1. **객체지향언어**

배경: 실제 세계와 유사한 가상 세계를 컴퓨터 속에 구현하고자 함.

개념: ‘실제 세계는 사물(객체)로 이루어져 있으며, 발생하는 모든 사건들은 사물간의 상호작용이다.’ 실제 사물의 속성과 기능을 분석한 다음, 변수와 함수로 정의함.

**객체지향언어의 특징**

- 코드의 재사용성이 높음.

- 코드의 관리가 용이함 (loose coupling)

- 프로그래밍의 신뢰성이 높아짐. (제어자와 메서드를 이용해서 데이터를 보호함)

**Encapsulation**

정의: 캡슐화는 한 객체 내의 데이터를 다른 객체의 접근으로부터 보호하고, 객체 스스로가 데이터 접근을 통제할 수 있게 한다.

사용 방법: 변수를 private으로 선언하고, public 메서드(getter, setter)를 사용해 변수를 변경한다.

1. **클래스와 객체**
2. 클래스: ‘객체를 정의해 놓은 것’ 또는 ‘객체의 설계도 또는 틀’. 클래스는 객체를 생성하는 데에 사용되며, 객체는 클래스에 정의된 대로 생성됨.
3. 객체: ‘실제로 존재하는 것’. 클래스에 정의된 내용대로 메모리에 생성된 것.
4. 인스턴스: 클래스로부터 객체를 만드는 과정을 클래스의 인스턴스화(instantiate)라고 하며, 어떤 클래스로부터 만들어진 객체를 그 클래스의 인스턴스라고 함.
5. **변수**
6. 인스턴스변수

클래스 영역에 선언됨. 클래스의 인스턴스를 생성할 때 만들어짐. 인스턴스는 독립적인 저장공간을 가지므로 서로 다른 값을 가질 수 있으며, 인스턴스마다 고유한 상태를 유지해야하는 속성의 경우 인스턴스 변수로 선언함.

1. 클래스변수(static변수)

인스턴스 변수 앞에 static을 붙여 선언함. 클래스변수는 모든 인스턴스가 공통된 저장공간을 공유하게 됨. 한 클래스의 모든 인스턴스들이 공통적인 값을 유지해야하는 속성의 경우 클래스변수로 선언함. 인스턴스를 생성하지 않고 바로 사용할 수 있음. 클래스가 로딩될 때 생성되어 프로그램이 종료될 때까지 유지됨.

1. 지역변수

메서드 내에 선언되어 메서드 내에서만 사용가능함. 사용하기 전에 반드시 초기화해야함.

1. **메서드**

정의: 어떤 작업을 수행하기 위한 명령문의 집합. 반복적으로 사용되는 코드를 줄이기 위해 사용함.

1. 클래스메서드(static메서드)

객체를 생성하지 않고 호출할 수 있는 메서드. 모든 인스턴스가 공통적으로 사용해야 하는 메서드는 static메서드로 선언함. Static메서드는 인스턴스변수를 사용할 수 없음.

1. 인스턴스메서드

인스턴스를 생성해야만 호출할 수 있는 메서드. 인스턴스멤버가 존재하는 시점에 클래스멤버는 항상 존재하므로, 클래스멤버를 사용할 수 있음.

1. **Method Overloading**

정의: 한 클래스 내에 같은 이름의 메서드를 여러 개 정의하는 것. 같은 일을 하지만 매개변수를 달리 해야 하는 경우에, 이름은 같고 매개변수를 다르게 하여 오버로딩을 구현함.

**오버로딩의 조건**

1. 메서드 이름이 같아야 한다.
2. 매개변수의 개수 또는 타입이 달라야 한다.
3. 매개변수는 같고 리턴타입이 다른 경우는 오버로딩이 되지 않는다. (리턴타입은 오버로딩에 아무 영항을 주지 못한다.)
4. **생성자(Constructor)**
5. 정의

인스턴스 초기화 메서드. 메모리 내에 객체가 생성될 때 호출되어 객체의 멤버변수들을 초기화함. 생성자명은 클래스명과 같아야 하고, return type을 정의하지 않는다.

연산자 new가 인스턴스를 생성하는 것이며, 생성자는 인스턴스변수 초기화에 사용되는 메서드이다.

1. 기본 생성자(Default Constructor)

클래스 안에 따로 생성자를 정의하지 않으면 JVM이 자동으로 기본 생성자를 제공한다. 기본 생성자는 매개변수가 없으며, 클래스 내의 멤버변수를 data type별 디폴트 값으로 초기화한다.

1. 생성자에서 다른 생성자 호출하기: this

인스턴스 자신을 가리키는 참조변수. (static메서드에서 사용 불가)

객체의 주소는 생성 전까지는 모르기 때문에 객체 생성 후 자신의 주소로 대치됨. 모든 인스턴스메서드에 지역변수로 숨겨진 채 존재함.

this()는 같은 클래스 안에서 오버로딩 된 다른 생성자를 호출할 때 사용함. 한 생성자에서 다른 생성자를 호출할 때에는 반드시 첫 줄에서만 호출 가능함.

1. **상속**

정의: 기존 클래스를 재사용하여 새로운 클래스를 작성하는 것.

필요성: 코드의 중복을 제거하여 보다 적은 양의 코드로 새로운 클래스를 작성할 수 있고, 코드의 재사용성을 높임. 코드를 공통으로 관리할 수 있기 때문에 프로그램의 생산성과 유지보수에 기여함.

사용법: 자식클래스의 이름 뒤에 부모클래스의 이름을 키워드 ‘extends’와 함께 쓰면 됨.

자손 클래스는 조상 클래스의 모든 멤버를 상속받으므로, 자손 클래스에 코드를 추가하면 조상 클래스보다 항상 같거나 많은 멤버를 가짐. 따라서 상속은 조상클래스를 확장(extend)한다는 의미로 해석할 수도 있으며, 그런 의미에서 키워드 ‘extends’를 사용함.

1. **Method Overriding**
2. 정의

부모클래스로부터 상속받은 메서드를 자식클래스 자신에 맞게 변경하는 것.

**오버라이딩의 조건**

1. 메서드 이름이 같아야 한다.
2. 매개변수가 같아야 한다.
3. 리턴타입이 같아야 한다.
4. 접근 제어자(Access modifier)는 부모클래스의 메서드보다 좁은 범위로 변경할 수 없다. 예를 들어, public에서 private으로 변경할 수 없다.
5. 부모클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.
6. 인스턴스메서드를 static메서드로 또는 그 반대로 변경할 수 없다.
7. super

자식클래스에서 부모클래스로부터 상속받은 멤버를 참조하는 데에 사용하는 참조변수.

super 역시 static메서드는 사용할 수 없고, 인스턴스메서드에서만 사용 가능함.

super()는 부모클래스의 생성자를 호출할 때 사용함. Object클래스를 제외한 코든 클래스의 생성자는 첫 줄에 반드시 자신의 다른 생성자 또는 조상의 생성자를 호출해야함. 그렇지 않으면 컴파일러는 생성자의 첫 줄에 super()를 자동으로 호출함.

1. **다형성**
2. 정의

한 타입의 참조변수로 여러 타입의 객체를 참조할 수 있도록 함. 부모클래스 타입의 참조변수로 자식클래스의 인스턴스를 참조할 수 있도록 함. 반대로 자식타입의 참조변수로 부모타입의 인스턴스를 참조할 수는 없음.

아래와 같이 부모클래스 타입의 참조변수로 자식클래스 타입의 객체를 생성함.

Parent p = new Child();

1. 참조변수의 형변환

자식타입 → 부모타입 (up-casting) : 형변환 생략가능

부모타입 → 자식타입 (down-casting) : 형변환 생략불가

1. **추상클래스**
2. 정의

추상메서드를 가지고 있는 클래스. 다른 변수와 메서드 포함 가능. 추상클래스의 인스턴스는 생성할 수 없으며, 상속을 통해 자식클래스에 의해서만 완성될 수 있음.

1. 추상메서드

선언부만 작성하고 구현부는 작성하지 않은 채로 남겨 둔 메서드. 메서드의 내용이 상속받는 클래스에 따라 달라질 수 있기 때문에 부모 클래스에서는 선언부만 작성하고 주석을 덧붙여 어떤 기능을 수행할 목적으로 작성되었는지 알려주고, 실제 내용은 상속받는 클래스에서 구현하도록 비워둠. 굳이 추상메서드를 선언하는 이유는 자식 클래스에서 추상메서드를 반드시 구현하도록 강요하기 위함.

1. **인터페이스**
2. 정의

사용자가 클래스를 사용하기에 편리하도록 클래스가 구현하는 일종의 껍데기. 오직 추상메서드와 상수만을 멤버로 가짐. 인터페이스 자체의 인스턴스는 생성할 수 없음.

* 모든 멤버변수는 public static final이어야 하며, 이를 생략할 수 있다.
* 모든 메서드는 public abstract이어야 하며, 이를 생략할 수 있다.

(인터페이스에 정의된 모든 멤버에 적용되는 사항이기 때문에 제어자를 생략할 수 있는 것이며, 생략된 제어자는 컴파일러가 자동으로 추가해줌.)

1. 장점
2. 개발시간을 단축시킬 수 있다.

메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되고, 다른 쪽에서는 구현부를 작성하도록 하여 양쪽에서 동시에 개발할 수 있다.

1. 표준화가 가능하다.

프로젝트의 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하게 함으로써 일관되고 정형화된 프로그램 개발이 가능하다.

1. 서로 관계없는 클래스들에게 관계를 맺어줄 수 있다.

상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 클래스들에게 하나의 인터페이스를 구현하게 함으로써 관계를 맺어줄 수 있다.

1. 독립적인 프로그래밍이 가능하다.

클래스와 클래스 간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 다른 클래스에 영향을 미치지 않게 된다.

1. **Exception**
2. 정의

에러: 프로그램 코드에 의해서 수습될 수 없는 심각한 오류 (OutofMemoryError, StackOverflowError 등)

예외: 프로그램 코드에 의해서 수습될 수 있는 오류

예외처리: 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것. 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하기 위함.

1. try-catch

try 블록 다음에는 여러 종류의 예외를 처리할 수 있도록 하나 이상의 catch 블록이 올 수 있으며, 이 중 발생한 예외의 종류와 일치하는 단 한 개의 catch 블록만 수행된다. 발생한 예외의 종류와 일치하는 catch 블록이 없으면 예외처리는 되지 않는다.

1. finally

예외의 발생여부에 상관없이 반드시 실행되는 블록.

try 또는 catch 블록에서 return해도 finally 블록은 반드시 실행됨.

1. 메서드에 예외 선언하기

메서드의 선언부에 throws룰 사용해서 메서드 내에서 발생할 수 있는 예외를 적어주는 것. 이는 예외를 처리하는 것이 아니라, 자신을 호출한 메서드에게 예외를 전달해 예외처리를 떠맡기는 것임. 결국 어느 한 곳에서는 반드시 try-catch문으로 예외처리를 해야 하며, 호출스택 제일 마지막에 있는 main메서드에서도 예외처리가 되지 않으면 main메서드가 종료되어 프로그램 전체가 종료됨.