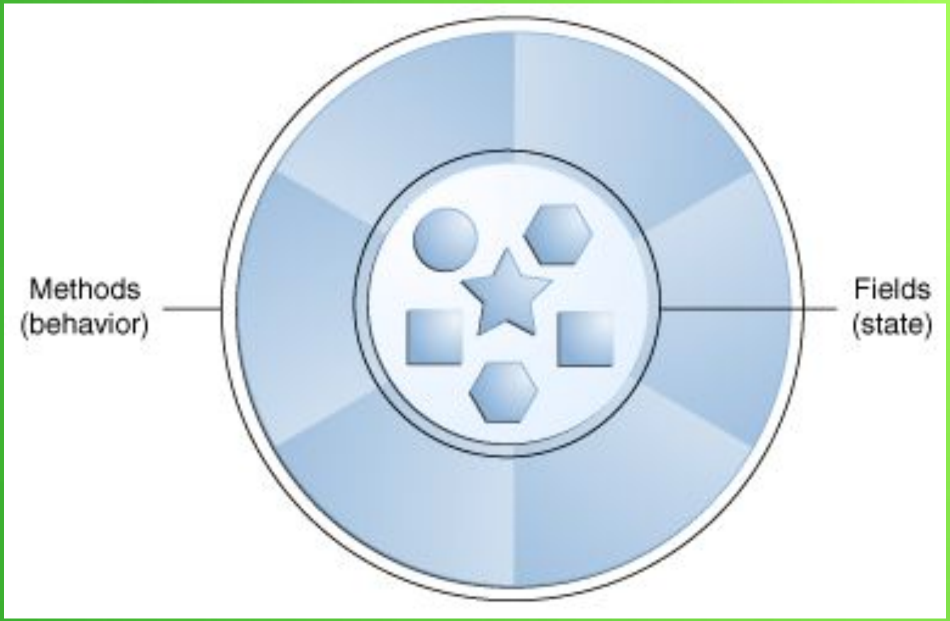
Java OOP/OOD concepts

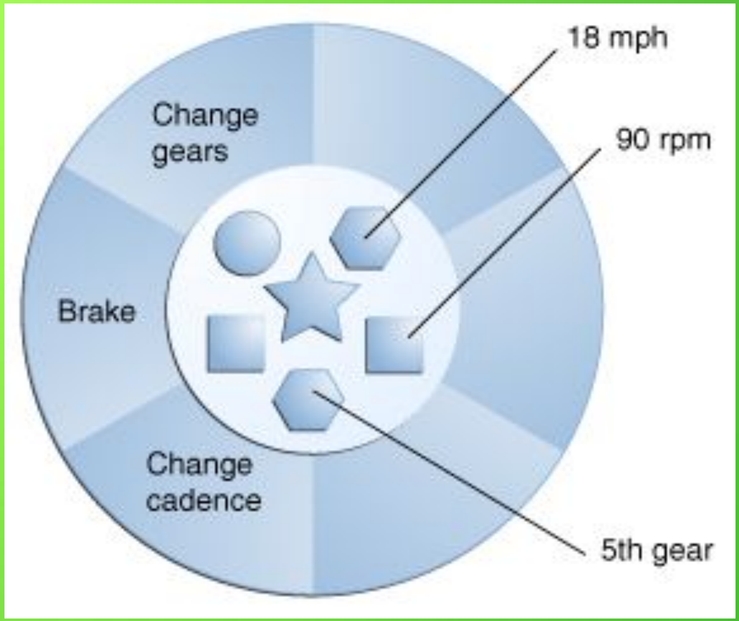
Main points

- What is an object ?
- What is a class ?
- What are messages ?
- What are S.O.L.I.D. principles ?

Object

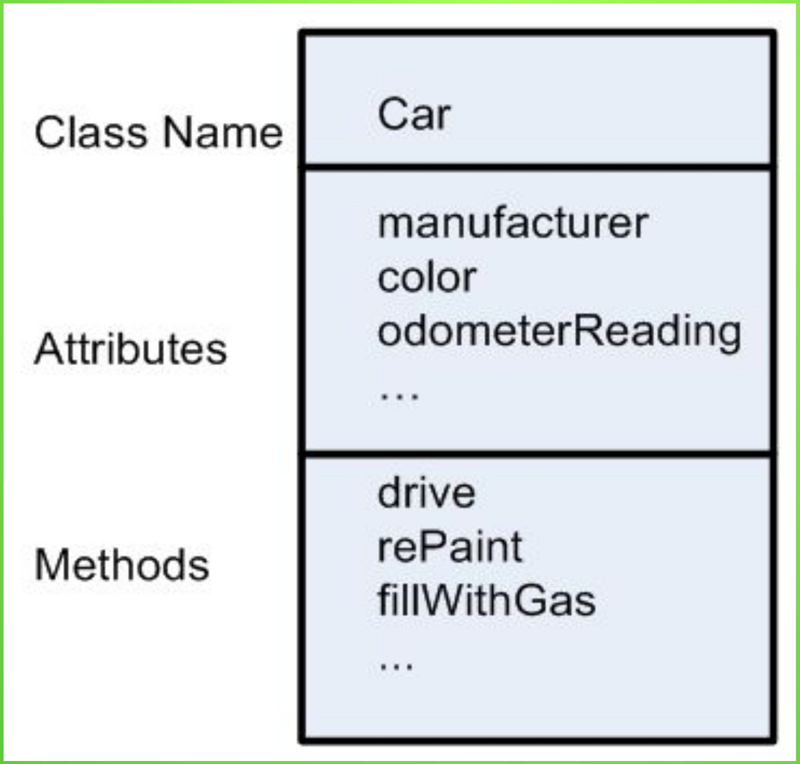
- An object is an instance of a class.
- Objects have states and behaviors.





- *Best practice: object should have an interface*

Class

- A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.
- 

Messages

- Objects interact and communicate with each other using messages. You are able to send message to object using object method.

  - Key things:
    - Send messages using method of object
    - Pass message parameters using method arguments.

Best practice: Use interface or any abstract data types in order to perform messaging between object

Inheritance

- Inheritance, therefore, defines an "is a" hierarchy among classes, in which subclass inherits from one or more superclasses. This is in fact the litmus test for inheritance. Given classes A and B, if A "is not a" kind of B, then A shouldn't be a subclass of B.
- **Use inheritance only if you have "IS A" relationship.**

Best practice: Use composition over inheritance if possible.

Polymorphism

- Polymorphism is the ability of an object to take on many forms.
- Polymorphism allows us to re-use code, and keep some parts of code as unchangeable.

- *Best practice: Use abstract data types over concrete*
- *implementation.*

Encapsulation

Change state of object using methods provided by object.

public - visible to all classes everywhere

no modifier (package-private) - visible only within its own package

protected - accessed within its own package and by a subclass of its class in another package

private - can only be accessed in its own class

Best practices: keep fields as private and change them by object methods, except constants

S.O.L.I.D. principles

- **SRP** - a class should have only a single responsibility
- **OCP** - software entities should be open for extension, but closed for modification.
- **LSP** - client shouldn't know about using object client have to deal with abstraction over this object
- **ISP** - many client-specific interfaces are better than one general-purpose interface
- **DIP** - one should Depend upon Abstractions. Do not depend upon concretions. (related Dependency Injection)

- Live coding