객체지향 프로그래밍 기본 개념 설명 (파이썬)

정보통신공학과

배종우 교수

2022년

객체지향 프로그래밍

- 클래스, 캡슐화, 상속, 다형성(polymorphism) 등의 새로운 개념
- “모든 사물은 객체다. 그것을 모델링 하는 것이 객체지향 프로그래밍이다” 라는 말은 틀린 것은 아니나 원하는 이해에 도달하기 어려움
- 객체지향 프로그래밍이 필요하게 된 배경을 이해하는 것이 중요함
- 이제까지 리스트 튜플 사전 등의 편리한 데이터 타입들과 흐름제어 방법 (if-else, while, for 구문들) 및 함수와 모듈 등을 배우며 대부분의 필요한 프로그램을 구현할 수 있음
- 그러면 된 것 아닌가? 무엇이 더 필요한가?

대형 프로그램 개발시의 어려움

- 거대 소프트웨어 업체에서 제품을 만들어 매년 수정 및 확장을 해나갈 때의 어려움 (윈도우 운영체제나 오피스 같은 프로그램 등)
- 기존 프로그램을 관리하는 것도 쉽지 않고
- 거기에 새로운 기능을 부가하고 확장해 나가는 것은 더욱 어렵다
- 기존 코드들을 다 이해하고 소화해 내기 어렵고
- 수많은 개발자들 간에 서로의 코드에 대한 이해 및 사용 등도 어렵다
- 심지어 내가 만든 코드도 시간이 흐르며 기능 확장이 필요하거나 할 때 기본 구조부터 갈아 엎어야 하는 경우도 있고 확장시 오류의 가능성이 점점 늘어 간다.
- 기존 high-level language 의 한계를 느끼게 됨 (C, Pascal, Fortran 등)

어떤 변화가 필요한가?

- 대형 프로그램의 안정된 관리가 필요함.
- 기존 데이터를 수정하는 것이 용이하고 기존 데이터를 필요에 따라 새로운 구조로 확장하는 것이 용이해야 좋겠음.
- 전체 코드나 기본 데이터 구조에 대한 깊은 이해가 없어도 필요한 기능의 추가 개발이 용이했으면 좋겠음 (데이터가 잘 보호되어야 하는 것을 의미하기도)
- 기존 언어에 struct 같은 구조체가 (C언어에) 있는데 이처럼 관련된 데이터들을 하나의 구조체로 만드는 것의 중요성을 깨닫게 됨
- 연관된 데이터들을 하나로 모으고 거기에 더해서 연관된 함수들도 하나의 구조 안에 모아서 그것을 새로운 구조로 정의함 → 이것이 class 라는 구조체임

클래스

- 클래스라는 구조에 변수와 함수를 포함시켜 정의함
- 멤버변수, 멤버함수 (혹은 메소드라고도 부름)
- 이렇게 하여 잘 관리해야 할 주요 데이터 들을 하나의 클래스에 묶어서 함부로 비전문가가 프로그래밍 중에 수정해 버리는 일이 없게 함 → 데이터의 캡슐화
- 접근제한자 (public, private) 사용으로 주요 데이터를 보호함
- 보호된 데이터의 외부 사용을 위해 메소드를 인터페이스로 제공함

상속

- 클래스라는 구조에 의한 데이터 캡슐화 정도는 기존 언어로도 비슷하게 흉내가 가능
- 기존 클래스 구조에 새로운 기능을 추가하고 싶을 때
- 매번 기존 클래스 구조를 변경해 나가면 그것을 사용하는 많은 연관 프로그램들이 혼란에 빠짐
- 인터페이스에 해당하는 메소드는 동일하게 유지하며 내부 데이터나 클래스 내부 구조 등을 변경할 수 있겠으며
- 더 용이한 방식으로 기존 클래스를 상속 받는 파생 클래스의 개념을 도입함.

상속

- 기존 클래스를 자세히 몰라도 필요한 새로운 나를 위한 클래스를 쉽게 정의해서 사용할 수 있음.
- 새로운 상속받은 클래스 정의에 기반한 코드의 확장이 용이함. 기존 코드는 안전하게 보존될 수 있음
- 거대 프로젝트 개발시 개발코드나 개발자 간의 독립성 및 비의존성을 확보할 수 있어서 빠르게 새로운 코드를 확장해 나갈 수 있음.

더 무엇이 필요한가?

- 클래스를 이용하고 상속을 지원하면 이제 충분히 객체지향 프로그램이 되었는가?
- 캡슐화를 통한 데이터 보호 및 상속에 의한 데이터 확장성은 확보가 되었다
- 더 유연한 프로그래밍 구조는?
- 내가 사용할 대상 데이터의 타입을 정확히 규정하지 않고도
- 혹은 다양한 데이터 유형에 대해서도 동작하는 프로그램을 작성할 수 있으면 좋겠음
- 좀 더 정확히 말하자면 하나의 함수 등의 구현으로 여러 데이터 타입들을 다 처리해 줄 수 있으면 좋겠음

다형성 (polymorphism)

- 이렇게 단순한 구현으로 여러 경우들을 커버할 수 있었으면 하는 필요성을 한 마디로 다형성에 대한 수요가 생겼다고 하겠음
- 함수중복: 동일한 함수명으로 다른 개수 또는 다른 데이터 타입의 매개변수를 갖는 함수를 구현함. (앞에서 배운 기본값을 갖는 매개변수의 함수정의)
- 연산자 재정의: 기존의 사칙연산 등에 사용되던 연산자 (+, -, *, / 등) 를 연산대상을 다양화 함.
- 새로운 데이터 타입들 (클래스의 파생으로 구현된 객체들) 간에 기존 연산자 사용
- 문자열들을 더하거나 리스트를 더하거나 하는 것이 직관적으로 가능해짐. (예: print(“파이썬“+”공부“), my_list = [1, 2, 3] + [4, 5, 6])

다형성 (polymorphism)

- 다이내믹 바인딩 혹은 업캐스팅
- 다형성의 핵심적인 방식
- 데이터 타입에 관계 없이 동작할 수 있는 코드
- 데이터 타입에 관계 없이 동작할 수 있는 함수라든가 프로그램의 어떤 부분.
- 프로그램 실행시에 데이터 타입을 결정해서 연결해 줌. 컴파일 타임에 정적으로 결정되지 않음. 이러한 것을 다이내믹 바인딩이라고 함 (dynamic binding). C++ 등에서 포인터를 사용할 경우 업캐스팅(up-casting) 방식으로 구현함.

제네릭 프로그래밍

- 더 진보된 다형성 구현방식
- Template 이라는 구조 사용
- 데이터에 무관하게 일반화된 프로그래밍이라고 하여 generic programming 이라고 부름
- Standard Template Library (STL) 등이 제공됨
- 데이터 타입에 무관한 여러 함수나 데이터 구조 제공됨
- 파이썬에서 배운 리스트, 튜플, 집합, 사전 등의 데이터 타입들이 기존 C++ 등의 STL 에 정의된 유사한 타입들을 파이썬은 프로그래밍 언어 자체의 기본 데이터형으로 제공하고 있다고도 볼 수 있음