01.16 月 _ 상속

추상화

> 특정 class 가 있으면 **그 class의 대표할 만한 속성들을 뽑아내서** 그것들을 이용해 설명하는 것

왜 추상화 작업을 해야 하나 ❓

➡️ → **상속** 때문이다

우리가 만들어야 할 하나의 System이 있다. 그리고 그 System을 구성하는 객체들이 있다. 그 객체들을 쭉 펼쳐 보니 뭔가 공통적인 특성들이 보이기 시작한다.
공통적인 특성들을 보기 위해서 각 class 마다 추상화 작업을 해야한다.

왜 상속을 하나 ❓

➡️ 1. 모든 객체들마다 공통적인 속성을 가지는 놈들을 전부 부모 객체로 올려서 안에 속성들의 관리를 효율적으로 할 수 있게 해주고

2. 부모의 형으로 다형성을 이용해서 자식들의 객체를 부모의 형으로 관리할 수 있기 때문!

상속을 하려면 공통적인 속성들을 일단 뽑아 내야 한다 👉 이 작업이 결국 **추상화** 작업이다.

class의 이름을 만드는 것 역시 **추상화** 작업이다 . ex) 농구 게임 (인간 , 선수….)

추상화 작업을 끝내 놓으면 각 class 마다 대표하는 속성들이 나타나기 시작한다.
이를 가지고 상속 관계를 이끌어 내야 한다.

class를 대표할만한 속성 자체를 뽑아내는 것이 **추상화**
→ 그 속성들을 위쪽의 부모로 올려 만들어내는 것 또한 **추상화**에 들어간다

> 🌟 추상 Method , 추상 class 없어도 프로그램은 돌아가지만
> **강제성（強制性）**을 적용 시키기 위해 사용한다
>
> 추상 Method , 추상 class 라는 것을 프로그래머에게 알려주기 위해
> **abstract** 붙인다

> ⭐ ⇒ 다형성과 추상화에는 반드시 **상속**이라는 개념이 들어가 있어야 한다!
>
> 1. 상속
> 2. 다형성 (가시성, 동적 바인딩) - 부모의 형으로 자식의 객체를 가르키는 것
> 3. 추상 (추상 클래스 , 추상 메서드) - 전체 class에서 각 class를 대표하는 공통 속성을 뽑음

타입 저글링을 사용하면 `다형성` , `Overroading` 이 적용되지 않는다!

> ⭐ ⭐⭐⭐⭐⭐
> ⭐⭐⭐⭐⭐
>
> `파이썬의 OOP`를 `자바OOP`와 비교해 가면서 공부해 본다

상속 (Inheritance)

⭐ **class 초기화 블록**은 class가 **사용될 때** 딱 1번만 호출된다. (상속 관계가 다 적용 된다)
따라 올라가면서 부모의 class 속성을 다 가지고 올라온다

⭐ 자식만 만들어 지는 경우는 없다❗
부모부터 다 완성되고 나서 자식이 만들어 진다❗

```
package Test;

//  상속

class A {
  static { System.out.println("S Initialization of A");} // class 초기화 블록
  { System.out.println("Initialization of A");}          // 인스턴스 초기화 블록
  A() { System.out.println("Constructor of A"); }        // 생성자
}

class B extends A {
  static int x = 3;
  static { System.out.println("S Initialization of B");}
  { System.out.println("Initialization of B");}
  B() { System.out.println("Constructor of B"); }
}

class C extends B {
  static int y = 3;

  static { System.out.println("S Initialization of C");}
  { System.out.println("Initialization of C");}
  C() { System.out.println("Constructor of C"); }
}

public class MyFoo {

  public static void main(String[] args) {
    System.out.println(C.y);
    System.out.println("====================");

    //1. A 의 인스턴스 초기화 블록 , 생성자 호출
    //2. B 의 인스턴스 초기화 블록 , 생성자 호출
    //3. C 의 인스턴스 초기화 블록 , 생성자 호출

    new C();
    //new C();
```

```
  }
}

# - *출력 내용 -*

S Initialization of A
S Initialization of B
S Initialization of C
3

Initialization of A
Constructor of A
Initialization of B
Constructor of B
Initialization of C
Constructor of C
```

```
package Test;

//  상속
//

class A {
  static { System.out.println("S Initialization of A");}
  { System.out.println("Initialization of A");}
  A() { System.out.println("Constructor of A"); }
}

class B extends A {
  static int x = 3;
  static { System.out.println("S Initialization of B");}
  { System.out.println("Initialization of B");}
  B() { System.out.println("Constructor of B"); }
}

class C extends B {
  static int y = 3;

  static { System.out.println("S Initialization of C");}
  { System.out.println("Initialization of C");}
  C() { System.out.println("Constructor of C"); }
}

public class MyFoo {

  public static void main(String[] args) {
    //System.out.println(C.y);
    //System.out.println("====================");

    new C();
    new C();

  }
```

```
}
```

생성자

```
package test;

// 상속
class A {
  A() {          // A의 부모는 Object
    //super( ) ; 생략 되어 있다         super( ) = 참조변수(부모 객체의 주소 값)
    System.out.println("Constructor of A");
    }
}

class B extends A {
  B() {
    //super( ) ; 생략 되어 있다
    System.out.println("Constructor of B");
    }
}

class C extends B {
  C() {
    //super( ) ; 생략 되어 있다
    System.out.println("Constructor of C");
    }
}

public class TEST7 {

  public static void main(String[] args) {

    new C();   //  ( )  <- 이게 생성자를 호출 하는 것
               //   이것은 사용자가 선택해 호출 하는 것
  //
  }
}

 *출력 내용-*

Constructor of A
Constructor of B
Constructor of C
```

- 전처리(기) 과정에서 생성자에 `super( ) ;`를 다 자동으로 붙인다
- 메모리 상에는 A B C 가 만들어지고 생성자를 호출한다, 호출하는 것은 제일 마지막 자식의 실제 찍고자하는 class의 생성자를 호출한다.

⭐ 자식이 존재하려면 부모가 존재해야 한다. → 초기화 작업도 부모부터 일어난다.
메소드는 실행이 다 되면 다시 원래대로 돌아와야 한다.

호출은 자식에서 했지만 실행은 부모까지 갔다가 부모부터 다 실행이 되고 자식 순으로 실행이된다.

Java에서 생성된 모든 class의 최상위 부모는 Object !

⭐ Java 에서는 모든 class 를 Object 로 관리한다.
Java에서 clss를 만드는데 상속 관계가 아니더라도
무조건 그 class는 자동으로 `exteds Object` 가 자동으로 붙는다❗ Object 위에는 없다
⇒ Java 에서 상속을 받지 않은 모든 class는 Object 를 부모로 가진다.

Java 에서는 모든 class를 하나로 관리하고 싶어한다. 그 class 마다 각각 공통적인 기능을 추상화 작업 시킨다. 그럼 Java에서 만들어진 하나의 자료형으로 관리하고 싶다. 그래서 Object 를 하나 만들고 추상화 해서 나오는 개념들을 Java의 객체라면 무조건 가져야 할 속성들을 Object 에 집어넣는다 (그 중에 하나가 toString)

`default constructor` 가 없는데 불러오니 에러가 뜬다!!

자바에서 생성자를 만드는데 , class 내에 생성자가 하나도 없으면 `default constructor` 를 자동으로 주입시키다

생성자가 하나라도 있으면 `default constructor` 는 만들어 내지 않는다.

생성자는 Overroading 이 가능하다

`default constructor` 를 만들어 놓지 않고 나중에 `default constructor` 를 호출하려 하면 에러!

```
package test;

// 상속
class A {
  A() {
    super();
    System.out.println("Constructor of A");
    }
}

class B extends A {
  B(int a) {      //  <- 기본 생성자가 있기 때문에 default 생성자를 자동으로 안 만든다!
    super();
    System.out.println("Constructor of B");
    }
}

class C extends B {
  C() {
    super();
    System.out.println("Constructor of C");
    }
}

public class TEST7 {
  public static void main(String[] args) {

    new C();
  }
}
```

▶️ 이렇게 사용자가 직접 super( )를 건드려 줘야 한다

```
package test;

// 상속
class A {
  A() {
    super();
    System.out.println("Constructor of A");
    }
}

class B extends A {
  B(int a) {
    super();
    System.out.println("Constructor of B");
    }
}
```

```
class C extends B {
  C() {
    super(2);     //  <- 이렇게 B 생성장 맞는 값을 super에 넣어줘야 한다
    System.out.println("Constructor of C");
    }
}

public class TEST7 {
  public static void main(String[] args) {

    new C();
  }
}
```

⭐ 부모 class도 생성자가 `overroading` 된다!
상속 받으면서 내 필요에 따라서 여러 개 중에 한 개를 선택할 수 있다.
→ 부모의 어떤 생성자를 호출하겠다 라고 super( ) 를 직접 건드려야 한다

생성자의 종류를 확인하고 맞춰서 불러와야 한다!!

⭐ 상속을 받을 때 위의 class를 제대로 확인 (생성자) 하고 상속 받아야 한다!

```
package test;

// 상속
class A {
  A() {
    super();
    System.out.println("Constructor of A");
    }
}

class B extends A {
  B(int a) {
    super();
    System.out.println("Constructor of B");
    }
  B(int a , int b){}
}

class C extends B {
  C() {
    super(2);  // 생성자의 종류를 확인하고 맞춰서 불러와야한다!!
    System.out.println("Constructor of C");
```

```
    }
}

public class TEST7 {

  public static void main(String[] args) {

    // 여기 2줄 주석 처리해서  실행결과 한번 보기
//    System.out.println(C.y);
//    System.out.println("====================");

//    new C();
    new C();
  }
}
```

⭐ 객체가 다 만들어지고 초기화 작업이 일어난다.
⇒ 메모리에 다 올라가고 난 다음에 초기화 작업!

Modifier

> class와 class의 구성 요소를 꾸며준다

1. Access Modifier (접근 제어자)
   : 내부에서 접근하거나 외부에서 접근할 수 있게
   제어할 수 있는 역할 (접근하는 정도를 지정해줄 수 있음)
2. 그 외 (다른 언어에 있는 것도 있고 , 없는 것도 있다)

❓ Access Modifier (접근 제어자) 는 언제 사용되나?

1. 참조 변수 관점에서 사용된다 (외부 접근)
2. 상속 관계에서 사용된다

제어자가 필요하다!

```
package test;

// 상속
class Car {
  int velocity = 0;
  void IncAcel() {++velocity;}
  void DecAcel() {--velocity;}

  int getVelocity () {return velocity;}
}


public class TEST7 {
  public static void main(String[] args) {

    Car obj = new Car();
    obj.IncAcel();
    obj.IncAcel();
    System.out.println(obj.getVelocity());

    obj.DecAcel();
    System.out.println(obj.getVelocity());

    obj.velocity = 100;   // 갑자기 지멋대로 100을 치노?  접근 제어자 필요!!!
    System.out.println(obj.getVelocity());
  }
}
```

	친절 (다 됨)	←	←	엄격(다 안됨)
	public	**protected**	**default**	**private**
외부 참조	🔵	🔺 같은 패키지 안에서만	🔺 같은 패키지 안에서만	🚫 본인 class 안에서만
상속	🔵	🔺 **다른 패키지에서 상속만 가능**	🔺 같은 패키지 안에서만	🚫 본인 class 안에서만

상속을 하는데 , 자식에게 빛을 물려주고 싶지 않다…

어떤 멤버를 자식에게 물려주고 싶지 않을 때 접근 제어자를 사용한다!