First OOP Language

Objects were invented in the design of Simula and refined in the evolution of Smalltalk.

**Object-oriented programming in Simula,** -

based on the concept of a procedure that returns a pointer to its activation record.

The development of a purely object-oriented paradigm in the Smalltalk project and programming language.

Twenty years after its development, Smalltalk provides an important contrast with C++ and Java both in simplicity of concept and in the way that its implementation provides maximal programming flexibility.



Object

**Object** means ***a real-world entity***.

An Object can be defined as ***an instance of a class***.

Any entity that has ***state*** and ***behaviour*** is known as an object.

For example, a chair, pen, table, keyboard, bike, etc.

It can be ***physical or logical.***

An ***object contains an address and takes up some space in memory***. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

**Collection of objects** is called class. It is a logical entity.

A class can also be defined as a ***blueprint*** from which you can create an individual object. Class ***doesn't consume any space***.

Inheritance

**When one object acquires all the properties and behaviours of a parent object,** it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Inheritance represents the **is-a** relationship

Polymorphism

If **one task is performed in different ways**, it is known as polymorphism.

For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use **method overloading** to achieve **compile time polymorphism**.

In Java, we use **method overriding** to achieve **run time polymorphism.**

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

**Hiding internal details and showing functionality** is known as abstraction.

For example, phone call, we don't know the internal processing.

In Java, we use **abstract class** and **interface** to achieve abstraction.

Encapsulation

**Binding (or wrapping) code and data together into a single unit** are known as encapsulation*.* For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Coupling

Coupling refers to the ***knowledge/ information/ dependency of another class***. It arises when classes are aware of each other. If a ***class has the details information of another class***, there is ***strong coupling***. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use ***interfaces*** for the ***weaker coupling*** because there is no concrete implementation.

Cohesion

Cohesion refers to the ***level of a component which performs a single well-defined task***. A ***single well-defined*** ***task*** is done by a ***highly cohesive*** method. The ***weakly cohesive method*** will ***split the task into separate parts***. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

Association

Association represents the ***relationship between the objects***. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

***One to One | One to Many | Many to One | Many to Many***

For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be unidirectional or bidirectional.

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the ***weak relationship***between objects. It is also termed as a **has-a** relationship in Java. It is another way to reuse objects.

Both entity exists independent of each other they come together to achieve some results.

If one of the entity is deleted other entity still exists

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a ***strong relationship*** between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

Part of, integral part of

Both entities exist depending on each other If one of the entity is deleted other entity also get deleted.

Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

Difference between method overloading and method overriding in java

|  |  |
| --- | --- |
| Method Overloading | Method Overriding |
| Method overloading is used *to* ***increase the readability***of the program. | Method overriding is used *to* ***provide the specific implementation***of the method that is already provided by its super class. |
| Method overloading is performed ***within class***. | Method overriding occurs ***in two classes*** that have ***IS-A (inheritance) relationship.*** |
| In case of method overloading, ***parameter must be different*.** | In case of method overriding, ***parameter must be same***. |
| Method overloading is the example of ***compile time polymorphism*.** | Method overriding is the example of ***run time polymorphism*.** |
| In java, method overloading can't be performed by changing return type of the method only. ***Return type can be same or different*** in method overloading. But you must have to change the parameter. | ***Return type must be same or covariant*** in method overriding. |

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

Simply, ***abstract class*** achieves ***partial abstraction (0 to 100%)*** whereas ***interface*** achieves ***fully abstraction (100%).***

|  |  |
| --- | --- |
| Abstract class | Interface |
| 1) Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods.  Since Java 8, it can have default and static methods also. |
| 2) Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| 3) Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| 4) Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class. |
| 5) The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| 6) An abstract class can extend another Java class and implement multiple Java interfaces. | An interface can extend another Java interface only. |
| 7) An abstract class can be extended using keyword "extends". | An interface can be implemented using keyword "implements". |
| 8) A Java abstract class can have class members like private, protected, etc. | Members of a Java interface are public by default. |

**Example: Abstract Class**   
public abstract class Shape{  
public abstract void draw();  
}

**Example: Interface**  
public interface Drawable{  
void draw();  
}

**NOTE**

**Sequence dig**: shows time dependent changes

**Use Case dig**: shows overall view of system

**state dig**: shows state changes (transitions)

**class dig**: shows detailed relationship among the objects

