#OOPs In JAVA

Introduction To Object-Oriented Programming

→ An Object-oriented program is made of objects. Each object has a specific functionality, exposed to its user and a hidden implementation

→ Traditional structured programming consists of designing a set of procedures (or algorithms) to solve a problem. Once the procedures are determined, the next step would be to find ~~appropi~~ appropriate ways to store the data.
OOP reverse the order : Put the data first then looks at the algorithms to operate on the data.

#Classes.

→ A class is the template or blueprint from which objects are made.
When you construct an object from a class, you are said to have created an instance of the class.

Encapsulation

→ Sometimes called information hiding → is a key concept in working with objects. formally, encapsulation is simply combining data and behavior in one package and hiding the implementation details from the user of the object. The bit of data in an object are called its instance fields, and the procedures that operate on the data are called its methods.

A specific object ~~of~~ that is an instance of operate ~~on the~~ ~~data are~~ called ~~co~~ class will have Specific values of its instance fields. The set of those values is the current state of the object.

The key to making encapsulation work is to have methods never directly access instance fields in a class other than their own.

Objects.

→ To work with OOP, you should be able to identify three key characteristics of objects.

•) The object's behaviour → What can you do with this object, or what methods can you apply to it?

•) The object's state → How does the object react when you invoke those methods?

•) The object's identity → how is the object distinguished from others that may have the same behaviour and state

→ All objects that are instance of the same class share a family resemblance by supporting the same behavior.

→ ~~All~~ obj

→ Each object stores information about what it currently look like (object's state). It may change.

→ Each object has distinct identity.
For example :- In a order processing system, two order are distinct even if they request identical items.

→ The state of an object can influence its behavior (If an order is "shipped" or "paid", it may reject a method call "add" or "remove" items. ~~Conversely~~

identifying classes.

→ In a procedural program, you start the process at the top, with the main function. When designing an object - oriented system, there is no "Top" ~~and~~
→ Identify your classes and then add methods to each class.

→ Rule for identifying classes is to look for nouns in the problem analysis. Methods, on the other hand, correspond to verbs.

Relationship Between Classes.

→ Common Relationship b/w classes are :

•) Dependence("uses - a")

•) Aggregation("has - a")

•) Inheritance("is - a")

→ The dependence, or "uses-a" relationship, is the most obvious and also the most general.

Ex:→ The "order" class uses the "Account" class bcoz the "Order" objects need to access "Account" objects to check for credit status. But the "Item" class does not depend on the "Account" class, bcoz "Item" objects never need to worry about customer class.

Note:→
The point is, if class A is unaware of the existence of a class B, it is also unconcerned about any changes to B (changes to B do not introduces bugs are into A) hence minimize the coupling b/w classes.

→ The aggregation, or "has-a" relationship

Eg:→ an "order" object contains "Items" objects. Containment means that objects of class A contain objects of class B.

→ The inheritance, or "is-a" relationship
A relationship b/w a more special and a more general class.
Ex:- a "RushOrder" class inherits from an "Order" class. the specialized "RushOrder" class has special methods for priority handling and a different methods for computing shipping charges, but its other methods such as adding items and billing are inherited from the order class.
In general, if class A extends class B, class A inherits methods from class B but has more capabilities.

2