OOP

객체지향프로그래밍

OOP - modifier

접근 제어자	
public	접근 제한이 없음
protected	같은 패키지 내에서 접근 가능, 다른 패키지의 자식 클래스에서 접근 가능
default	같은 패키지 내에서만 접근 가능
private	같은 클래스 내에서만 접근 가능

기타 제어자	
static	- class의, 공통의 - class정보이기 때문에, 인스턴스를 생성 하지 않고도 사용할 수 있다. - static변수는 모든 인스턴스에서 공통적으로 사용하는 class변수가 된다. - class가 메모리에 로드 될 때 생성된다.
final	최종의, 변경될 수 없는
abstract	추상의, 미완성의
synchronized	동기화 된

OOP

객체 지향 언어

객체 지향 : 현실 세계는 사물이나 개념처럼 독립되고 구분되는 각각의 객체로 이루어져 있으며,
현실에서 발생하는 모든 사건들은 객체간의 상호작용이다.

객체 지향 언어 : 프로그램에서 발생하는 모든 사건(기능)들을 객체간의 상호작용으로 나타내는 언어

객체 지향 프로그래밍 언어의 목표 : 추상화를 통한 프로그램의 유연성 확보
* 유연성 : 프로그램을 변경 하기 쉬운 정도

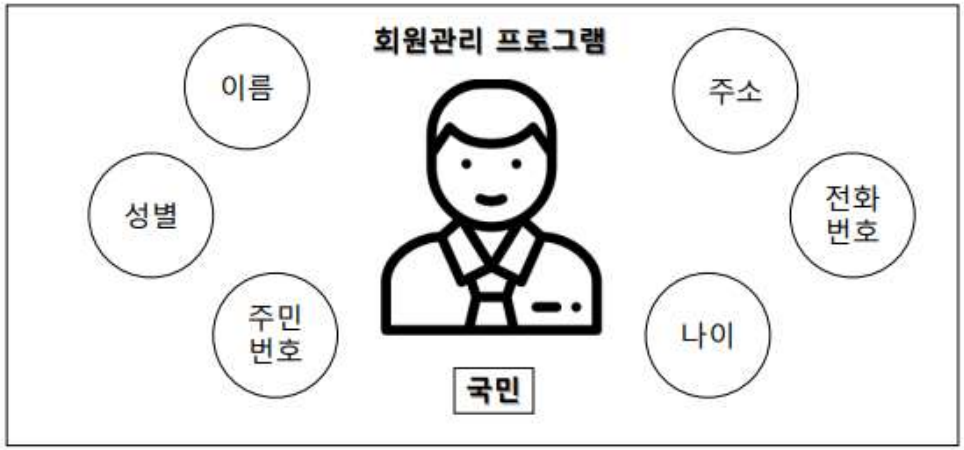
OOP

추상화

프로그램에 필요한 속성을 추출 하고, 이외의 속성은 감춰 현실 세계의 구체적인 대상을 추상화 하는 과정

프로그램을 통해 구현할 기능을 정의하고, 기능을 구현하는데 필요한 속성을 추출하여 정의한다.

ex) 국민 정보 관리 프로그램을 만들 때, 프로그램에서 요구되는 "국민" 의 정보를 추상화 한다면?



OOP

캡슐화

프로그램 내에서 같은 기능을 목적으로 작성된 코드들을 클래스로 묶어 관리하는 기법

클래스 내부 구현(속성, 기능)을 감춰 프로그램에 수정이 발생 했을 때 영향을 클래스 단위로 격리 시킨다

*** 캡슐화 방법**

1. 속성의 접근 제한자는 private을 기본으로 한다.
2. 외부에서 접근이 필요한 속성은 getter/setter 메서드를 제공한다.
3. 외부에서 기능을 호출 하기 위한 public 메서드를 제공하고, 구체적인 구현은 private 메서드에 구현해 숨긴다. (추상화)

OOP

상속

부모 클래스의 속성과 기능, 타입을 자식 클래스가 물려받는 것.

- private 속성은 상속 받지 않는다.
- 자식 멤버의 접근제한자는 부모 멤버의 접근제한자 보다 넓은 범위의 접근제한자만 가능하다.

자식 클래스는 물려받은 부모 클래스의 속성과 기능, 타입을 사용할 수 있다.

부모 클래스의 기능을 자식 클래스가 재정의(Overriding) 할 수 있다.

- 부모 클래스에 선언된 메서드 선언부를 정확하게 지켜야 한다.
- 접근제한자는 부모 클래스 보다 넓은 범위로만 Overriding 할 수 있다.

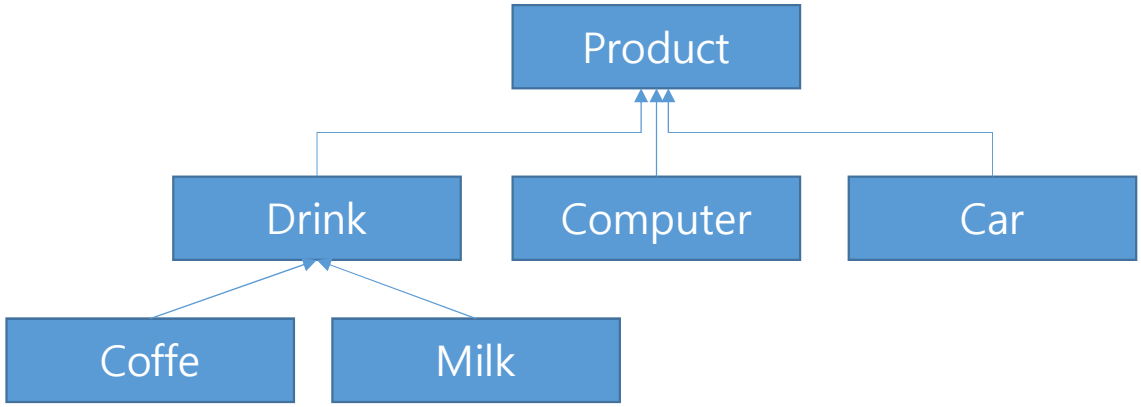
자식 클래스의 인스턴스는 부모 타입으로 다루어 질 수 있다. (타입의 공유)

OOP

다형성

자식 클래스의 인스턴스는 부모 타입으로 다루어 질 수 있다. (다형성)

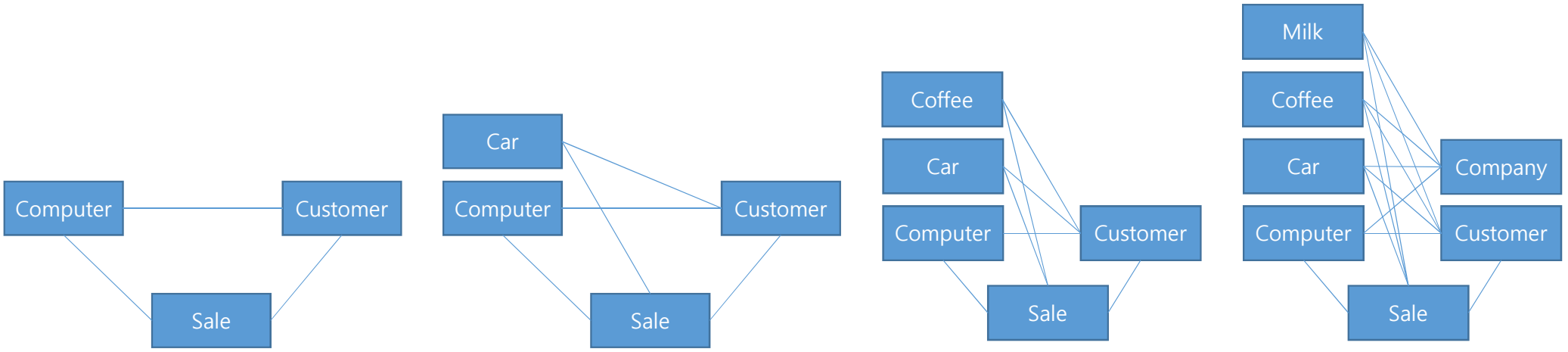
다형성은 구체적인 자식 인스턴스들을 추상적인 부모 타입으로 다루는 방식이다.



Caffee, Computer, Car, Milk 의 인스턴스를 Product라는 하나의 타입으로 다루는 것

OOP

다형성을 사용하지 않고 기능을 확장 할 때 클래스간 관계 복잡도

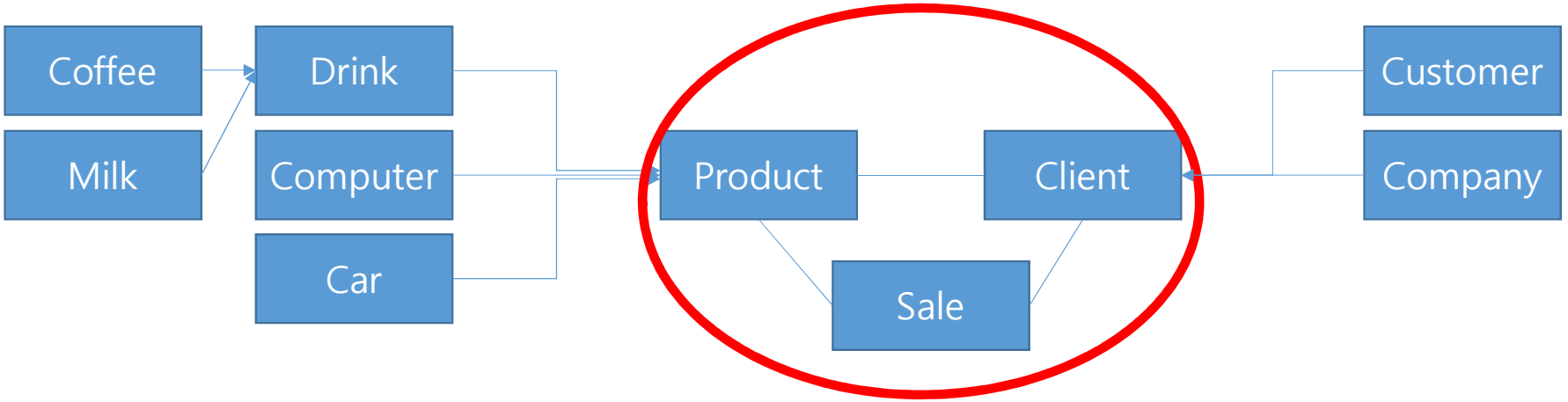


Computer **Car 추가** **Coffee 추가** **Milk 추가**
기업 고객 추가

OOP

다형성을 사용할 때 클래스간 관계 복잡도

구체적인 타입을 감춰 (추상화) 설계의 유연성을 확보



OOP

Overloading, Overriding

스펠링이 비슷할 뿐 둘은 별 상관이 없다.

Overloading : 매개변수의 타입이나 개수를 다르게 지정하여 하나의 이름을 가진 메서드를 여러개 만드는 것

Overriding : 부모클래스의 메서드를 재정의하는 것
접근제한자를 제외한 메서드의 선언부를 변경할 수 없다.
접근제한자는 부모보다 넓은 범위로만 변경이 가능하다.

OOP – Abstract Class

추상클래스(Abstract Class)

클래스 선언부에 abstract를 선언한 클래스

0개 이상의 추상메서드를 가지고 있다.

구현되지 않은 추상메서드가 있음으로 instance화 할 수 없다.

다른 Concrete Class를 만들 때 도움을 줄 목적으로 사용 된다.

추상클래스를 상속 받는 Concrete Class들이 추상클래스의 추상메서드를 구현한다.

OOP – interface

Interface

- 설계 관점에서 인터페이스는 클래스를 수식하기 위한 형용사적 의미를 지니게 된다.
  ex) comparable String, clonable Person

- 클래스를 구현하는데 필요한 규칙을 미리 정의하기 위한 용도로 사용한다.

- default메서드와 static메서드를 제외하면 추상메서드만 가질 수 있다.
  static메서드 : 인터페이스의 타입만으로 호출이 가능한 메서드
  default메서드 : 인터페이스 구현체의 인스턴스를 통해 호출해야 하는 메서드

- 인터페이스의 필드에 작성한 변수는 자동으로 public static final이 선언 된다.

- 인스턴스를 생성할 수 없다.

- 클래스는 인터페이스를 다중 구현할 수 있다.

- 인터페이스는 인터페이스 간 다중 상속이 허용 된다.

OOP – final

final class

클래스 선언부에 final을 선언한 클래스
다른 클래스가 상속 받는 것을 금지한다.

final method

메서드 선언부에 final을 선언한 메서드
자식 클래스에서 override하는 것을 금지한다.

final variables

변수를 선언할 때 final 예약어를 사용하면 초기화 이후 값을 재할당하는 것을 금지한다.

OOP – Inner Class

내부클래스

클래스 내부에 선언되는 클래스

클래스의 멤버와 동일한 접근성을 가진다. -> 외부 클래스의 private 변수에 접근 가능

내부 클래스	특징
instance class	외부 클래스의 필드 위치에 선언한다. 외부 클래스의 instance 멤버처럼 다루어 진다. 외부 클래스의 instance 멤버들과 관련된 작업에 사용할 목적으로 선언된다.
static class	외부 클래스의 필드 위치에 선언한다. 외부 클래스의 static 멤버처럼 다루어 진다. 외부 클래스의 static 멤버와 관련된 작업에 사용할 목적으로 선언된다.
local class	외부 클래스의 local(지역)에 선언한다 선언된 영역 내부에서만 사용 될 수 있다.

OOP – Anonymous Class

익명클래스

이름이 없는 일회용 클래스

클래스의 선언과 동시에 객체의 생성을 동시에 한다.

부모클래스나 인터페이스 타입의 일회성 객체가 필요할 때 사용한다.

new 부모클래스이름(){}

new 인터페이스이름(){}

OOP – Anonymous Class

Comparator 인터페이스의 익명클래스 예시

* Collection 공부하고 오시면 잘 이해 됩니다!

```
List<Student> studentList = new ArrayList<>();
...
studentList.sort(new Comparator<Student>() {
        @Override
        public int compare(Student o1, Student o2) {
                return o1.getAge() - o2.getAge();
        }
});
```