# JAVA OOP

Purpose or OOP languages is to implement real word entities

OOP language over procedural

* It’s easy to develop and support applications over procedural language
* OOP provides data hiding. Whereas in procedural language global data can be accessed anywhere
* Make code reuale

**CLASS** is blue print from which you can create [objects]

Class is logical entity

**OBJECT** is runtime entity that has state and behavior. Object is created from the class or instance of class. Everything that surrounds us is ab object (pen, table, chair, car)

For example, dog is an object.

* It has state - color, name, whiskers
* It has behavior – barking, eating, wagging tail

Object has address to the memory and therefore it takes up some space

Object is physical entity

Object has:

* **state** - represent data (value) of an object
* **behavior** – represents functionality (deposit, withdraw)
* **identity** – is ID. It is not visible to external users. It’s used internally by JVM to identify each object uniquely

[**new**] keyword is used to allocate memory at runtime. All objects get memory in [Heap memory] are

**Declaration -> Instantiation-> Initialization**

**1.Declaration** assign type to the variable

Person p;

**2.Instantiation** means to creating new object. It means allocation memory at runtime in [Heap memory ] area

Person p = new Person();

Person p = new Person(“Alex”);

**3.Initialization** – passing parameters to constructor to define

public class Person

{

private String name;

private String color;

public Person(String name)

{…}

public Person(String name, String color)

{…}

}

# INITIALIZE OBJECT

There are 3 ways to initialize object:

* by reference variable
* by method
* by constructor

# INHERITANCE

**INHERITANCE** – is OOP concept. When one object acquires all the properties and behaviors of parent class it’s called inheritance.

It’s used to achieve [runtime polymorphism]

# POLYMORFISM

**POLYMORFISM** is when task can be performed in different ways

**POLYMORFISM** is the ability of object takes different forms

Polymorphism:

* static binding
* dynamic binding

# ADBSTRACTION

**ABSTRACTION** is hiding details and showing only functionality

# COUPLING

**COUPLING** refers to dependency to another class. You can use interface to achieve weakly coupling, because it does not have concrete implementation.

# COHESION

Weakly cohesive method will split the task into separate parts

# ASSOCIATION

**ASSOCIATION** represents relationship between objects:

* 1:1
* 1:n
* n:1
* n:n

# AGGREGATION

**AGGREGATION** (**HAS-A**) is a way to achieve association.

**AGGREGATION** (**HAS-A**) represents the relationship where one object contains others as a part of its own state. It represents a **weak relationship** between objects

represents re

# COMPOSITION

**COMPOSITION**(**IS-A**) is also way to represents the relationship where one object contains others as a part of its own state. It represents a **strong relationship** between objects

|  |  |
| --- | --- |
| **HOW ACHIEVE CONCEPT** | **DESC** |
| abstraction | Using abstract class and inheritance |
| runtime polymorphism | Inheritance (reference variables) |
|  |  |
|  |  |