Java OOPs:

1) **Class:**
    1) Is a blueprint through which objects can be created
    2) Fields, methods, constructors etc.
    3) Naming:
        1) Should start with Capital letters
        2) Eg: Employee, EmployeeDetails
    4) It should always contains a java doc at the start stating what are its features
2) **Objects:**
    1) Instance of a class
    2) Object is an entity which has a state and behaviour
3) **Methods:**
    1) Functions
    2) Define behaviour of the class
    3) Naming:
        1) Starts with small letters and flows camel-casing:
            1) Eg: setSalary()
    4) Should have their own java docs
4) **Packages:**
    1) Naming:
        1) All small case letters
5) **Constructors:**
    1) Is used to assign the class field/initialise the class states while creation
    2) When no constructor is defined in the class then Java provides us with the default constructor
        1) Person p = new **Person()**;
    3) A constructor should be public, to enable the class to create objects, because if constructor is private it cannot be accessed
    4) Syntax:
        1) public <<class-nam>>(<<field-names>>){}
    5) When we have define any constructor the default constructor that was provided to us by Java is not available anymore. We have to explicitly write the no-argument constructor
    6) Constructor Overloading:
        1) A class can have multiple constructors but with a different set of arguments
6) **Inheritance:**
    1) It is a mechanism in java in which one object acquires all the behaviours of the parent object

2) Inheritance -> **IS-A** relation:
    1) Eg: Dog **IS A** Animal
3) Enhances code reusability
4) Type of Inheritance:
    1) Single
    2) Multiple: Not supported in Java:
        1) It may happen that both the parent classes have same method name, creating confusion while overriding during runtime.
    3) Hierarchical
    4) Multi-level
    5) Hybrid: Not supported in Java
5) Inheritance we use **extends** keyword
    1) <<child-class>> extends <<parent-class>>

7) Polymorphism:

1) Multiple-Forms
2) Types:
    1) Compile-Time:
        1) Method Overloading
    2) Run-Time:
        1) Method Overriding:
            1) We can either assign the same visibility as the parent method or the greater visibility
                1) Eg:
                    1) Animal.java -> boolean layEggs(){}
                    2) Bird.java -> we can either assign the same visibility i.e. fault or greater visibility i.e. protected and public
            2) Final Methods cannot be overridden
            3) In static methods I cannot use @Override annotation-> method hiding
            4) We cannot override a constructor because different classes cannot have same constructor names
            5) Parent method can be called in overriding methods using **super** keyword