**1) 추상화(Abstraciton)**

- 공통의 속성이나 기능을 묶어 이름을 붙이는 것

* 추상화 작업을 통해 실 세계 상황을 간결하고 명확하게 모델링하면, 그것을 프로그램으로 구현한다.
* 객체지향에서는 클래스를 이용하여 실 세계에 대응하는 추상 모델을 만든다.
* 추상 클래스를 상속하면(extends) 하위 클래스에서는 추상 클래스에 선언되어 있는 추상 메소드를 구현해야 한다. 어떤 면에서 추상 클래스는 인터페이스와 유사하다. 그러나 인터페이스와는 다르게, 추상 클래스는 추상 메소드가 아닌 메소드를 포함할 수도 있고 상수 이외의 데이터 선언도 포함할 수 있다.
* 추상 클래스의 자식 클래스는 부모로부터 상속받은 모든 추상 메소드들을 정의해야 한다. 그렇지 않으면 자식 역시 추상 클래스가 된다.
* 추상 메소드를 final 이나 static 으로 정의하는 것은 모순이라는 점을 주의해야 한다. final 메소드는 자식 클래스에서 재정의 될 수 없다. static 메소드는 클래스 객체 선언 없이 클래스 이름으로 호출하는 이 방법은 그 메소드에 대한 구현이 이미 있어야만 가능하다.

**2) 캡슐화(Encapsulation)**

- 데이터 구조와 데이터를 다루는 방법들을 결합 시켜 묶는 것. 다시 한번 말하자면 변수와 함수를 하나로 묶는것을 말한다.

- ex) public String test() {

          string aa = "aaa";

      }

- 하지만 무작정 한 데 묶으면 되는 것이 아니라  객체가 맡은 역할을 수행하기 위한 하나의 목적을 한 데 묶는다고 생각해야 한다. 이것을 **은닉화**라고 한다.

- 또한 데이터를 절대로 외부에서 직접 접근을 하면 안되고 오로지 함수를 통해서만 접근해야 하는데 이를 가능하게 해주는 것이 바로 캡슐화이다.

- 따라서 캡슐화에 성공하면 당연히 은닉화도 자연스럽게 효력이 나타난다.

**3) 상속성, 재사용(Inheritance)**

- 상위 개념의 특징을 하위 개념이 물려받는 것

- 부모 클래스를 extends로 상속받아 자식 클래스에서 내 것처럼 사용할 수 있는 것이다.  시스템을 설계하다 보면, 공통된 부분으로 인하여 중복되는 것들이 계속적으로 나올 수 있다. 이럴 때 상속을 이용하여, 비슷한 객체끼리 하나의 묶음으로 묶어주고, 공통된 부분을 뽑아서 상위 개념의 부모 클래스를 생성한다. 그리고 이 객체들이 부모 객체를 상속하면 중복이 제거되며, 공통된 방식으로 설계할 수 있다.

**- 상속과 관련된 접근 제한자 protected** : 객체를 캡슐화하기 위해서 변수들의 modifier를 private로 설정하면, 이 객체를 상속받은 자식 클래스에서는 접근을 할 수가 없다. 외부에서의 변수에 접근은 막고 싶고, 자식 클래스에는 접근을 허용하고 싶을 때는 protected modifier를 사용한다. protected modifier는 외부에는 접근이 제한되고, 같은 패키지와 하위 클래스에만 접근을 허용한다.  
**- super 참조** : 예약어 super 는 그 부모 클래스를 참조하는데 사용한다. 생성자인 super() 를 호출하면, 부모 클래스의 생성자를 호출하게 된다. 상위 클래스의 생성자를 호출하는 super() 는 반드시 해당 클래스의 생성자 첫 번째 라인에 위치해야 한다. 이것은 상위 클래스의 생성자가 항상 하위 클래스의 생성자보다 먼저 수행되어야 하기 때문이다.

**- 단일 상속** : 자바에서는 다중 상속을 지원하지 않는다. 따라서 extends 를 통해 하나의 부모만을 가질 수 있는 단일 상속만 사용할 수 있다.

**- 상속의 제약** : 상수 final 을 선언하면, 다른 클래스에서 해당 클래스를 상속할 수 없다. final을 메소드에서 사용할 경우 하위 클래스에서 오버라이딩이 불가능하다.  final 은 최종 클래스에서 사용 가능하며, 더이상 확장할 수 없다.

**- 오버라이딩(Overriding)** : 상위 클래스의 메소드를 하위 클래스가 재정의해서 사용하는 기술이다. 하위 클래스에서 상위 클래스의 메소드 이름, 매개변수 유형을 동일하게 정의하면 하위 클래스에서 해당 메소드를 재정의하게 된다.

**4) 다형성(Polymorphism)**

- '여러가지 형태를 가질 수 있는 능력'을 의미하며, 자바에서는 한 타입의 참조변수로 여러 타입의 객체를 참조할 수 있도록 함을 의미한다.

|  |
| --- |
| **조상 타입의 참조 변수로 자손 타입의 인스턴스를 참조할 수 있다. 반대는 성립하지 않는다.**  class car { }  // 부모 클래스 class firecar extends car {}  // 자손 클래스  있다고 하면, 다음과 같은 참조가 가능하다.  car newcar = new firecar();   하지만 다음과 같은 참조는 불가능하다.  firecar newcar = new car(); |
| **같은 타입의 인스턴스이지만 참조변수의 타입에 따라 사용할 수 있는 멤버의 개수가 달라진다.** class car {                   public void go(){ // 앞으로 가는 메소드 작성}                  public void stop(){// 멈추는 메소드 작성}                 }  // 부모 클래스 class firecar extends car {                  public void putoff(){ // 불을 끄는 메소드 작성}                }  // 자손 클래스  car newcar = new firecar(); firecar newcar2 = new firecar();  **newcar.putoff() // 컴파일 에러** newcar2.putoff() // 컴파일 가능  부모의 참조변수로 자손클래스를 참조한 경우, 부모에서 정의된 메소드 멤버변수만을 참조할 수 있다. |
| **형변환시 자손 타입은 조상타입으로 형변환할때 생략이 가능하다. 그러나, 조상타입은 자손타입으로 형변환할 때는 생략이 불가능하다.**  class car {                   public void go(){ // 앞으로 가는 메소드 작성}                  public void stop(){// 멈추는 메소드 작성}                 }  // 부모 클래스 class firecar extends car {                  public void putoff(){ // 불을 끄는 메소드 작성}                }  // 자손 클래스  ..... 생략  car newcar = new firecar(); firecar newcar2 = new firecar();   car repaircar = newcar2; // 컴파일가능 firecar repaircar2 = (firecar)newcar // 컴파일 가능 firecar reparicar3 = newcar // 컴파일 에러   자손클래스의 참조변수가 더 많은 영역에서 변수들을 참조 할 수 있기 때문에, 참조변수 형을 변환할 때는 위의 규칙을 따르도록 되어있다. |
| **멤버 변수가 조상클래스와 자손클래스에 중복으로 정의되어 있는 경우, 참조변수의 타입에 따라 사용될 수 있는 변수가 달라진다.**  class car {                   public void go(){ // 앞으로 가는 메소드 작성}                  public void stop(){// 멈추는 메소드 작성}                   int door = 4;               }  // 부모 클래스  class firecar extends car {                  public void putoff(){ // 불을 끄는 메소드 작성}                  int door = 2;                }  // 자손 클래스   .... 생략  firecar newcar =  firecar(); System.out.println(newcar.door); // 2 출력  car newcar2 = car(); System.out.println(newcar2.dorr); // 4출력 |