Contents

[Questions 1](#_Toc499883502)

[Answers 1](#_Toc499883503)

[What are main features of OOP? 1](#_Toc499883504)

[Advantages of OOP 1](#_Toc499883505)

[Abstraction 2](#_Toc499883506)

[**Encapsulation** 4](#_Toc499883507)

[Polymorphism 4](#_Toc499883508)

[Generalization 4](#_Toc499883509)

[Specialization 5](#_Toc499883510)

[*Object/Class/State/Instance/Method* 5](#_Toc499883511)

[**Object** 5](#_Toc499883512)

[**Class** 5](#_Toc499883513)

[**Instance** 5](#_Toc499883514)

[**State** 5](#_Toc499883515)

[**Method** 5](#_Toc499883516)

Questions

1. Real-world objects contain \_\_\_ and \_\_\_.
2. A software object's state is stored in \_\_\_.
3. A software object's behavior is exposed through \_\_\_.
4. Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as data \_\_\_.
5. A blueprint for a software object is called a \_\_\_.
6. Common behavior can be defined in a \_\_\_ and inherited into a \_\_\_ using the \_\_\_ keyword.
7. A collection of methods with no implementation is called an \_\_\_.
8. A namespace that organizes classes and interfaces by functionality is called a \_\_\_.

Answers

1. Real-world objects contain **state** and **behavior**.
2. A software object's state is stored in **fields**.
3. A software object's behaviour is exposed through **methods**.
4. Hiding internal data from the outside world and accessing it only through publicly exposed methods is known as data **encapsulation**.
5. A blueprint for a software object is called a **class**.
6. Common behavior can be defined in a **superclass** and inherited into a **subclass** using the **extends** keyword.
7. A collection of methods with no implementation is called an **interface**.
8. A namespace that organizes classes and interfaces by functionality is called a **package**.

What are main features of OOP?

* **Inheritance**
* **Abstraction**
* **Encapsulation**
* **Polymorphism**

## Advantages of OOP

Object-Oriented Programming has the following advantages over conventional approaches:

1. OOP provides a clear modular structure for programs which makes it good for defining abstract data types where implementation details are hidden and the unit has a clearly defined interface.
2. OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
3. OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer.

# Abstraction

"Abstraction is accomplished using an Interface. Just giving the abstract information about what it can do without specifying the details."

* Abstraction is "To represent the essential feature without representing the background details."
* Abstraction lets you focus on what the object does instead of how it does it.
* Abstraction provides you a generalized view of your classes or objects by providing relevant information.
* Abstraction is the process of hiding the working style of an object, and showing the information of an object in an understandable manner.  
    
  **Real-world Example of Abstraction**

Suppose you have an object Mobile Phone.  
  
Suppose you have 3 mobile phones as in the following:   
  
Nokia 1400 (Features: Calling, SMS)  
Nokia 2700 (Features: Calling, SMS, FM Radio, MP3, Camera)  
Black Berry (Features:Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)  
  
Abstract information (necessary and common information) for the object "Mobile Phone" is that it makes a call to any number and can send SMS.

**Example**

**abstract class MobilePhone {**

**public void Calling();**

**public void SendSMS();**

**}**

**public class Nokia1400: MobilePhone {}**

**public class Nokia2700: MobilePhone {**

**public void FMRadio();**

**public void MP3();**

**public void Camera();**

**}**

**public class BlackBerry: MobilePhone {**

**public void FMRadio();**

**public void MP3();**

**public void Camera();**

**public void Recording();**

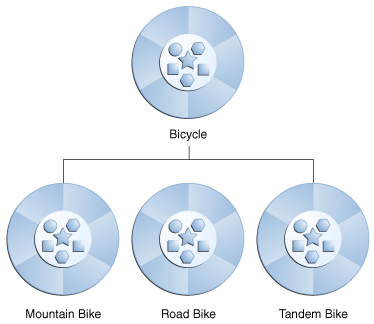
**public void ReadAndSendEmails();**

**}**

**What is the purpose of inheritance? What is** [**Inheritance**](http://en.wikipedia.org/wiki/Inheritance_%28object-oriented_programming%29)**?**   
The purpose of inheritance is Code Reuse.

Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes. In this example, Bicycle now becomes the *superclass* of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*:

Different kinds of objects often have a certain amount in common with each other. Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear). Yet each also defines additional features that make them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.



public class ParentClass {

public ParentClass() {

Console.WriteLine("Parent Constructor.");

}

public void print() {

Console.WriteLine("I'm a Parent Class.");

}

}

public class ChildClass: ParentClass {

public ChildClass() {

Console.WriteLine("Child Constructor.");

}

public static void Main() {

ChildClass child = new ChildClass();

child.print();

}

}

# **Encapsulation**

"Encapsulation is accomplished using classes. Keeping data and methods that access that data into a single unit."

"Information/Data hiding is accomplished using modifiers by keeping the instance variables private or protected."

* Wrapping up a data member and a method together into a single unit (in other words class) is called Encapsulation.
* Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object
* Encapsulation is like your bag in which you can keep your pen, book etcetera. It means this is the property of encapsulating members and functions.

**Real-world Example of Encapsulation**

Let's use as an example Mobile Phones and Mobile Phone Manufacturers.  
Suppose you are a Mobile Phone Manufacturer and you have designed and developed a Mobile Phone design (a class). Now by using machinery you are manufacturing Mobile Phones (objects) for selling, when you sell your Mobile Phone the user only learns how to use the Mobile Phone but not how the Mobile Phone works.  
  
This means that you are creating the class with functions and by with objects (capsules) of which you are making available the functionality of your class by that object and without the interference in the original class.

# Polymorphism

Polymorphism means the ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends on the data types used in the operation.

It is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. In general, polymorphism means "one interface, multiple methods", This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler's job to select the specific action (that is, method) as it applies to each situation.

In the example below of polymorphism as an OOPs concept, we have two classes: Person and Employee. The Employee class inherits from the Person class by using the keyword **extends**. Here, the child class overrides the parent class

class Person {

void walk() {

System.out.println(“Can Run….”);

}

}

class Employee extends Person {

void walk() {

System.out.println(“Running Fast…”);

}

public static void main(String arg[]) {

Person p = new Employee(); //upcasting

p.walk();

}

}

# Generalization

Generalization describes an is-a relationship which represent a hierarchy between classes of objects. Eg:- a "fruit" is a generalization of "apple", "orange", "mango" and many others. animal is the generalization of pet.

# Specialization

Specialization means an object can inherit the common state and behavior of a generic object. However, each object needs to define its own special and particular state and behavior. Specialization means to subclass. animal is the generalization and pet is the specialization, indicating that a pet is a special kind of animal.

# *Object/Class/State/Instance/Method*

### **Object**

Objects are the basic run-time entities in an object-oriented system. Programming problem is analyzed in terms of objects and nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data or code.

An object is an instance of a class. A class must be instantiated into an object before it can be used in the software. More than one instance of the same class can be in existence at any one time.

### **Class**

A class is a collection of objects of a similar type. Once a class is defined, any number of objects can be created which belong to that class. A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.

### **Instance**

The instance is the actual object created at runtime. One can have an instance of a class or a particular object.

### **State**

The set of values of the attributes of a particular object is called its state. The object consists of state and the behaviour that's defined in the object's class

### **Method**

Method describes the object’s abilities. A Dog has the ability to bark. So bark() is one of the methods of the Dog class.