객체 지향의 특징

1. 추상화(Abstraction)

: 자료 추상화는 불필요한 정보는 숨기고 중요한 정보만을 표현함으로써 프로그램을 간단히 만드는 것이다. 자료 추상화를 통해 정의된 자료형을 추상 자료형이라고 한다. 추상 자료형은 자료형의 자료 표현과 자료형의 연산을 캡슐화한 것으로 접근 제어를 통해서 자료형의 정보를 은닉할 수 있다. 객체 지향 프로그래밍에서 일반적으로 추상 자료형을 클래스, 추상 자료형의 인스턴스를 객체, 추상 자료형에서 정의된 연산을 메소드, 메소드의 호출을 메시지라고 한다.

상속화를 통해 구현해야 하는 공통의 기능을 가진 새로운 데이터 형을 만듦으로써, 필요한 기능을 빠짐없이 만들 수 있는 틀을 제공할 수 있으며 동일한 이름의 메서드를 가지게 되므로 후에 변경, 수정하기가 쉬워진다.

|  |
| --- |
| **package** com.samsung.tv;  **public** **interface** TV {  **public** **void** powerOn();  **public** **void** powerOff();  **public** **void** volumeUp();  **public** **void** volumeDown();  } |

위와 같이 TV가 가져야 하는 공통적 기능을 구현부 없이 선언해둔다.

1. 상속(Inheritance)

: 상위개념의 특징을 하위 개념이 물려받는 특징. 추상클래스나 인터페이스를 만들고 상속(extends)하거나 구현(implements)하는 클래스를 만들어 구체화한다. 상위 수준 클래스의 모든 특성을 하위 수준 클래스가 이어받음으로써, 이미 정의한 클래스를 재사용하고, 확장 할 수 있도록 지원하는 개념이다. 공통된 성질은 모두 상위 클래스에 정의하고 그 성질을 상속받은 하위 클래스에는 상위 클래스에 없는 독자적인 성질만 추가한다. 상속을 통해 확장성, 유지보수성, 재사용성, 이해용이성, 추상화, 대치기능과 같은 장점을 얻을 수 있다.

위의 예시를 이용해서 보면,

|  |
| --- |
| **package** com.samsung.tv;  **public** **class** SamsungTV **implements** TV{    **public** **void** powerOn(){  System.*out*.println("SamsungTV => 전원 켜기");  }    **public** **void** powerOff(){  System.*out*.println("SamsungTV => 전원 끄기");  }    **public** **void** volumeUp(){  System.*out*.println("SamsungTV => 소리 올리기");  }    **public** **void** volumeDown(){  System.*out*.println("SamsungTV => 소리 내리기");  }  } |

|  |
| --- |
| **package** com.samsung.tv;  **public** **class** LgTV **implements** TV{    **public** **void** powerOn(){  System.*out*.println("LgTV => 전원 켜기");  }    **public** **void** powerOff(){  System.*out*.println("LgTV => 전원 끄기");  }    **public** **void** volumeUp(){  System.*out*.println("LgTV => 소리 올리기");  }    **public** **void** volumeDown(){  System.*out*.println("LgTV => 소리 내리기");  }  } |

1. 다형성(Polymorphism)

: 동일한 외부 명령에 대해 각 객체가 서로 다른 방식으로 그 명령을 수행하는 것을 말한다. 오버라이딩이라는 개념이 가능한 것도 이 다형성 덕분인데, 오버라이딩이란, 상위클래스의 속성이나 오퍼레이션을 하위 클래스에서 내용을 다르게 재정의해서 사용하는 것을 말한다. 특히 메소드 오버라이딩은 같은 이름을 갖는 오퍼레이션이 호출 할 때 , 매개변수에 따라 그 동작을 달리하도록 구현 할 수 있도록 해 놓은 것이다.

추상클래스를 상속하거나, 인터페이스를 구현하는 클래스들의 경우, 각 클래스마다 동일한 이름의 메소드 가지고 있더라도 서로 다른 내용으로 구현할 수 있다.

1. 캡슐화(Encapsulation)

: 객체지향에서 캡슐화라는 개념은 클래스 내부에 여러 속성과 여러 오퍼레이션을 함께 묶음을 의미한다.

그리고 캡슐화는 클래스 내부의 속성이나 오퍼레이션을 외부에 노출하지 않고 보호하는 것을 의미한다.

이렇게 캡슐화는 묶는 것과 보호하는 것을 생각할 수 있다. 좀 더 상세하게 생각해 보면, 여러 속성과 여러 오퍼레이션을 함께 묶어 클래스로 취급하는 것과 클래스 내부를 외부에서 접근하지 못하도록 보호하는 역할을 한다.

클래스 내부에 여러 함수를 내포할 수 있기 때문에 프로그램 소스 코드를 바라보는 단위가 커졌으며, 그로 인해 프로그램 관리가 좀 더 수월해진다는 장점이 있다. 또한 내부를 숨김으로써 내부를 좀더 자유롭게 변경할 수 있게 되었다. 이전의 함수 중심적인 구조적 프로그래밍 언어에서는 프로그램 내부에서 데이터가 어디서 어떻게 변경되는지 파악하기 어려웠고, 그로 인해 유지 보수가 힘들었기 때문에 자료를 중심으로 함수가 종속되는 구조가 되기도 하였다. 객체 지향에서는 클래스 내부의 데이터를 외부에서 참조하지 못하도록 차단하여 이러한 폐단을 없앨 수 있다. 이렇게 내부의 데이터나 함수를 외부에서 참조하지 못하도록 차단하는 개념을 정보 은닉Information Hiding)이라고 한다.