Object Oriented Programming (OOP)

Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

Object

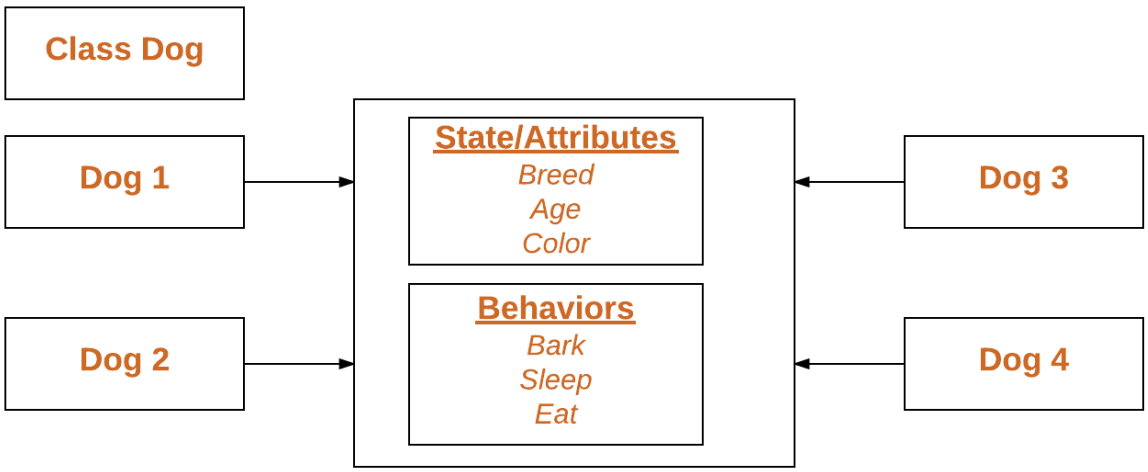
It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

- **State**: It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances. Example:

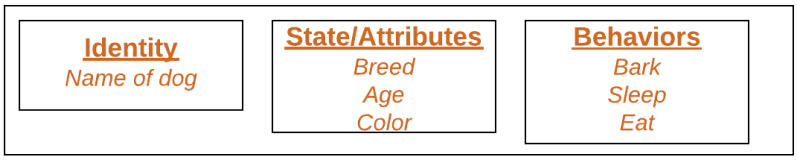


As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general, we can't create objects of an abstract class or an interface.
Dog tuffy;

If we declare reference variable(tuffy) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

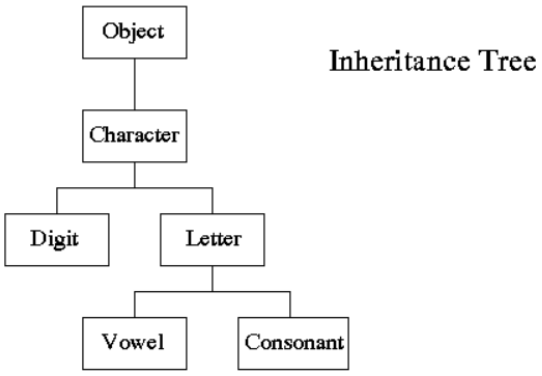


Objects correspond to things found in the real world. For example, a graphics program may have objects such as "circle", "square", "menu". An online shopping system might have objects such as "shopping cart", "customer", and "product".

Inheritance

It works by letting a new class adopt the properties of another. We call the inheriting class a **subclass** or a **child class**. The original class is often called the **parent**. We use the keyword **extends** to define a new class that inherits properties from an old class.

Class Hierarchy



Inheritance Tree

The classes form a class hierarchy, or inheritance tree, which can be as deep as needed. The hierarchy of classes in Java has one root class, called Object, which is superclass of any class. Instance variable and methods are inherited down through the levels. In general, the further down in the hierarchy a class appears, the more specialized its behavior. When a message is sent

to an object, it is passed up the inheritance tree starting from the class of the receiving object until a definition is found for the method. This process is called upcasting. For instance, the method toString() is defined in the Object class. So every class automatically has this method. If you want that your particular toString() method looks differently, you can reimplement it in your class. In this way you can override a method in a given class by redefining it in a subclass.

Polymorphism

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, so it means many forms.

Method Overloading

When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

Scoping

The scope of a name binding—an association of a name to an entity, such as a variable—is the part of a program where the name binding is valid, that is where the name can be used to refer to the entity. In other parts of the program the name may refer to a different entity (it may have a different binding), or to nothing at all (it may be unbound). The scope of a name binding is also known as the visibility of an entity, particularly in older or more technical literature—this is from the perspective of the referenced entity, not the referencing name. The term "scope" is also used to refer to the set of all name bindings that are valid within a part of a program or at a given point in a program, which is more correctly referred to as context or environment.

Data Hiding

As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of a class private, and the class is exposed to the end-user or the world without providing any details behind implementation using the abstraction concept, so it is also known as a **combination of data-hiding and abstraction**.

Accessibility levels

Four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Abstract classes and interfaces.

What is Interface?

The interface is a blueprint that can be used to implement a class. The interface does not contain any concrete methods (methods that have code). All the methods of an interface are abstract methods.

An interface cannot be instantiated. However, classes that implement interfaces can be instantiated. Interfaces never contain instance variables but, they can contain public static final variables (i.e., constant class variables)

- Interfaces are used to achieve abstraction.
- Designed to support dynamic method resolution at run time
- It helps you to achieve loose coupling.
- Allows you to separate the definition of a method from the inheritance hierarchy

What Is Abstract Class?

A class which has the abstract keyword in its declaration is called abstract class. Abstract classes should have at least one abstract method. , i.e., methods without a body. It can have multiple concrete methods.

Abstract classes allow you to create blueprints for concrete classes. But the inheriting class should implement the abstract method.

Abstract classes cannot be instantiated.

- Abstract classes offer default functionality for the subclasses.
- Provides a template for future specific classes
- Helps you to define a common interface for its subclasses
- Abstract class allows code reusability.

Abstract class vs Interface

An abstract class permits you to make functionality that subclasses can implement or override whereas an interface only permits you to state functionality but not to implement it. A class can extend only one abstract class while a class can implement multiple interfaces.

It is better to use interface when various implementations share only method signature. Polymorphic hierarchy of value types. It should be used when various implementations of the same kind share a common behavior.

Type members.(?)

All programs are composed of two items: Data and Operations on that Data. Because, at their heart, computers are simple devices, they can only represent very simple pieces of information. All complex information must be built up from these basic Data Types. The data types can roughly be described as: numbers, booleans, characters, arrays, and structures. Some languages like ActionScript replace characters with "strings". Object oriented languages, such as C++ and Java replace "structures" with "objects".

Type members define storage locations and executable code. They can be methods, constructors, events, constants, variables, and properties.