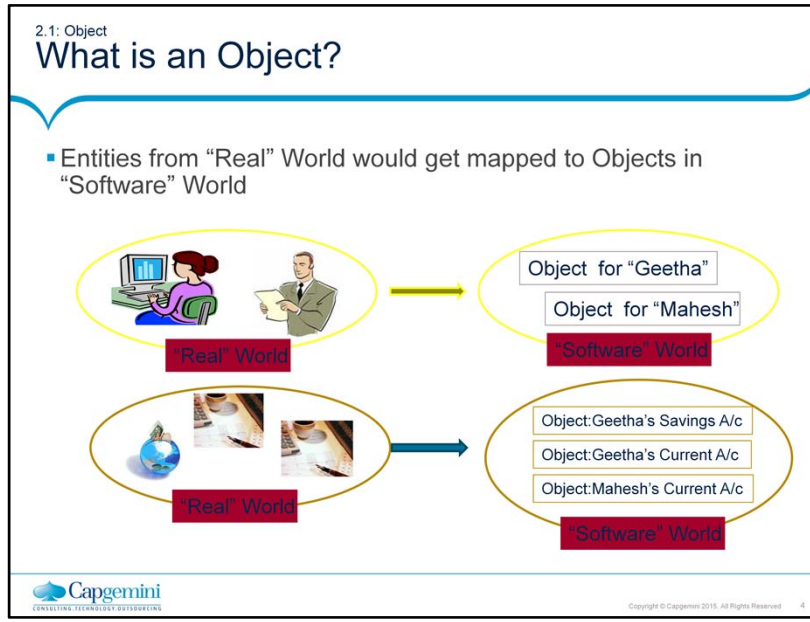



What is an Object?
An object in the real-world, can be physical, conceptual, or a software entity. We come across so many objects in the real world. In fact, everything can be considered as an object - a person, a pen, a vehicle, a book, etc. Essentially these are all tangible things that exist, can be felt, or can be destroyed. However, there could be other entities which may not be considered as objects in the real world, like an account or a contract, or a set of business charts, or a linked list since they are “intangible” or “conceptual”. Nevertheless, they also have a well defined structure and behavior, and hence are treated as objects in the software domain.

Examples of tangible entities
Person, Pen, Vehicle

Examples of intangible entities
Account, Contract, Business Charts

Examples of software entities
Database Management System




Remember the scenario from Banking System? "Geetha" and "Mahesh" from our example of Banking System are entities in the Real World. These would get mapped into corresponding objects in the software world. So in our OO application, we will have an object corresponding to "Geetha" and another object corresponding to "Mahesh". Similarly, we would have objects corresponding to Geetha's Savings Account, Geetha's Current Account as well as Mahesh's Current Account.

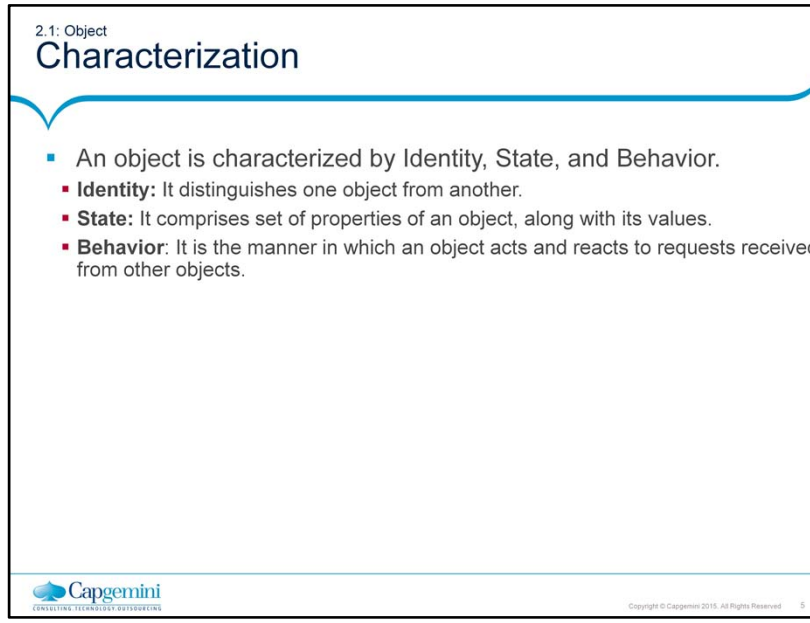
Typically objects could correspond to following

Roles played by people interacting with system (Like Geetha and Mahesh who are "Customer" objects)

Structures used for storing and processing data (Like Account objects for Geetha and Mahesh)

Other systems or devices interacting with system (Like Utility Payment System that can be accessed from Bank System)

Events and entities of the system (Like a transaction or a account status report)



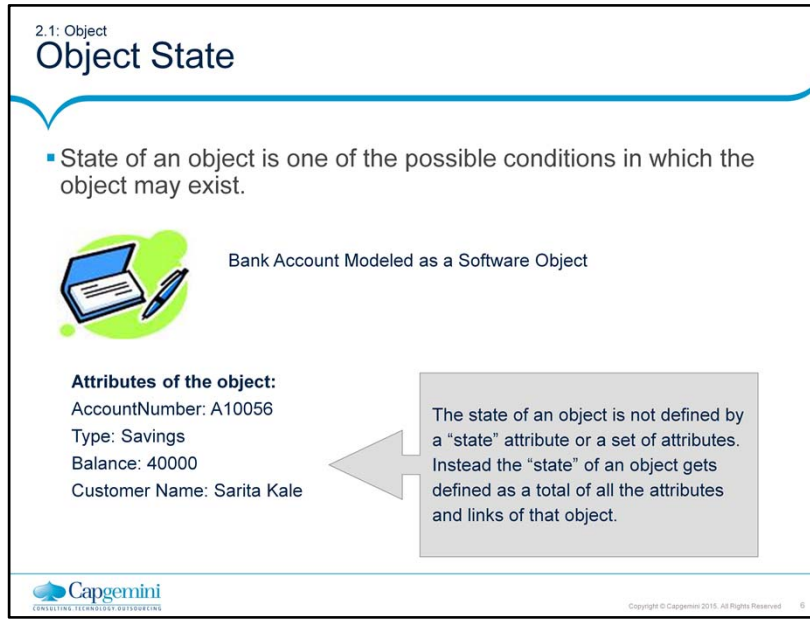

Each object is characterized by identity, state, and behavior.

Identity: Two books of same title are still two different books - they are two instances of a "book" which happen to have similar properties, just as there will be two copies if they existed in the library. The identity of one is to be distinguished from the other.

State: It is one of the possible conditions that an object may be in. It is indicated by the set of values that each of its attributes possesses.

For example: An account object may be in an active or suspended state depending on the balance that it possesses.

Behavior: It is what an object does when it receives instructions. For example: Deposit or withdrawal that occurs against an account object.




What is an Object? – Object State:
The current state of an object is defined by the set of values of its attributes and the links that the object has with other objects. The current state of an object is said to have changed, if one or more attribute values change. The object remains in control of how the outside world is allowed to use it:

by assigning a state (e.g., account number, Account type, balance) to itself, and
by providing methods for changing that state

2.1: Object

## Object State

- Behavior of the Object depends on the State of the Object

Withdrawal depends on the State of the Account Object

**State where withdrawal is permitted**

**Attributes of the object:**
AccountNumber: A10056
Type: Savings
Balance: **40000**
Customer Name: Sarita Kale

**State where withdrawal is NOT permitted**

**Attributes of the object:**
AccountNumber: A10056
Type: Savings
Balance: **500**
Customer Name: Sarita Kale

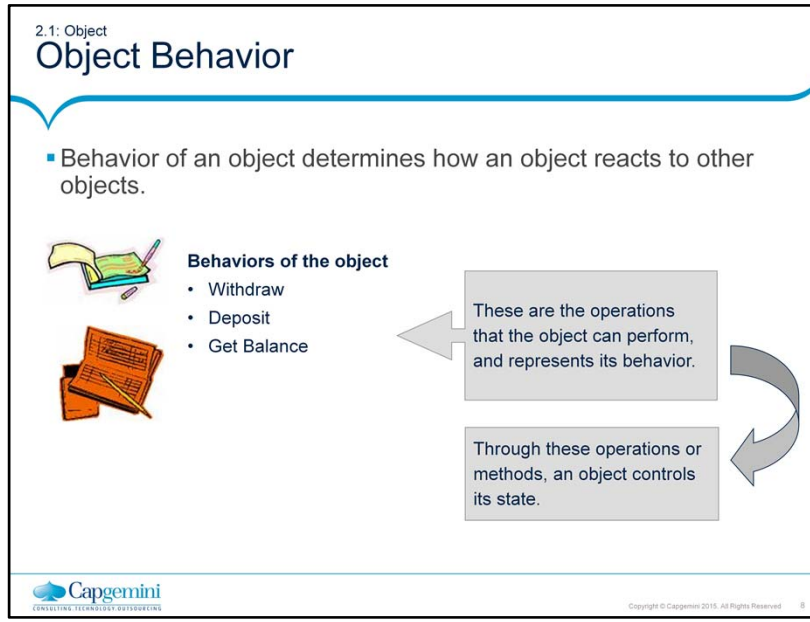
How many States of an Object should we consider??

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

The behaviour of an object in terms of how an object responds, depends on state of object. If the bank has a business rule on the minimum account balance, then an operation such as withdrawal will not be permitted if the balance is less than what is permitted.

At any point, an object will be in a single state. As such, any new combination of attribute values would imply a new state for an object. That means there could be infinitely many states for each object. Should we consider each of these states?? Well No! We only need to consider the object states which will have an impact on the behaviour of the object.




What is an Object? – Object Behavior:
Behavior is what an object does on receiving a set of instructions. Object behavior is represented by the operations that the object can perform.
For example: A Bank ATM object will have operations such as withdraw, print transactions, swipe card, and so on. A bicycle object may have operations such as change gear, change speed, and so on.

2.1: Object

## Object Identity

- Two objects can posses identical attributes (state) and yet have distinct identities.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

What is an Object? – Object Identity:
Two accounts may possess same attributes like type, balance, etc. Yet these two accounts have separate distinct identities. One account could be mine, and the other could be yours. Although objects may share the same state (attributes and relationships), they are separate and independent objects with their own “unique identity”.

2.2: Class

## What is a Class?

- A Class characterizes common structure and behavior of a set of objects.
- It constitutes of Attributes and Operations.
- It serves as a template from which objects are created in an application.

| Class name | Customer |
|---|---|
| Class attributes | Name, Address, Email-ID, TelNumber |
| Class operations | displayCustomerDetails()<br>changeContactDetails() |

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 10

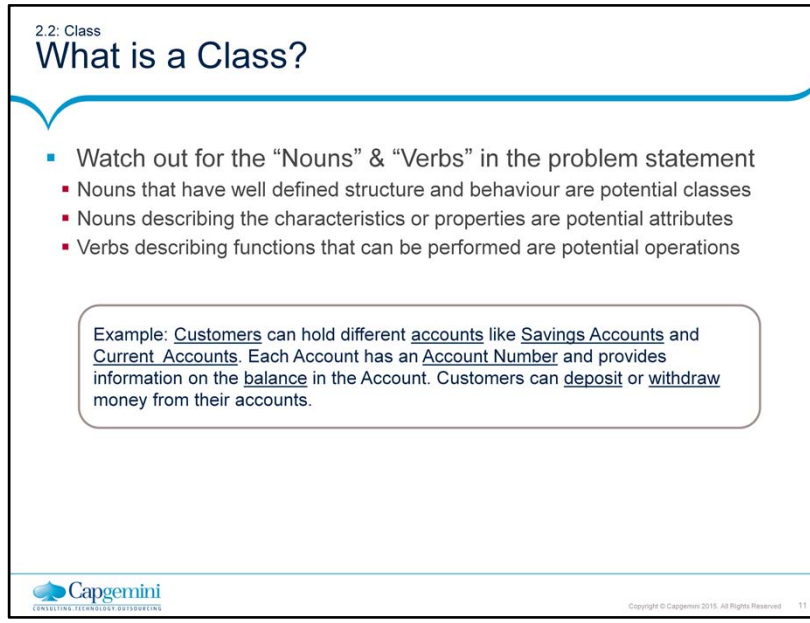
What is a Class?

Classes describe objects that share characteristics, methods, relationships, and semantics. Each class has a name, attributes (its values determine state of an object), and operations (which provides the behavior for the object).

What is the relationship between objects and classes?

What exists in real world is objects. When we classify these objects on the basis of commonality of structure and behavior, what we get are classes. Classes are “logical”, they don’t really exist in real world. While writing software programs, it is the classes that get defined first. These classes serve as a blueprint from which objects are created.

For example: In the example shown in the slide, there may be thousands of bank customers all having same set of attributes (i.e., Name, Address, Email-ID, TelNumber). Each customer is created from the same set of blueprints, and therefore contains the same attributes. Similarly, there can be thousands of Bank Accounts instantiated from the same “Account” class! In terms of Object-Oriented technology, we say that these customers are all “instances” of the “class of objects” known as Customer. A “class” is the blueprint from which individual “objects” are created.




To help identify potential classes, their attributes and their operations, watch out for the nouns and verbs of the problem statement. A Noun having a well defined structure and behaviour, which can be a standalone entity, is a potential class. Nouns which cannot be a stand alone entity but point to properties or characteristics of something could be potential attributes. And finally, verbs describing what could be “done” are potential operations of the class.
In the example here, note that all nouns and verbs are underlined. Potential Classes could be Customer, Account, Savings Account, Current Account. Potential Attributes (of Account Class) could be Account Number and Account Balance. Potential operations (of Account Class) could be deposit and withdraw.

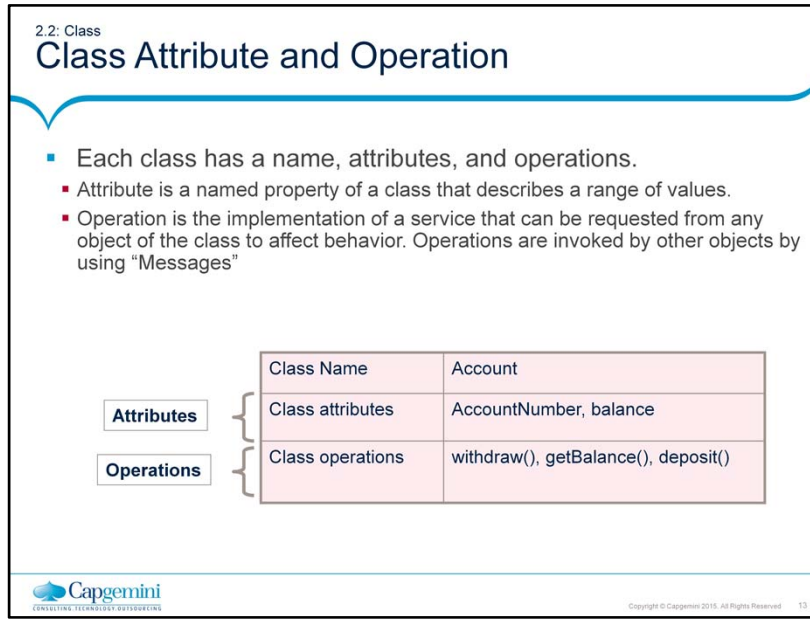
2.2: Class

## Lab

- Objects and Classes
  - Labs 1.1 and 1.2

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

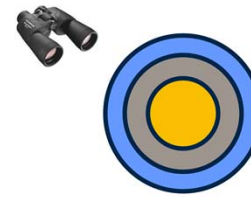



Getting into Details – Class Attribute and Operation:
A class has named properties, which are attributes of the class. An attribute would be of a specific type. At runtime, an object will have associated values for each of its attributes.
A class can have several operations. An operation is an implementation of a service that can be requested from an object.
When an operation of an object has to be invoked by another object, it passes a "message" to the object. Messages would correspond to the operation name.

2.2: Class

## Access Modifiers

- Access Modifiers specify how members of a class will be accessible.
- Three types of Access Modifiers
  - public
  - private
  - protected



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 14

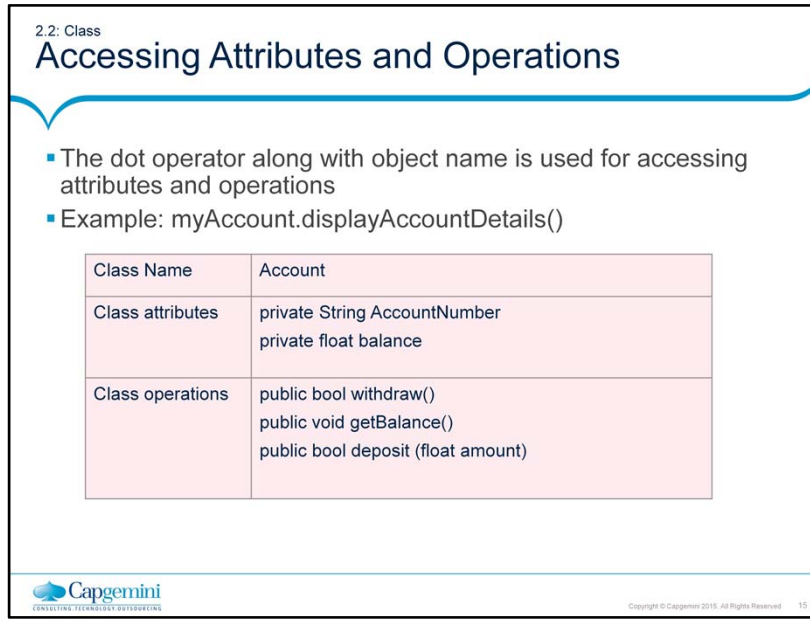
Getting into Details – Access Modifiers:
Access Modifiers specify how members of a class will be accessible. OOP supports the following three types of access modifiers:

Public means accessible to all. An object can access the public variable outside its own class.

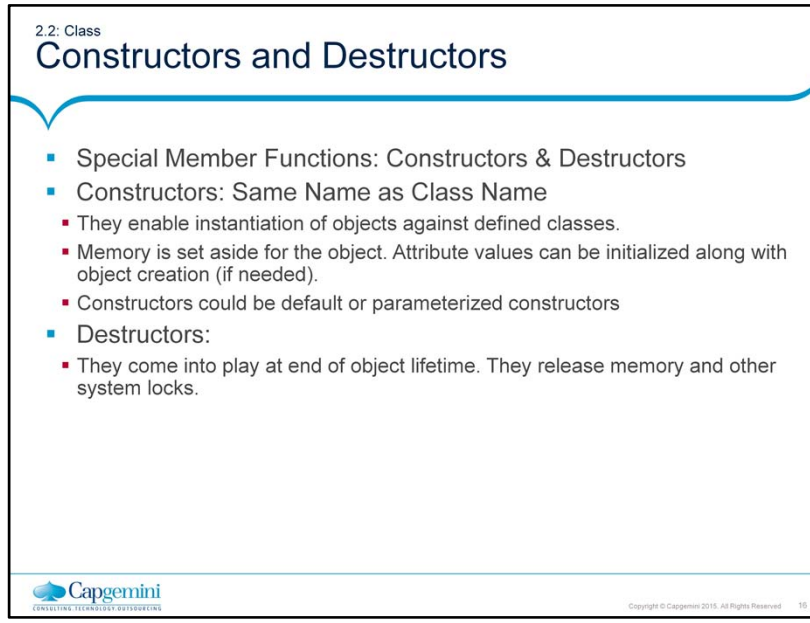
Private means accessible only to the own class. An object can access the private variable only in its own class.

Protected means accessible to own class and its subclass. An object can access the protected variable only in its own class or its subclass. Concept of subclass is discussed later.

Information Hiding means hiding all the details of an object that do not contribute to its essential characteristics. To this end, access modifiers help to restrict the accessibility to attributes and operations.

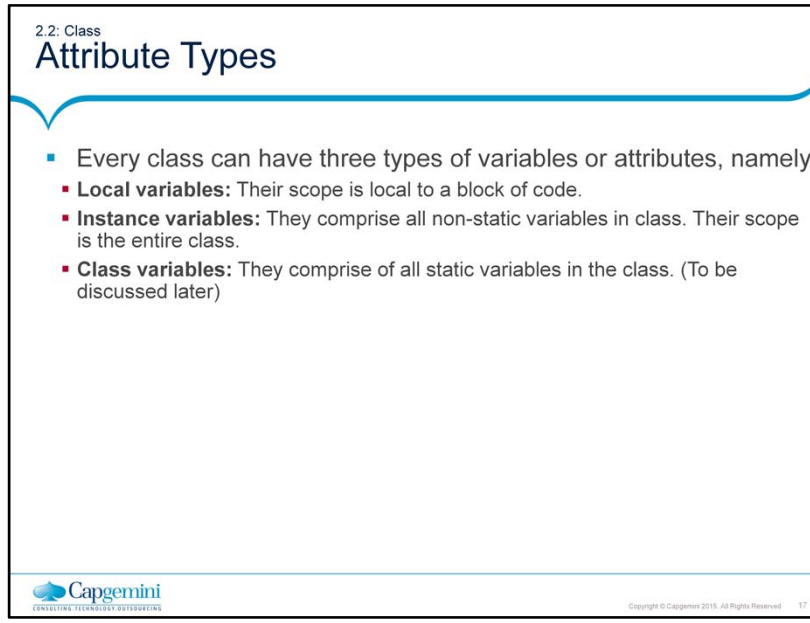



Getting into Details – Access an attribute or an operation:
To access members, a dot operator is used against the corresponding Account object.
For example: If MyAccount is an object of Account type, to display the details of this object, we would specify MyAccount.displayAccountDetails().
From where an attribute or operation can be accessed depends on the access modifiers. For eg. if the Account Class had a private operation for calculating interest, this operation can only be invoked from within another operation of the same class.
Both attributes of the Account Class are private here, which means they are not accessible from outside this class.




Getting into Details – Constructors and Destructors:
Objects are created using Constructors. The Constructors are special member functions of the class that share the same name as that of the class. When objects are created, memory is set aside for them. The attribute values of objects may be initialized, if needed. A class may have multiple constructors since we may want objects to be created and attributes initiated in various ways. The constructor that takes no parameters is a default constructor, while constructors that take 1 or more parameters are the parameterized constructors.

When objects are no longer needed, Destructors come into play. The most common use of destructors is to de-allocate memory that was allocated for the object by the Constructor. There can be only 1 destructor for a class. The destructor name is same as class name, and preceded with tilde (~) sign. Eg. ~Account().




Getting into Details – Attribute Types:
Local variables: These are variables declared within a function, or method, or any block of code, and the arguments passed to a function. They are not accessible outside the block in which they are defined.
Instance variables: These are declared as a part of class. We have seen examples of such variables earlier – AccountNumber, Balance, etc. These are accessible by all the functions written within the class.
Class variables: They are also called Static variables. (This will be covered in detail later) Such variables are useful when all the objects of the same class need to share a common item of information.

2.2: Class

## Lab

- Objects and Classes
  - Labs 2.1 and 2.2

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

## Summary

- In this Lesson, you have learnt that:
  - Objects have an identity, state, and behavior.
  - Class is a user-defined description of set of objects, sharing same structure and behavior.
  - Access modifiers help to restrict the accessibility.
  - Constructors and Destructors help in memory allocation and de-allocation, respectively


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

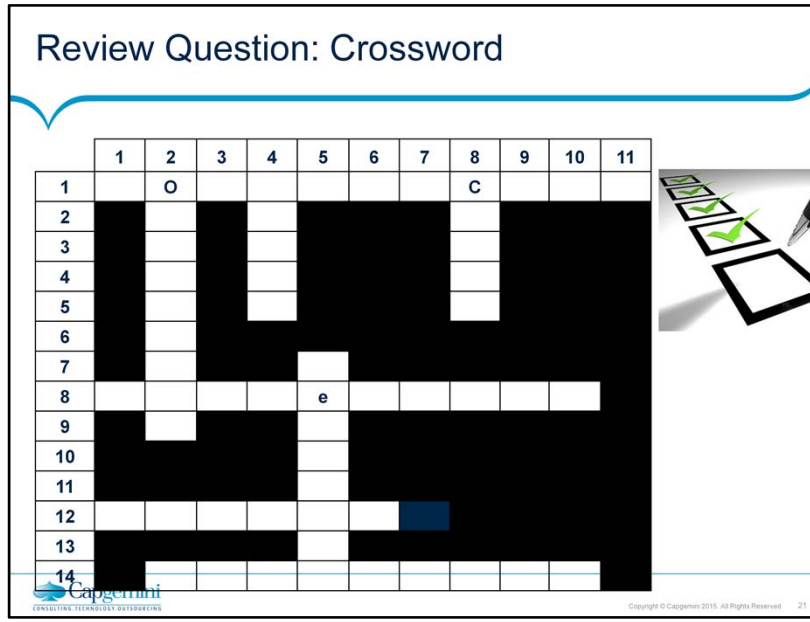
 

## Review Question

- Question 1: ___ determines how an object reacts to other objects.
  - Option 1: State
  - Option 2: Behavior
  - Option 3: Identity
  - Option 4: Attribute
- Question 2: A class can have zero or more number of operations.
  - True / False
- Question 3: ___ variables are accessible by all the functions written within the class.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING




Clues. 14 rows (1-14) , 11 Cols (1-11), (Row,Col) Combination
Across:
1-1: Objects attributes can be initialized using this special function (11)
8-1: A contract is a _____ type of an entity (10)
12-1: This data is allocated once and shared among all object instances of the same type(6)
14-2: Named property of a class that describes a range of values. (9)

Below:
1-2: Implementation of a service (9)
1-4: This describes the possible conditions that an object may be in (5)
1-8: Blue print of an object (5)
7-5: Operations the object can perform (8)

# Object-Oriented Programming

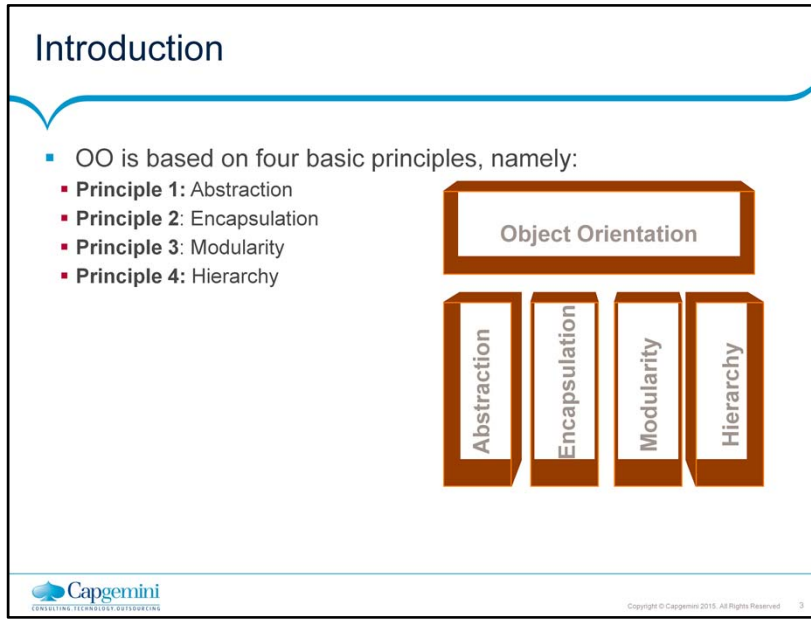
Lesson 3: Principles in Object-Oriented Technology

## Lesson Objectives

- In this lesson, you will understand the basic principles of Object-Oriented technology, namely:
  - Abstraction
  - Encapsulation
  - Modularity
  - Hierarchy and its types
    - Types of Inheritance Hierarchy
  - Polymorphism and Types of Polymorphism

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

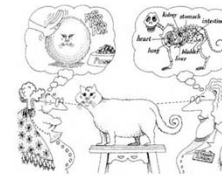



Object-Oriented Principles:
Object-Oriented technology is built upon a sound engineering foundation, whose elements are collectively called the “object model”. This encompasses the following principles – Abstraction, Encapsulation, Modularity, and Hierarchy
Each of these principles has been discussed in detail in the subsequent slides.

3.1: Object-Oriented Principles

## Concept of Abstraction

- Focus only on the essentials, and only on those aspects needed in the given context.
  - **For example:** Customer needs to know what is the interest he is earning; and may not need to know how the bank is calculating this interest
  - **For example:** Customer Height / Weight not needed for Banking System!



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

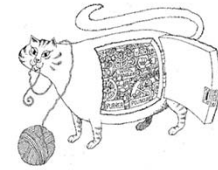
 

Abstraction:
Abstraction is determining the essential qualities. By emphasizing on the important characteristics and ignoring the non-important ones, one can reduce and factor out those details that are not essential, resulting in less complex view of the system. Abstraction means that we look at the external behavior without bothering about internal details. We do not need to become car mechanics to drive a car!
Abstraction is domain and perspective specific. Characteristics that appear essential from one perspective may not appear so from another. Let us try to abstract “Person” as an object. A person has many attributes including height, weight, color of hair or eyes, etc. Now if the system under consideration is a Banking System where the person is a customer, we may not need these details. However, we may need these details for a system that deals with Identification of People.

3.1: Object-Oriented Principles

## Concept of Encapsulation

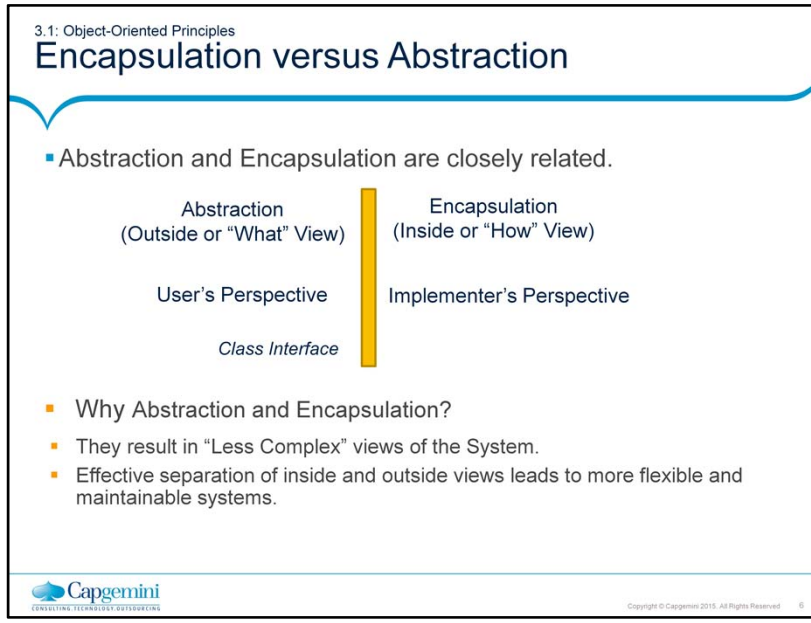
- "To Hide" details of structure and implementation
  - **For example:** It does not matter what algorithm is implemented internally so that the customer gets to view Account status in Sorted Order of Account Number.



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 5

Encapsulation:
Every object is encapsulated in such a way, that its data and implementations of behaviors are not visible to another object. Encapsulation allows restriction of access of internal data. Encapsulation is often referred to as information hiding. However, although the two terms are often used interchangeably, information hiding is really the result of encapsulation, not a synonym for it.




Encapsulation versus Abstraction:
The concepts of Abstraction and Encapsulation are closely related. In fact, they can be considered like two sides of a coin. However, both need to go hand in hand. If we consider the boundary of a class interface, abstraction can be considered as the User's perspective, while encapsulation as the Implementer's perspective.
Abstraction focuses on the outside view of an object (i.e., the interface). Encapsulation (information hiding) prevents clients from seeing its inside view, where the behavior of the abstraction is implemented.
The overall benefit of Abstraction and Encapsulation is "Know only that, what is totally mandatory for you to Know". Having simplified views help in having less complex views, and therefore a better understanding of system. Increased Flexibility and Maintainability comes from keeping the separation of "interface" and "implementation". Developers can change implementation details without affecting the user's perspective.

3.1: Object-Oriented Principles

## Examples: Abstraction and Encapsulation

- Class is an abstraction for a set of objects sharing same structure and behavior

→ Customer Class

- "Private" Access Modifier ensures encapsulation of data and implementation

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

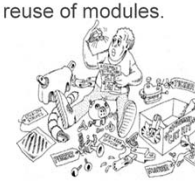
 7

Abstraction and Encapsulation:
When we define a blueprint in terms of a class, we abstract the commonality that we see in objects sharing similar structure and behaviour. Abstraction in terms of a class thus provides the "outside" or the user view.
The implementation details in terms of code written within the operations need not be known to the users of the operations. This is again therefore abstracted for the users. The implementation details are completely encapsulated within the class.
The data members and member functions which are defined as private are "encapsulated" and users of the class would not be able to access them.

3.1: Object-Oriented Principles

## Concept of Modularity

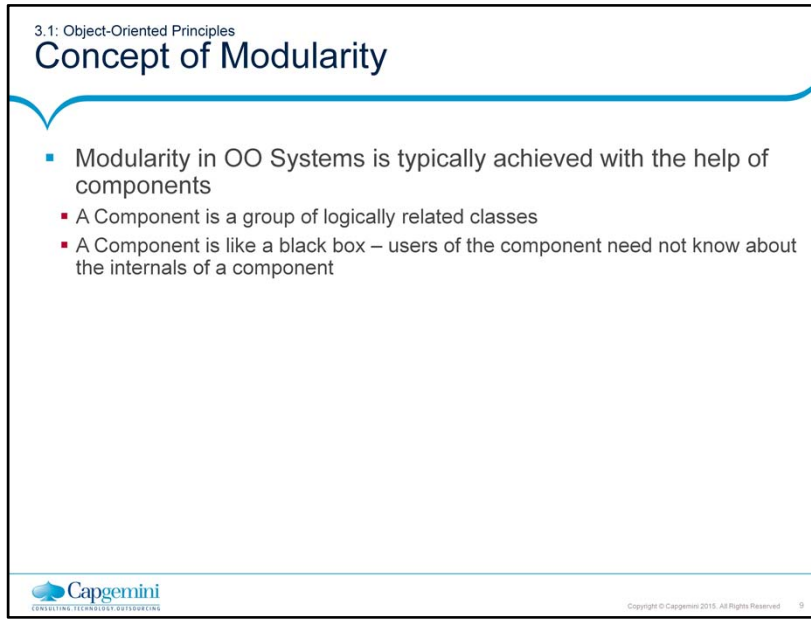
- Decomposing a system into smaller, more manageable parts
  - Example: Banking System can have different modules to take care of Customer Management, Account Transactions, and so on.
- Why Modularity?
  - Divide and Rule! Easier to understand and manage complex systems.
  - Allows independent design and development, as well as reuse of modules.



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

Modularity:
Modularity is obtained through decomposition, i.e., breaking up complex entities into manageable pieces. An essential characteristic is that the decomposition should result in modules which can be independent of each other.
As modules are groups of related classes, it is possible to have parallel developments of modules. Changes in one may not affect the other modules. Modularity is an essential characteristic of all complex systems. Well designed modules can be reused in similar situations in other designs.




Modularity:
Modularity is one of the corner stones of structured or procedural approach, where functions or procedures are the smallest unit of the application, and they help in achieving the modularity required in the system. In contrast, it is the class which is the smallest unit in OO Systems.
Modularity in OO systems is implemented using Components. A component is a set of logically related classes. For eg. several classes may need to be used together for an application to retrieve data from the underlying databases. So this collection of logically related set of classes for retrieving data can be bundled together as a component for Data Access.
A user of a component need not know about the internals of a component. Modularity thus helps in simplifying the complexity.

3.1: Object-Oriented Principles

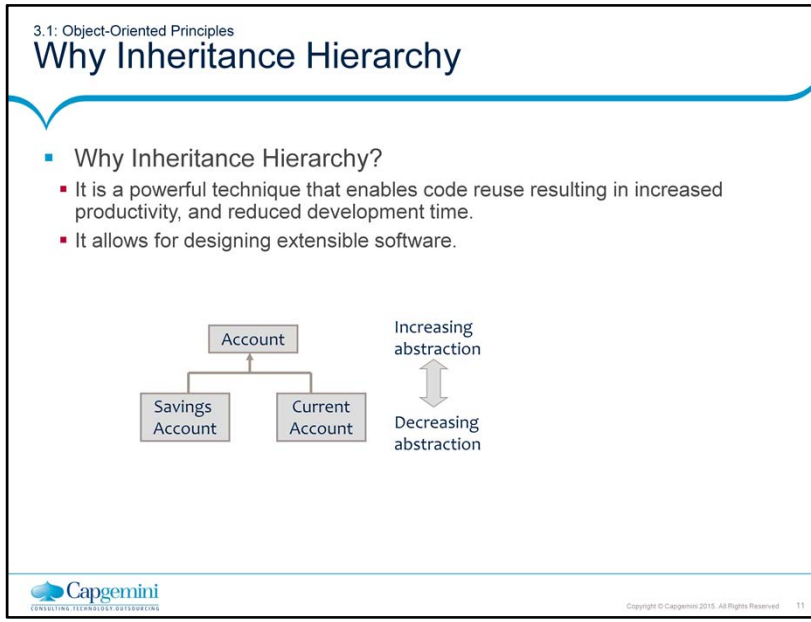
## Concept of Hierarchy

- A ranking or ordering of abstractions on the basis of their complexity and responsibility
- It is of two types:
  - Class Hierarchy: Hierarchy of classes, Is A Relationship.
    - Example: Accounts Hierarchy
  - Object Hierarchy: Containment amongst Objects, Has A Relationship.
    - Example: Window has a Form seeking customer information, which has text boxes and various buttons.

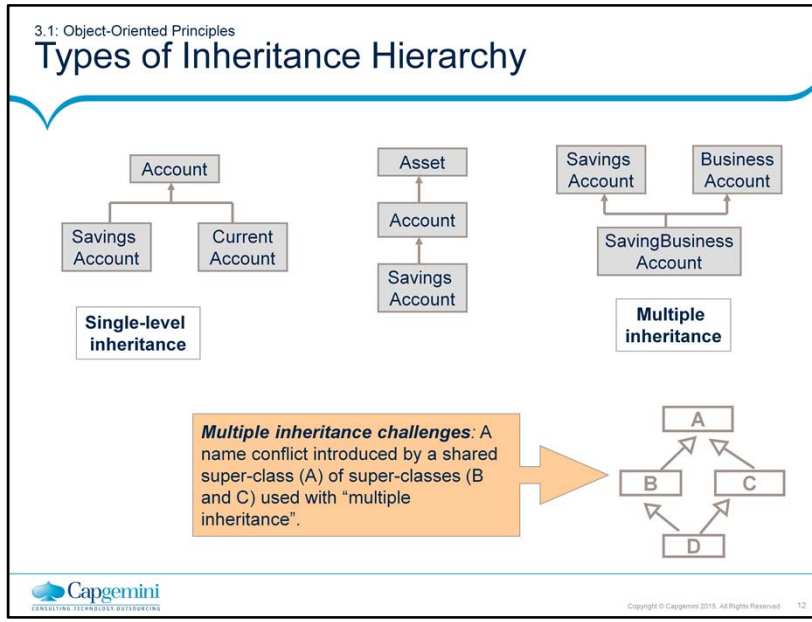
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

Hierarchy:
Hierarchy is the systematic organization of objects or classes in a specific sequence in accordance to their complexity and responsibility. In a class hierarchy, as we go up in the hierarchy, the abstraction increases. So all generic attributes and operations pertaining to an Account are in the Account superclass. Specific properties and methods pertaining to specific accounts like current and savings account is part of the corresponding sub class. Is A relationship holds true – Current Account is an account; Savings Account is an Account.
In object hierarchy, it is the containership property, where one object is contained within another object. So Window contains a Form, a Form contains textboxes and buttons, and so on. Here we have “Has A” relationship – Form has a textbox.




Why Inheritance Hierarchy?
Inheritance is the process of creating new classes, called derived classes, from existing or base classes. The derived class inherits all the capabilities of the base class, but can add some specificity of its own. The base class is unchanged by this process.
Once the base class is written and debugged, it need not be touched again, but can nevertheless be adapted to work in different situations. Reusing existing code saves time and money and increases the program reliability.

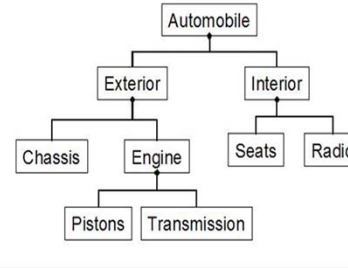



Types of Inheritance Hierarchy:
Single-level inheritance: It is when a sub-class is derived simply from its parent class.
Multilevel Inheritance: It is when a sub-class is derived from a derived class. Here a class inherits from more than one immediate super-class. Multilevel inheritance can go up to any number of levels.
Multiple Inheritance: It refers to a feature of some OOP languages in which a class can inherit behaviors and features from more than one super-class.
Finally, we could have Hybrid inheritance, which is essentially combination of the various types of inheritance mentioned above.

3.1: Object-Oriented Principles

## Object Hierarchy

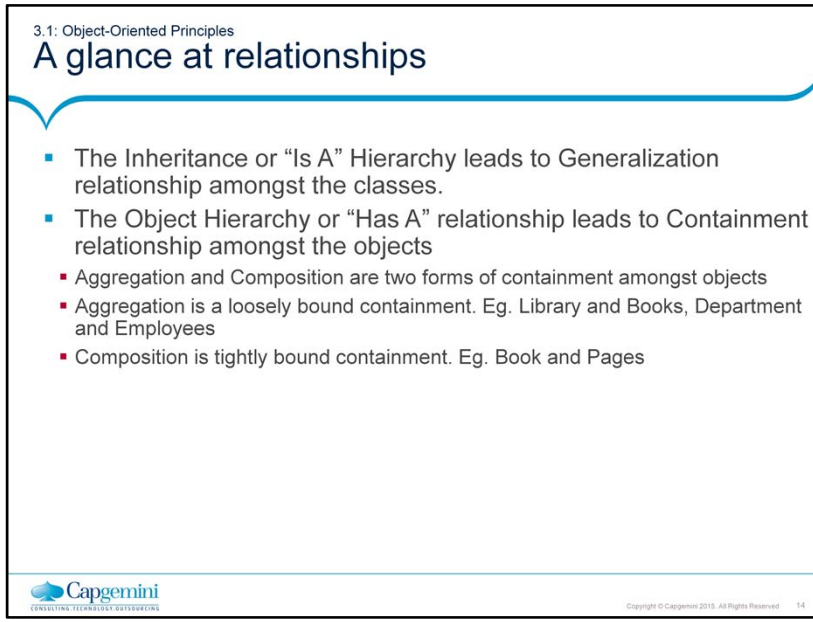
- "Has-a" hierarchy is a relationship where one object "belongs" to (is a part or member of) another object, and behaves according to the rules of ownership.



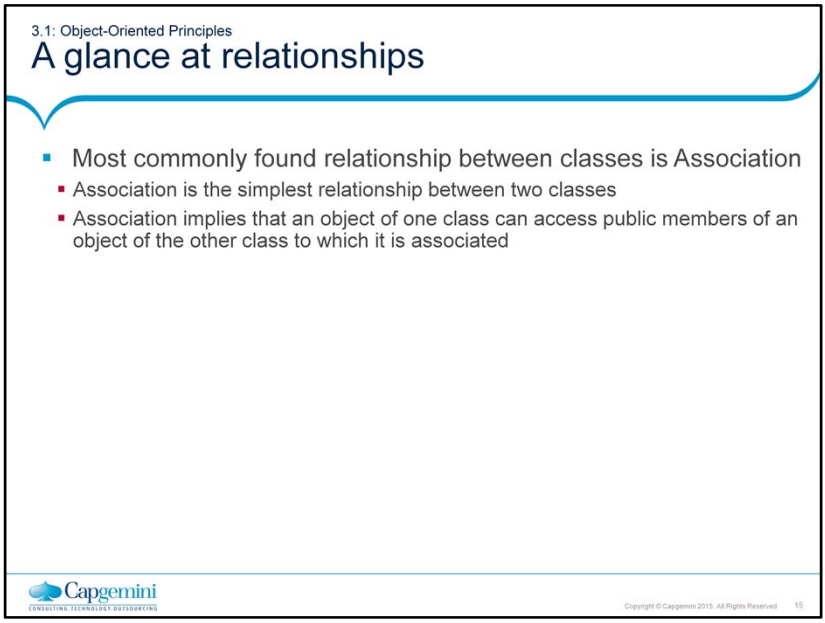

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

Note: The container hierarchy (has-a hierarchy) is in contrast to the inheritance hierarchy, i.e., the "generic-specific" levels do not come in here.




As seen earlier, in an inheritance hierarchy, the super class is the more generic class, and subclasses extend from the generic class to add their specific structure and behaviour. The relationship amongst these classes is a generalization relationship. OO Languages provide specific syntaxes to implement inheritance or the generalization relationship.
Has A or Containment is further of two forms depending on how tight is the binding between the container (“Whole”) and its constituents (“Part”). In the whole-part relationship, if the binding is loose i.e. the contained object can have an independent existence, the objects are said to be in an aggregation relationship. On the other hand, if the constituent and the container are tightly bound (Eg. Body & parts like Heart, Brain..), the objects are said to be in a composition relationship.




The relationship we are most likely to see amongst classes is the Association Relationship. When two classes have an association relationship between them, it would mean that an object of one class can access the public members of the other class with which it is associated. For eg. if a “Sportsman” class is associated with “Charity” class, it means that a Sportsman object can access features such as “View upcoming Charity Events” or “Donate Funds” which are defined within the “Charity” class.

3.2: Polymorphism

## Key Feature – Polymorphism

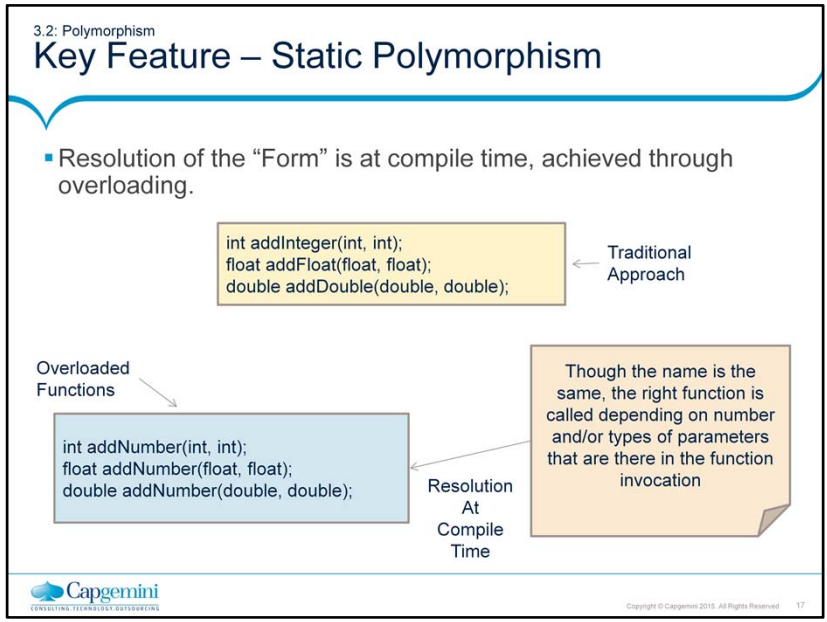
- It implies One Name, Many Forms.
- It is the ability to hide multiple implementations behind a single interface.
- There are two types of Polymorphism, namelv:
  - Static Polymorphism
  - Dynamic Polymorphism


Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

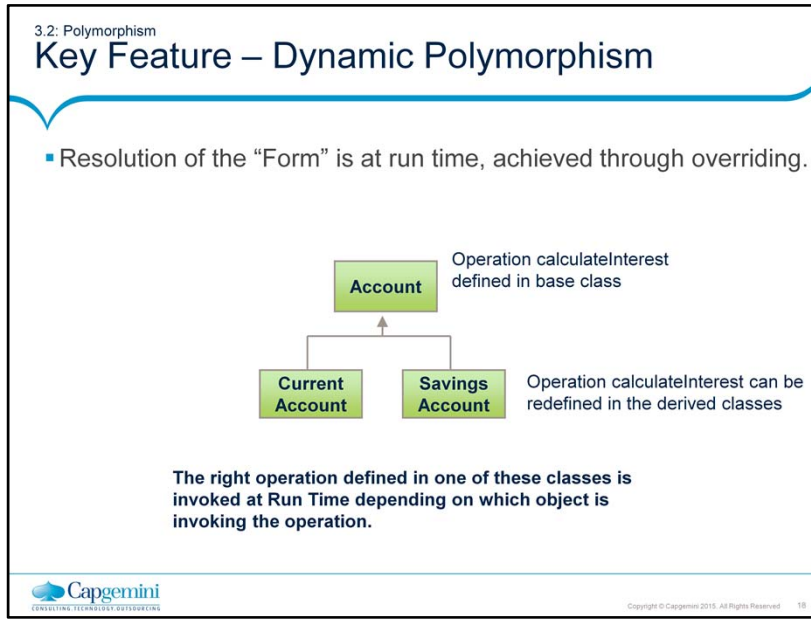
 16

Polymorphism:
The word Polymorphism is derived from the Greek word "Polymorphous", which literally means "having many forms".
Polymorphism allows different objects to respond to the same message in different ways!
There are two types of polymorphism, namely:

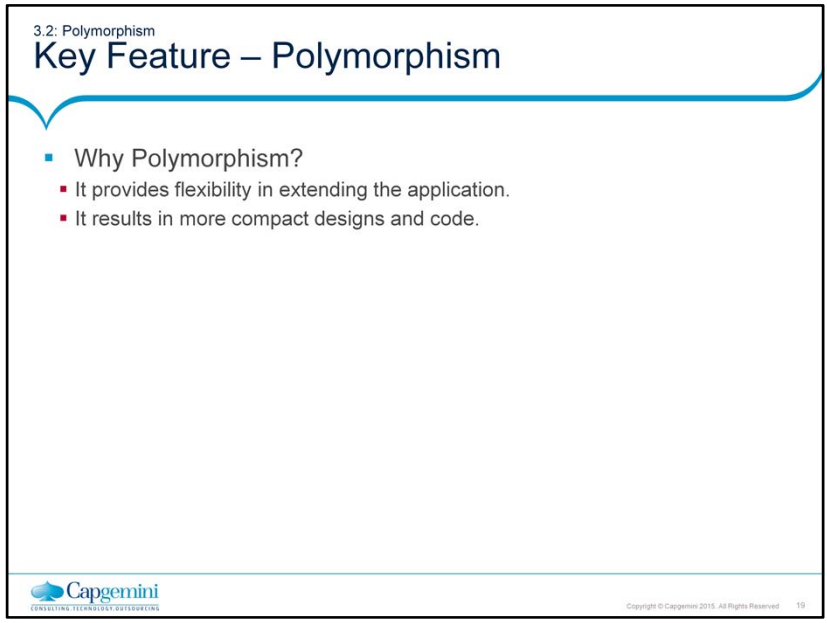
Static (or compile time) polymorphism, and
Dynamic (or run time) polymorphism




Polymorphism (contd.):
Overloading is when functions having same name but different parameters (types or number of parameters) are written in the code. When Multiple Sort operations are written, each having different parameter types, the right function is called based on the parameter type used to invoke the operation in the code. This can be resolved at compile time itself since the type of parameter is known.




Polymorphism (contd.):
On the other hand, the function calculateInterest can be coded across different account classes. At runtime, based on which type of Account object (i.e., object of Current or Savings Account) is invoking the operation, the right operation will be referenced. Overriding is when functions with same signature provide for different implementations across a hierarchy of classes.
As seen in the example here, inheritance hierarchy is required for these objects to exhibit polymorphic behaviour. The classes here are related since they are different types of Accounts, so it is possible to put them together in an inheritance hierarchy. Does that mean that polymorphism is possible only with related classes in an inheritance hierarchy? The answer is No!
We can have unrelated classes participating in polymorphic behavior with the help of the "Interface" concept, which we shall study in a subsequent section.




Polymorphism (contd.):
If the banking system needs a new kind of Account, extending without rewriting the original code, then it is possible with the help of polymorphism.
In a Non OO system, we would write code that may look something like this:

```
IF Account is of CurrentAccountType THEN
        calculateInterest_CurrentAccount()
IF Account is of SavingsAccountType THEN
        calculateInterest_SavingAccount()
```

The same in a OO system would be myAccount.calculateInterest(). With object technology, each Account is represented by a class, and each class will know how to calculate interest for its type. The requesting object simply needs to ask the specific object (For example: SavingsAccount) to calculate interest. The requesting object does not need to keep track of three different operation signatures.

3.2: Polymorphism

## Lab

- Class Relationships
  - Labs 3.1 and 3.2

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20

## Summary

- In this lesson, you have learnt that:
  - Object-Orientated Programming is guided by four basic principles, namely:
    - Abstraction
    - Encapsulation
    - Modularity
    - Hierarchy
  - Polymorphism allows multiple implementations to be hidden behind a single interface.


Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

## Review Question

- Question 1: The 4 basic principles of Object Model are ___, ___, ___ and ___.
- Question 2: Function Overriding is kind of polymorphism.
  - True / False
- Question 3: ___ hierarchy is a relationship where one object behaves according to the rules of ownership.

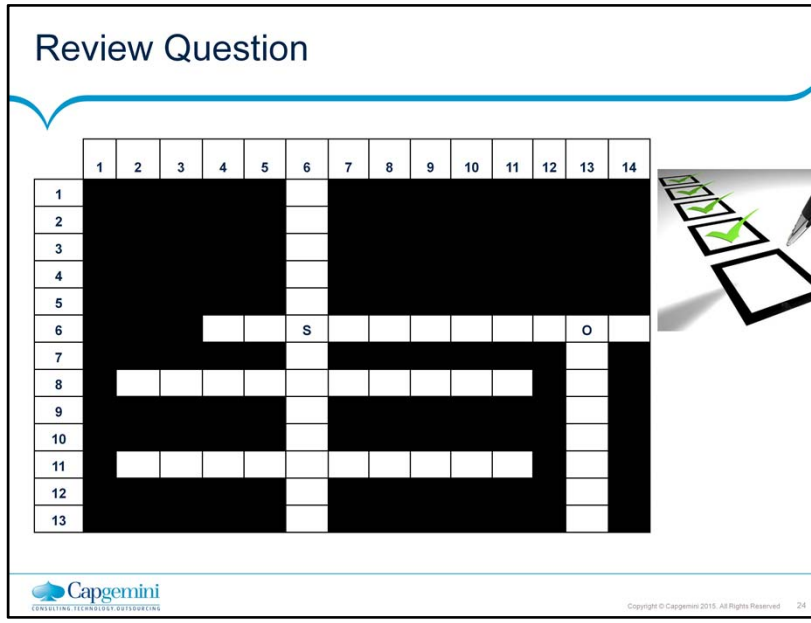
## Review Question

- Question 4: Abstraction focuses on:
  - Option 1: implementation
  - Option 2: observable behavior
  - Option 3: object interface

- Question 5: Polymorphism can be achieved by:
  - Option 1: Hierarchy of Classes providing polymorphic behavior
  - Option 2: Interfaces
  - Option 3: Containment of Objects

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23




Clues. 13 rows (1-10) , 14 Cols (1-14), (Row,Col) Combination
Across:
6-4 : Determining the essential qualities of an object (11)
8-2 : One of the basic advantages of OO (10)
11-2 : Type of inheritance where sub-class is derived from a derived class (10)

Below:
1-6 : Helps to restrict access to its internal data (13)
6-13 : Sub-class extends the existing version of a base class method (8)

# Object-Oriented Programming

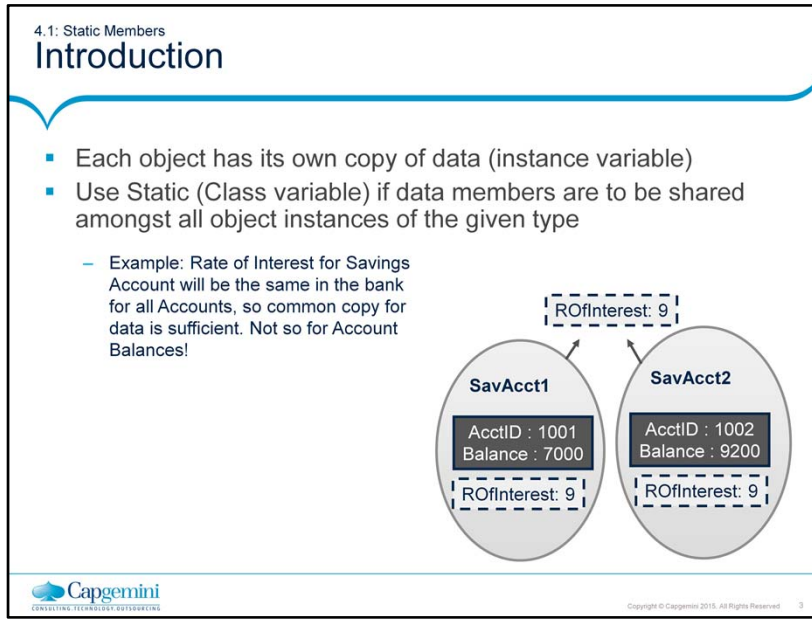
Lesson 04: Some More Concepts in OOP

## Lesson Objectives

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

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

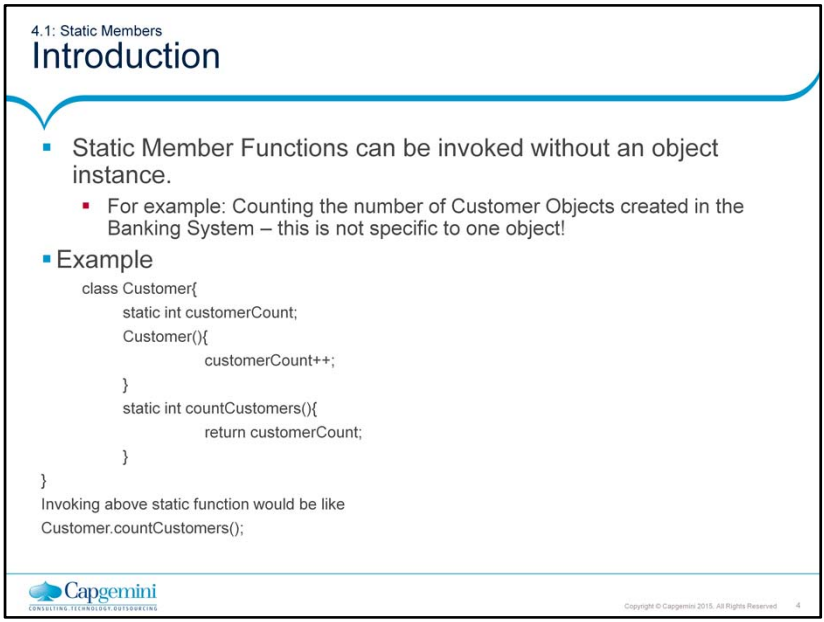
 2



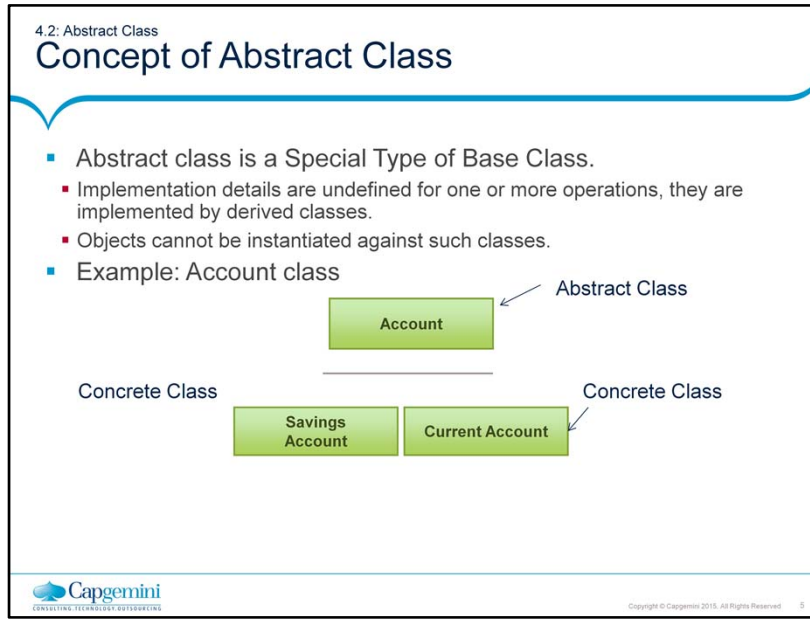

Getting into Details – Static Members:

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!

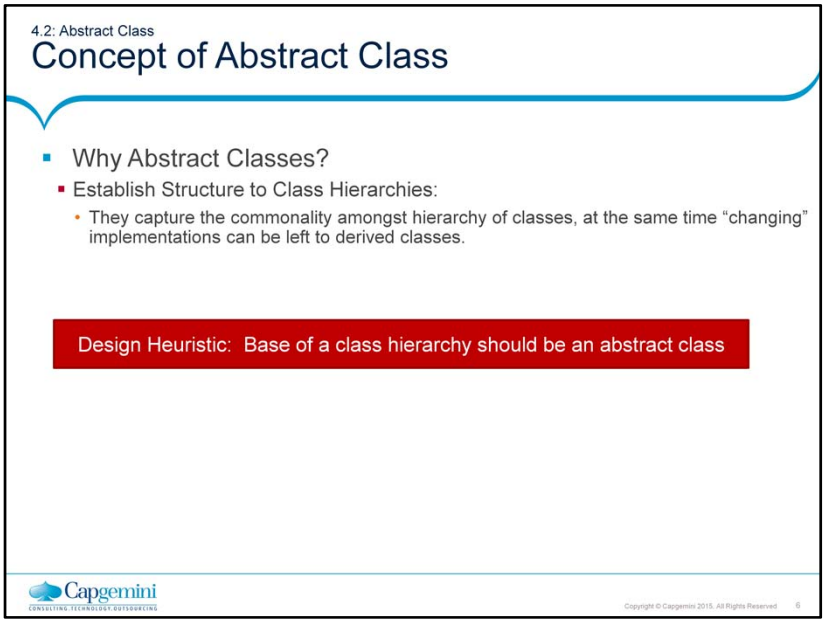
Here we need a mechanism whereby data stored once can be shared by all objects of a given class. Static or Class Variables provide this mechanism. Static data is allocated once and shared among all object instances of the same type. So as in the example above, Rate of Interest need not be shared within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




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.




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.




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.

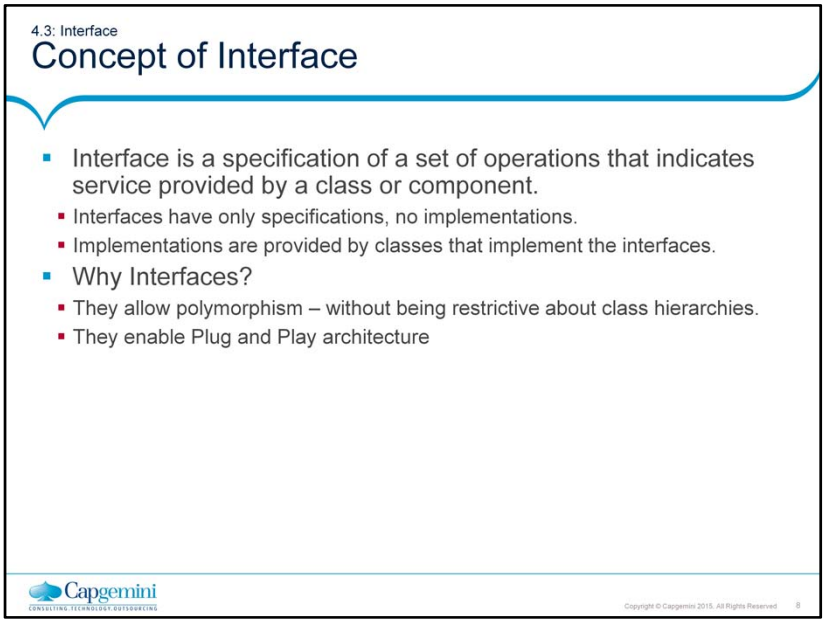
4.2: Abstract Class

## Lab

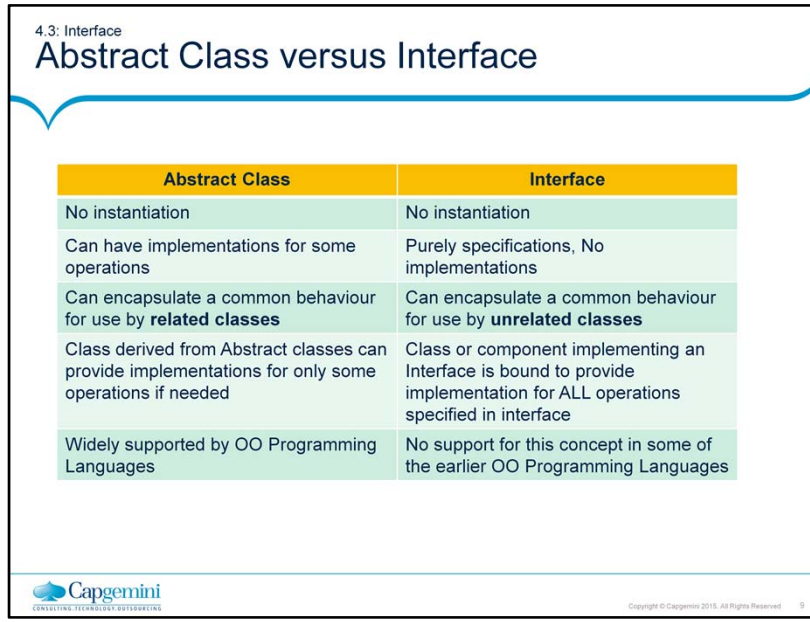
- Abstract Classes and Polymorphism
  - Lab 4.1

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

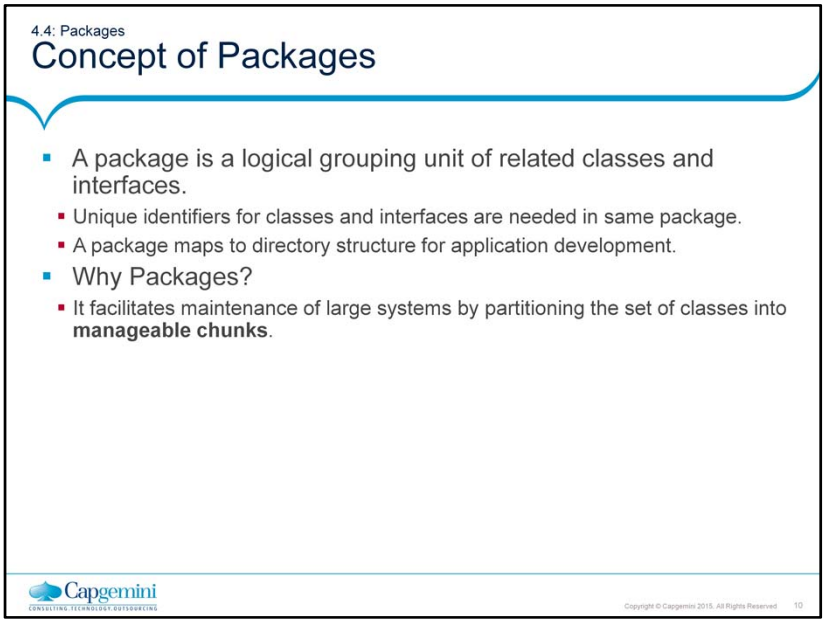
 7




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.




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.




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

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 11

## 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


Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

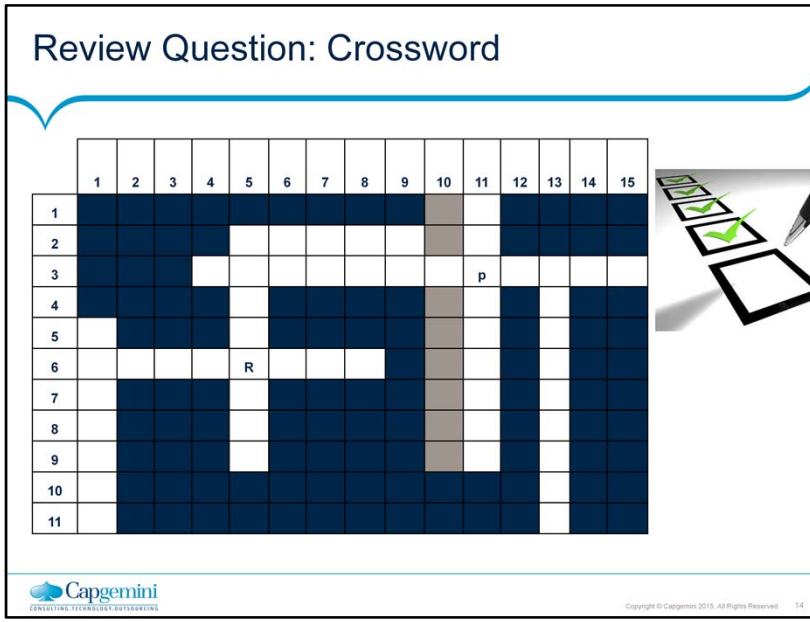
## Review Question

- 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.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 13




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)

# Object Oriented Programming

Lab Book

# Document Revision History

| Date | Revision No. | Author | Summary of Changes |
|---|---|---|---|
| 18-Dec-2008 | 0.1D | Veena Deshpande | Creation |
| 22-Dec-2008 | | CLS team | Review |
| Jan-2009 | 1.0 | Veena Deshpande | Baselined |
| 8 May 2009 | 1.1 | Veena Deshpande | Updated for including comments related to tool usage for Labs. |
| 20-May-11 | 1.2 | Latha S | Added the OOP exercises |

# Table of Contents



# Getting Started

## Overview

This lab book comprises of assignments to be done for OOP. The course is focused towards understanding Object Oriented Concepts. If the participants are not familiar with UML, it is recommended to the trainers to explain the notations of class diagrams and relationship. The participants can use pen and paper to depict the classes.

## Setup Checklist for OOP

- None

## Instructions

- Specified for each of the individual assignments.

## Learning More (Bibliography)

- Thinking in Java/C++ - First chapter
- Head First Object Oriented Analysis and Design
- Applying UML – Advanced Applications by Rob Pooley , Pauline Witcox
- UML User's Guide by Grady Booch, Ivar Jacobson and James Rambaugh

# Lab 1. Introduction to Classes and Objects

| | |
|---|---|
| **Goals** | Identifying potential objects from a given business scenario<br>Creating classes<br>Distinguish between objects and classes |
| **Time** | 20-30 minutes |

1.1. Given a university scenario

i. There are several courses offered by the university, for example Discrete Mathematics, Database & Information Systems, Islamic History etc
ii. Each course has several sections (for example Discrete Mathematics has Graph Theory ,Information Systems has ORDBMS , OLAP and Distributed systems etc
iii. All the courses have unique course codes and duration with specific pre requisite
iv. A course section is handled by a Professor
v. A professor is assigned a course section based on his experience and his qualification.
vi. Each course is handled in a specific location /room like lecture halls , workshops , seminar rooms etc
vii. Each course begins with a pre test to check if students meet the pre requisite, mid level tests and end semester tests , which can be theoretical, practical or thesis
viii. Marks are collated to calculate the grade.

Identify the objects and classes here.

1.2. One of the team has identified **Discrete Mathematics** as a class. Give suitable justification whether you agree or not agree. Discuss the solution with your instructor and team mates.

# Lab 2. Writing down class structure

| | |
|---|---|
| **Goals** | • Write classes with properties and functionalities (attribute and behaviors)<br>• Write classes with overloaded constructors |
| **Time** | 15-20 minutes |

1.1. Now that you have identified the classes in previous questions, list down the structure [attributes, behaviors] of all the identified classes and relevant initialization functions.

1.2. Design a Date class. The class should support the following functionalities

i. Store a specific date
ii. Display the date in a certain format – American (mm/dd/yyyy), British (dd/mm/yyyy), Japanese – dd.mm.yyyy)
iii. Extract out parts of date – day , month , year , day of the week , month name etc
iv. Support manipulation of the date like adding and subtracting days and months.
v. Compare two dates and give the difference in no of days

1.3. Define and implement a class matrix having the following functionality.

i. The matrix can store integer values.
ii. The non-parameterize constructor creates a matrix of 10 rows and 10 columns.
iii. The size of a matrix can be defined through a constructor.
iv. The class allows addition of two matrices.
v. There must be readMatrix and printMatrix function in the class.
vi. The readMatrix function can read from standard input or from file.
vii. The printMatrix function can print on standard output or on file

# Lab 3. Class Relationship

| | |
|---|---|
| **Goals** | • Identify class relationship – Aggregation, Generalization , Association |
| **Time** | 10-15 minutes |

1.1. Meera is designing classes for an online garment shopping application. She has been given the following requirements

i. Registered Customers can only shop
ii. Shop sells a variety of Indian and western garments
iii. Customers can place multiple orders
iv. Each order can have several products
v. Customers make payment via credit card /net banking

Meera has identified the following classes: Customers, Products, Orders. Now she has to identify the class relationships, can you help her?

1.2. Nothing can be more entertaining like a game of cards. Now that most of the card games have become online, design the classes for a Deck of cards, which can be used to design any new card game. Here is a small introduction to all those non card game players

i. Deck means a pack of cards, usually 54 with 2/3 joker cards. These joker cards can be substituted as any card depending on game
ii. Each card has the following symbols and colors
    1. Red – Diamond, Hearts
    2. Black - Spade, Clover or Clubs
iii. Each set has 13 cards starting from numbers 2 – 10 then K (King) , Q (Queen) , J (Jack) and A (Ace)

# Lab 4. Abstract classes , Polymorphism

| | |
|---|---|
| **Goals** | • Identify concrete and abstract classes<br>• Identify polymorphic methods |
| **Time** | 5-10 minutes |

1.1. Today, mobile phones support different kinds of messages like Text message, audio message, voice messages, multimedia messages etc. All messages have common information like sender details, receiver details, content, size etc, differencing only in rendering. Design the classes along with relationships and overloadable and overridable methods.

# Lab 5. Consolidated Exercise

| | |
|---|---|
| **Goals** | • Identify class hierarchy<br>• Identify static members |
| **Time** | 10-15 minutes |

1.1. There are two types of employees in our organization: Regular and Contract. Regular employees are on rolls of the organization and are entitled for various allowances and benefits like HRA, Conveyance, Medical reimbursements, LTA and Special allowances, along with basic salary. Regular Employee is also entitled for 10 days of CL and 30 days of PL every year. The contract employees on the other hand are entitled for Basic and Conveyance. Contract Employee is also allotted 1 day of leave every month and no other. As per government rules, PF is 12 % of Basic and is deducted from all employees (employers contribution being 13 %). Tax is deducted from all employees as applicable depending on the gross salary of the employees.

Design the class hierarchy. You may have additional classes also as applicable.