

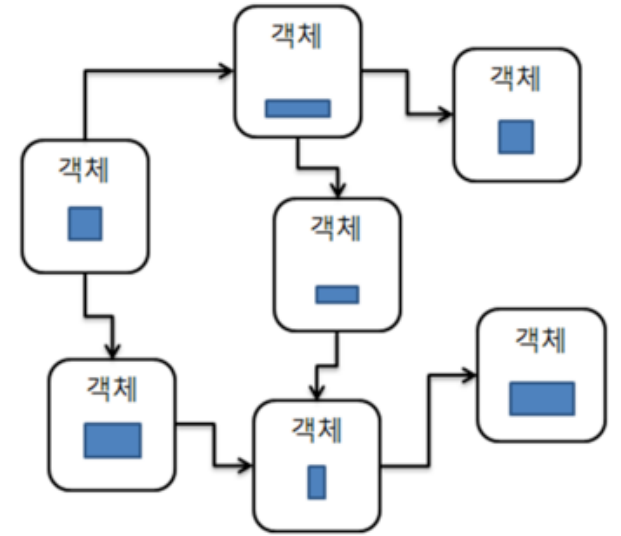
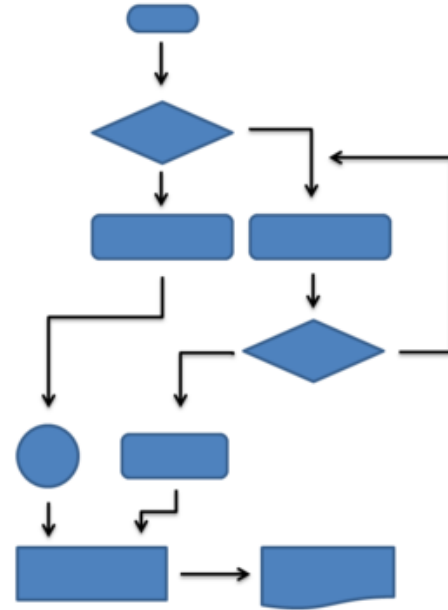
# 객체지향 프로그래밍

---

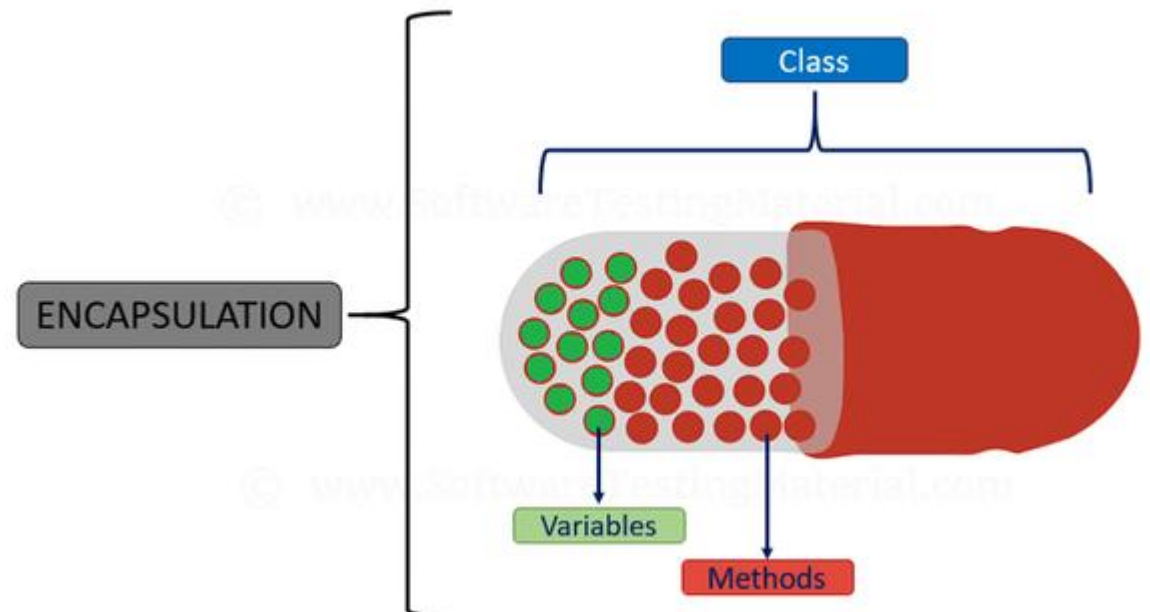
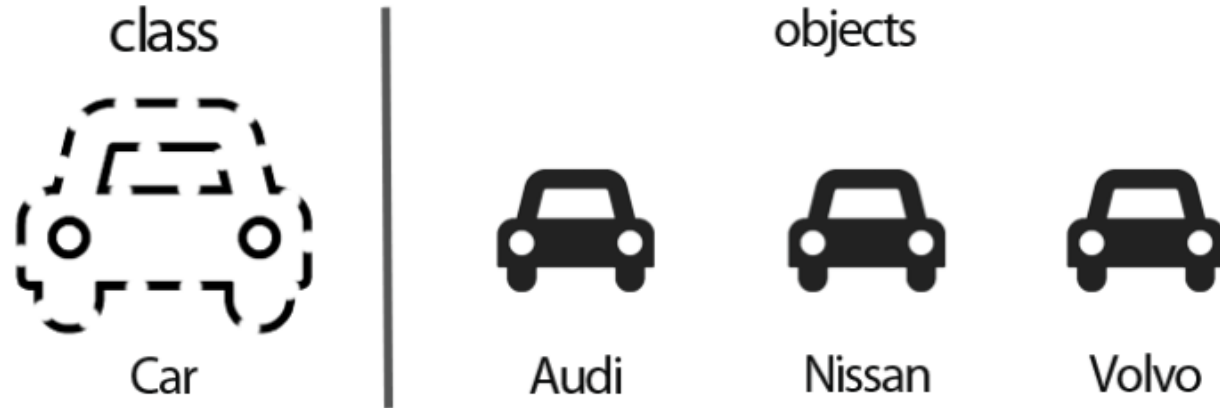
# 01. 객체 지향 프로그래밍이란?

---

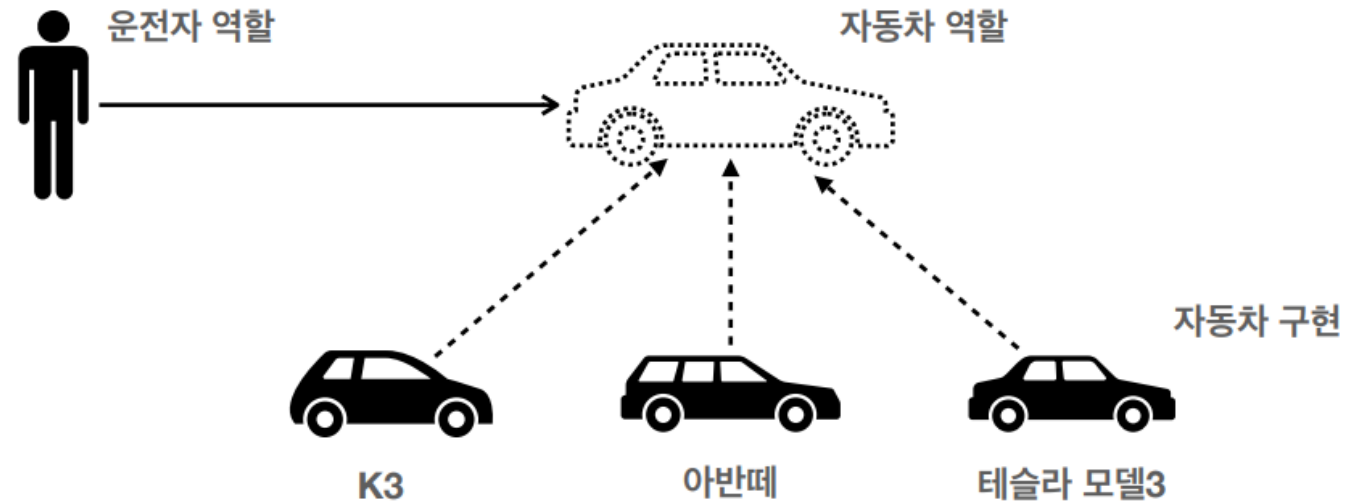
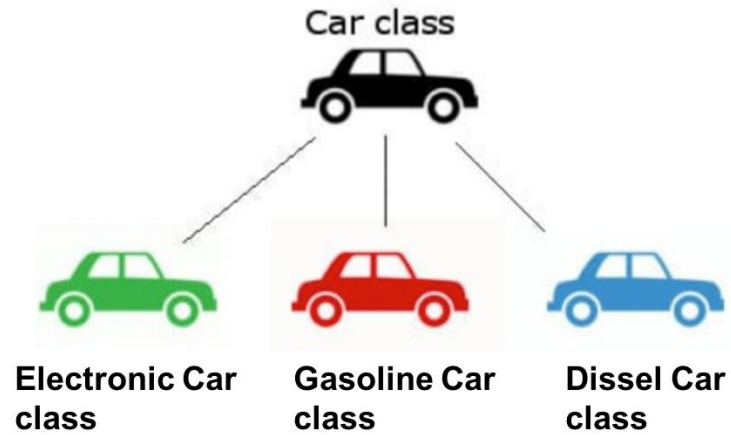
## Object Oriented Programming



## 02. 객체지향 프로그래밍의 특징



## 02. 객체지향 프로그래밍의 특징



## 03. 객체지향 프로그래밍의 설계원칙(SOLID)

---

1. **SRP (Single Responsibility Principle)**
2. **OCP (Open Close Principle)**
3. **LSP (Liskov Substitute Principle)**
4. **ISP (Interface Segregation Principle)**
5. **DIP (Dependency Inversion Principle)**

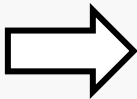
# 04. Why OOP?

---

```
1  let name = 'CAFE'
2
3  ✓ let latte1 = {
4      name: 'Vanilla latte',
5      price: 4500,
6      size: 'tall'
7  }
8
9  ✓ let latte2 = {
10     name: 'caramel latte',
11     price: 5500,
12     size: 'tall'
13 }
14
15 ✓ let latte3 = {
16     name: 'milktea latte',
17     price: 4500,
18     size: 'tall'
19 }
20
```

# 04. Why OOP?

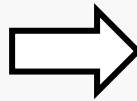
```
1  let name = 'CAFE'
2
3  ✓ let latte1 = {
4    name: 'Vanilla latte',
5    price: 4500,
6    size: 'tall'
7  }
8
9  ✓ let latte2 = {
10   name: 'caramel latte',
11   price: 5500,
12   size: 'tall'
13 }
14
15 ✓ let latte3 = {
16   name: 'milktea latte',
17   price: 4500,
18   size: 'tall'
19 }
20
```



```
1  let name = 'CAFE'
2
3  ✓ class Latte {
4    name = '';
5    price = 0;
6    size = '';
7  ✓ constructor(name, price, size) {
8    this.name = name;
9    this.price = price;
10   this.size = size;
11 }
12 }
13
14 let latte1 = new Latte('Vanilla latte', 4500, 'tall');
15 let latte2 = new Latte('caramel latte', 5500, 'tall');
16 let latte3 = new Latte('milktea latte', 4500, 'tall');
17
```

# 03. Why OOP?

```
1  let name = 'CAFE'
2
3  class Latte{
4    name='';
5    price=0;
6    size=''
7    constructor(name,price,size){
8      this.name=name
9      this.price=price
10     this.size=size
11   }
12 }
13
14 class Ade{
15   name='';
16   price=0;
17   fruits='';
18 }
19 class Shake{
20   name='';
21   price=0;
22   flavor='';
23 }
```



```
1  let name = 'CAFE'
2
3  class Menu{
4    name='';
5    price=0;
6    constructor(name,price){
7      this.name=name
8      this.price=price
9    }
10 }
11
12 class Latte extends Menu{
13   size=''
14   constructor(name,price,size){
15     super(name,price)
16     this.size=size
17   }
18 }
19
20 class Ade extends Menu{
21   fruits='';
22 }
23 class Shake extends Menu{
24   flavor='';
25 }
26
```



## 04. Why OOP?

---

1. 코드 재사용이 용이하다
2. 유지보수가 쉽다
3. 캡슐화의 경우 유저의 직접적인 접근 방지

# 05. 퀴즈

#1. 전체코드는 다음과 같습니다. 빈칸을 완성하여 아래와 같은 실행결과가 나타나도록 하세요

```
PS C:\Users\suaajj\Desktop\4-winter\CS스터디> node main.js
Latte { name: 'milktea latte', price: 4500, size: 'tall' } 5500원 입니다 vanilla latte
```

```
let name = 'CAFE'

class Menu{
  name='';
  price=0;
  constructor(name,price){
    this.name=name
    this.price=price
  }
  getPrice(){
    return ;
  }
}

class Latte extends Menu{
  size=''
  constructor(name,price,size){
    super(name,price)
    this.size=size
  }
}
```

```
class Ade extends Menu{
  fruits='';
}

class Shake extends Menu{
  flavor=''
}

let latte1=new Latte('Vanilla latte',4500,'tall');
let latte2=new Latte('caramel latte',5500,'tall');
let latte3=new Latte('milktea latte',4500,'tall');

console.log();
```

# 05. 퀴즈

---

**#2. 객체지향 프로그래밍이 무엇인지 설명하세요**