**객체 지향 프로그래밍** (Object-Oriented Programming)

객체의 관점에서 프로그래밍 하는 것

구성 요소들을 객체로 바라보고, 객체들을 유기적으로 연결해 프로그래밍 하는 것

목적: 코드의 재사용성과 중복제거

1. **추상화** (Abstraction) // 모델링의 기초

공통의 속성이나 기능을 묶어서 하나의 집합으로 다루는 것 = 클래스를 정의

데이터에 접근할 때 외부에서 직접 접근이 불가하고 함수를 통해서만 접근해야 함

1. **캡슐화** (Encapsulation) // 정보 은닉

객체의 특정한 목적을 위해 필요한 변수나 메소드를 하나로 묶는 것을 의미함

* 캡슐화의 목적: 코드를 재수정 없이 재사용하는 것

캡슐화를 통해 기능과 특성을 한 곳에 모아 분류하기 때문에 재활용이 원활해짐

* 정보 은닉

캡슐화를 하면 불필요한 정보를 감출 수 있어 정보 은닉 가능

불필요한 정보에 대해 외부 접근 불가능

1. **상속성** (Inheritance) // 재사용+새 기능 확장

기존 부모 클래스의 속성과 기능을 이어받아 재사용할 수 있고 변경할 경우 자식 클래스에서 기능만 수정하여 사용할 수 있음

* 상속의 목적: 코드의 중복을 없애기 위함

Java 에서는 클래스의 상속 관계에 혼란을 줄 수 있기 때문에 단일상속만 지원되지만 필요에 따라 인터페이스를 임시적인 다중상속에 대해 지원함

C++ 에서는 다중상속이 지원됨.

1. **다형성** (Polymorphism) // 사용 편의성 향상

상속을 통해 기능을 확장하거나 변경하는 것을 가능하게 함

└ 같은 형태이지만 다른 기능을 함

코드의 재사용, 코드의 길이가 감소되어 유지보수가 용이하도록 도와줌

└ 생산성 증대, 품질 향상, 비용 축소

* 오버라이딩 (Overriding)

부모 클래스(상위 클래스)에서 만들어진 메소드를 자식 클래스(서브 클래스)에서

재창조하여 사용하는 것

* 오버로딩 (Overloading)

메소드끼리 이름은 같지만 매개변수의 개수나 데이터 타입이 다르면 오버로딩이 적

용 됨. 메소드의 이름이 같아도 에러 안 남

\* 클래스 (Class)

비슷한 특성을 가진 객체들의 모델링 (그룹화한 틀, 캡슐화 한 것)

\* 객체 (Object)

속성을 가지며 특정한 작업을 수행할 수 있도록 설계된 것

\* 속성 (Attribute)

객체와 관련된 특성이나 변수 같은 것

\* 메소드 (Method) = 객체지향

객체에 대해 수행할 수 있는 작업/조작 (객체 외부와의 상호작용을 메소드를 통해 수행)

\* 연산 (Operation)

객체 자신을 바꾸거나, 외부적으로 바꿈의 효과가 나타나는 것