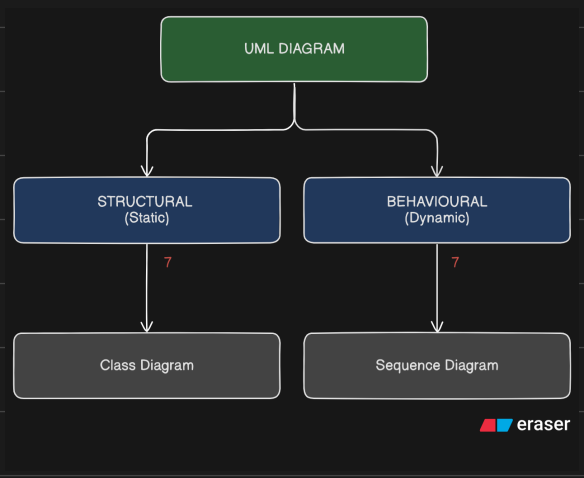
UML DIAGRAM




```
Class Car{
   private:
     string brand;
     string model;
     int enginCC;

   public:
     void StartEngine();
     void StopEngine();
     void Accelerate();
     void Break();
}
```

public	-->	+
protected	-->	#
private	-->	-

<< abstract >>

Car → Class Name

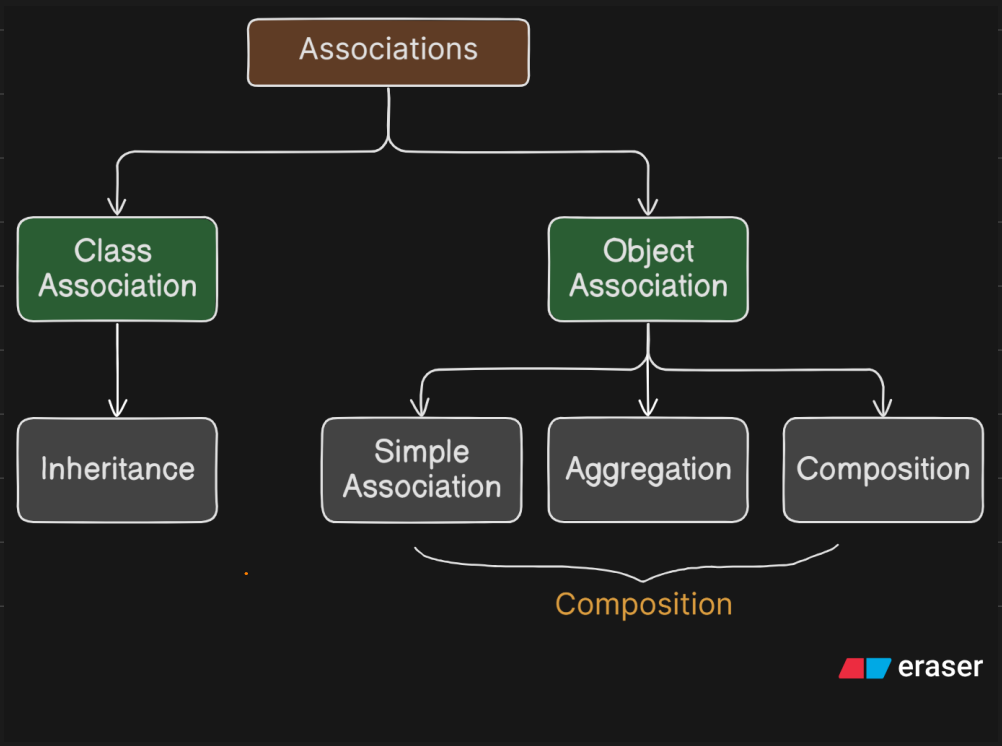
```
# brand: string
# model: string
# enginCC: int
```

→ Charesteristic / Variables

```
- StartEngine(): void
- StopEngine(): void
- Accelerate(): void
- break(): void
```

→ Behaviours / Methods

eraser

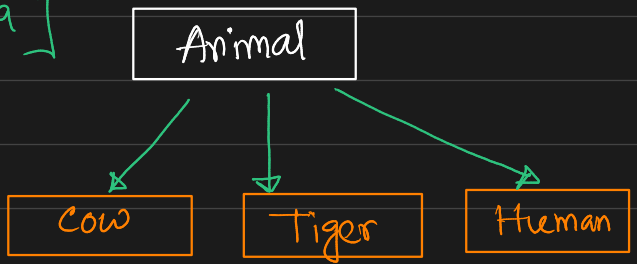


Inheritance : (is – a relationship)

[→ is - a]

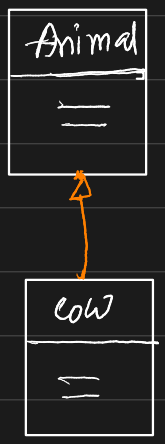
↳ closed arrow

```
class A {
    method1 ();
}
```




```
class B : Public A {
    method2 ();
}
main () {
    B * b = new B ();
    b → method1 ();
    b → method2 ();
}
```

Cow is - a Animal

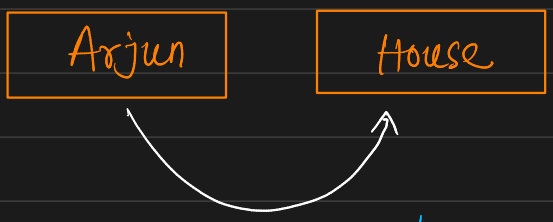


Composition : → (has - a relationship)

[→ has - a]

↳ open arrow

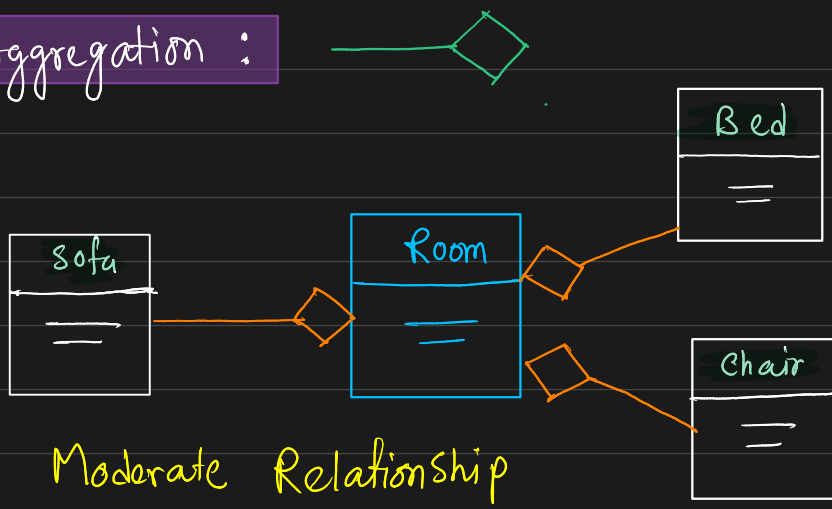
① Simple Association : →



lives in a / has - a

week Relationship

② Aggregation :

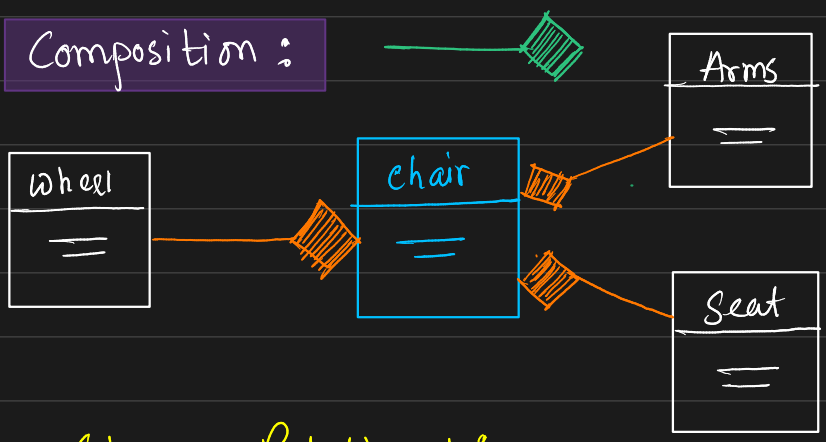


Sofa, Bed, Chair } can exist Separately

Moderate Relationship

Simple Association < Aggregation < Composition

③ Composition :



Wheel, Arms, Seat } Can't exist separately

Strong Relationship

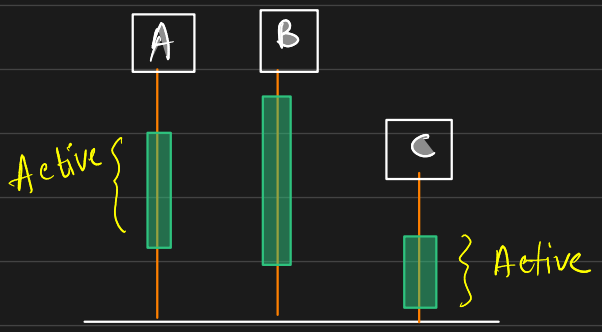
```
class A {
    method 1 ();
}

class B {
    A* a;
    B () { a = new A(); }
    method 2 ();
}
```

```
main () {
    B* b = new B();
    b → method 2 ();
    b → a → method 1 ();
}
```

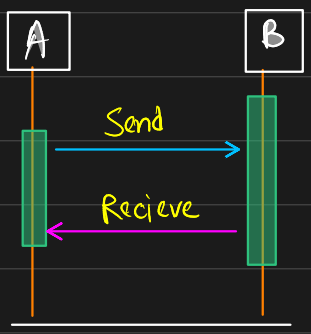
Composition

Sequence Diagram :



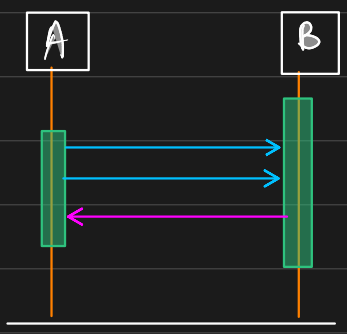
1. A B C
2. Lifeline
3. Activation Bar
4. Message → Sync / Async

Sync Message



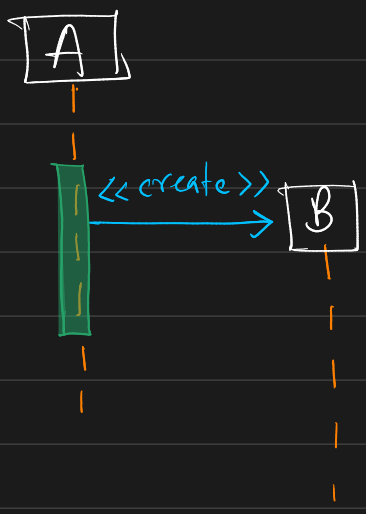
Wait untill message Recieve

Async Message

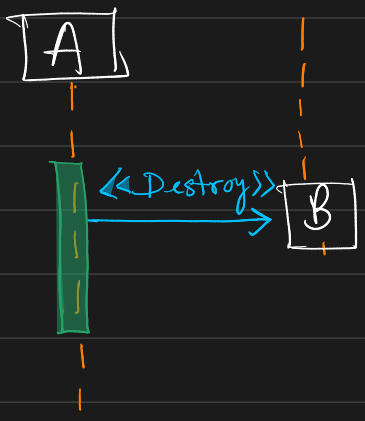


Don't wait for recieve message (continious Send)

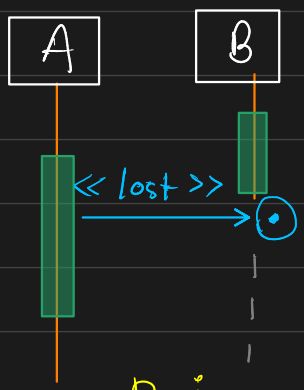
Create Message



Destroy Message

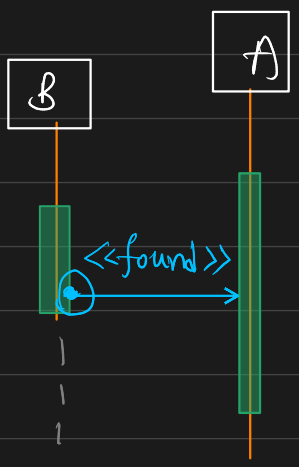


lost Message

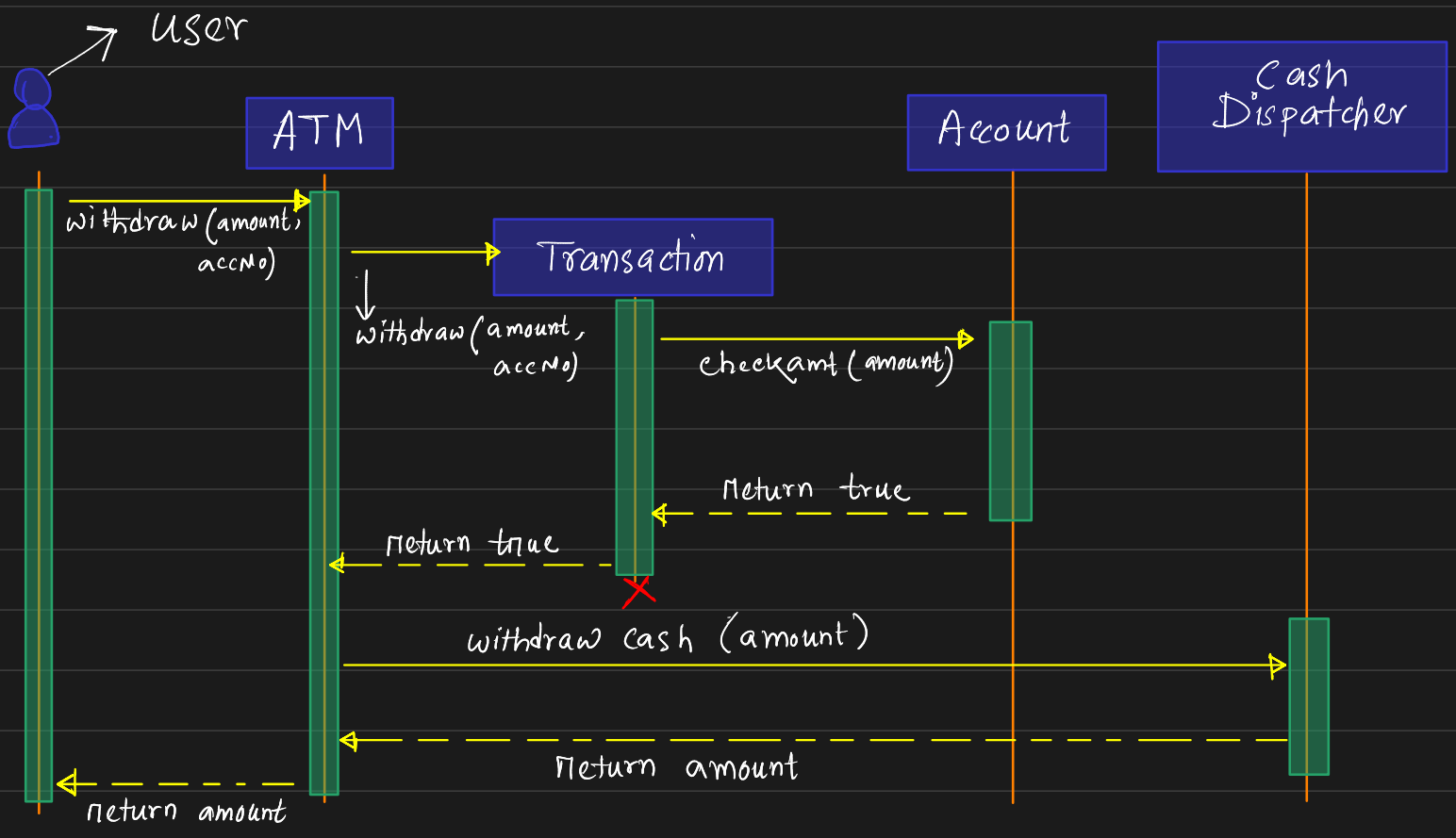


B is not Activate anymore ⟶ Can't recieve message.

Found Message



After sending the message Activation of B is end ⟶ Source of B is not found



Sequence Diagram of ATM Money Withdraw