OOPS- a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) based on the concept of "[objects](https://en.wikipedia.org/wiki/Object_%28computer_science%29)", which may contain [data](https://en.wikipedia.org/wiki/Data), in the form of [fields](https://en.wikipedia.org/wiki/Field_%28computer_science%29), often known as attributes; and code, in the form of procedures, often known as [methods](https://en.wikipedia.org/wiki/Method_%28computer_science%29).

Objects – Instantiation of classes.

Classes – They are blue print from which objects are made. describe the type of objects. usable instances of classes.

Encapsulation- Wrapping up data member and method together into a single unit (i.e. Class/Object) is called Encapsulation

Inheritance-When a class acquire the property of another class is known as inheritance.

Abstraction-abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. It is implemented using access modifiers,abstract classes and using properties instead of fields.

Polymorphism- Polymorphism is the ability of an object to take on many forms. A person behaves the son in a house at the same time that the person behaves an employee in an office.

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| 1. Abstraction solves the problem in the design level. | 1. Encapsulation solves the problem in the implementation level. |
| 2. Abstraction is used for hiding the unwanted data and giving relevant data. | 2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world. |
| 3. Abstraction lets you focus on what the object does | 3. Encapsulation means hiding the internal details or mechanics of how an object does something. |
| 4. Abstraction- Outer layout, used in terms of design.  For Example:-   Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number. | 4. Encapsulation- Inner layout, used in terms of implementation.  For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits. |

The “IS A” relationship: Inheritance

If you look at the first requirement (Manager is an employee of XYZ limited corporation), it’s a parent child relationship or inheritance relationship.

Requirement 2 : The use a relationship : Association

It is an interesting requirement (Manager **uses** a swipe card to enter XYZ premises). In this requirement, the **Manager** object and the **Swipecard** object use each other but they have their own object life time. In other words, they can exist without each other.

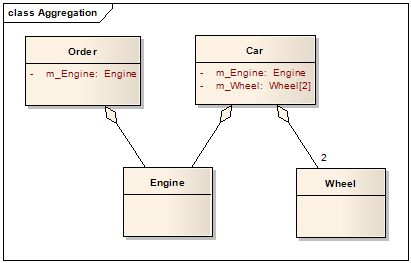


Requirement 3: The “has” relationship with Parent: Aggregation

The third requirement from our list (Manager **has** workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner. So as per the requirement,

The **Manager** object will own **Worker** objects.

But… the **Worker** object can have its own life time which is completely disconnected from the **Manager** object. Looking from a different perspective, it means that if the **Manager** object is deleted, the **Worker** object does not die.



Requirements 4 and 5: The “depends” relationship: Composition

The last two requirements are actually logically one. If you read closely, the requirements are as follows: Project success **depends** on Manager and Manager's incentive **depends** on project success.

So the conclusion from analyzing the above requirements is that Manager and the Project objects are dependent on each other.

The lifetimes of both the objects are the same. In other words, the project will not be successful if the manager is not good, and the manager will not get good incentive if the project fails.



Class Diagram

