# Extend

▼ 상속의 개념

부모가 자식에게 물려주는 행위.

객체에 대한 클래스를 여러 개 만드는 과정에서 각 클래스들이 서로 공통된 (중복된) 필드변수나 메소드를 가지고 있는 경우,

이것들을 하나로 묶어줄 수 있는 클래스를 하나 만든다.

```
// 중복된 필드변수를 하나로 모은 StarUnit 이라는 부모 클래스

public class StarUnit {
    private String name;
    private int hp;
    private int damage;
    private int armor;
    private int kill;
    private int moveSpeed;

    public StarUnit() {
    }
}

// StarUnit이라는 부모클래스를 상속 받은 Marine 클래스
public class Marine extends  StarUnit{
//    private String name;
//    private int hp;
//    private int damage;
//    private int armor;
//    private int kill;
```

```
//    private int moveSpeed;

// 필드변수 선언없이 부모 클래스에 있는 필드변수를 가져다 쓸 수 있다.
    public Marine() {
        this("마린", 200,10,0,0,1);
    }
}
```

▼ 클래스 상속

extends 뒤에 상속받을 부모클래스를 작성한다

다른 언어와 달리 자바는 다중 상속을 허용하지 않아,
extends 뒤에는 단 하나의 부모 클래스만 와야한다

접근 제어자가 private인 필드변수와 메소드는 상속을 받더라도 사용할 수 없다

```
public class 자식클래스 extends 부모클래스 {
}

public class 자식클래스 extends 부모클래스A, 부모클래스2{
}
```

▼ 부모 생성자 호출

super( );

부모 클래스에 있는 필드를 상속받아 올 경우 super(); 을 반드시 작성해야한다.

```
public class Marine extends  StarUnit{

public Marine(String name, int hp, int damage, int armor,
        super(name, hp, damage, armor, kill, moveSpeed);
    }
}
```

▼ 메소드 재정의(메소드 오버라이딩)

메소드 오버라이딩(=메소드 재정의, 메소드 덮어쓰기)

부모 클래스에 있는 메소드를 자식 클래스 내에서
입맛에 맞게 재정의하여 사용하는 것

메소드가 오버라이딩되었다면 해당 부모 메소드는 숨겨지고, 자식 메소드가 우선적으로 사용된다.

부모 메소드 호출

메소드 오버라이딩의 특징에 따라 부모 메소드의 일부분만 수정하고 싶더라도
전체를 다시 작성해야하므로 .super.method( );를 사용하여
숨겨진 부모 메소드를 호출할 수 있다.

부모클래스 Parent

```
public class Parent {
    private String name;
    private int age;

    public Parent(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void hello(){
        System.out.println("안녕하세요. 어른입니다.");
    }
```

Parent 클래스를 상속받는 Child 클래스

자식클래스에서 부모클래스의 메소드를 오버라이딩하면

오버라이딩한 자식클래스 내의 메소드가 실행된다.

```
public class Child extends Parent{

private String school;

public Child(String name, int age,String school) {
        // super는 부모 객체를 의미
        // super()는 부모의 생성자
        super(name, age); // 부모의 생성자를 앞에
        this.school = school; // 자식 생성자는 뒤에
    }


// Parent에서 만든 hello 메소드를 오버라이딩하기
    // 메소드명 입력 후 자동완성 이용
public void hello() {
        System.out.println("안녕하세요. 어린이입니다.");
```

```
    }
}
```

▼ final 클래스, final 메소드

final 클래스 : final 키워드를 class 앞에 붙이면 최종적인 클래스이므로 더 이상 상속할 수 없는 클래스가 된다.

따라서 final클래스는 부모 클래스가 될 수 없어 자식 클래스를 만들 수 없다.

final 메소드 : final class와 마찬가지로 최종적인 메소드가 되므로 오버라이딩할 수 없게 된다.

즉  자식 클래스에서 부모 클래스의 메소드를 재정의할 수 없이 그대로 가져다써야한다

▼ 타입 변환

자동 타입 변환(자동형변환)

자식 타입 == 부모 타입으로 취급하는 자동 형 변환

부모 타입으로 자동 타입 변환된 이후에는 부모 클래스에 선언된 필드와 메소드만 접근 가능하다

**자식 클래스에서 오버라이딩된 메소드가 있다면 부모 메소드 대신 오버라이딩된 메소드가 호출된다.**

📚 강제 타입 변환

자식 타입은 부모 타입으로 자동 변환되지만, 반대로 부모 타입은 자식 타입으로 자동 변환되지 않아
**캐스팅 연산자( = (부모타입명) )로 강제 타입 변환을 해야 한다**.

▼ 다형성

📚 다형성 (Polymorphism)
하나의 객체가 여러 타입을 가질 수 있는 성질
상속으로 인해 자식 객체가 부모 객체가 될 수 있고,
부모 객체가 캐스팅 연산자를 사용하여 강제 형변환할 수 있는
강제 형변환과 자동형변환의 성질을 '다형성'이라 칭한다.

```
StarUnit starZealot = zealot;
        StarUnit starHigh = highTemplar;

// StarUnit을 담을 수 있는 ArrayList
        ArrayList<StarUnit> starList = new ArrayList<>();
        starList.add(starZealot);
        starList.add(zealot2); // Zealot -> StarUnit으로 자
        starList.add(zealot3);
        starList.add(zealot4);
        starList.add(zealot5);
        starList.add(zealot6);
        starList.add(highTemplar); // highTemplar -> StarU

// StarUnit으로 강제 형변환되어 HighTemplar 자식 객체에 있는 고유한
```

```
// StarUnit -> HighTemplar 자식 객체도 다시 강제 형변환
((HighTemplar)starList.get(6)).psionicStorm();
```

▼ 객체 타입 확인 instance of

객체 instanceof 클래스

좌측의 객체가 우측의 클래스 타입으로 형변환이 가능하다면 true

가능하지 않다면 false 출력

```
// 1번 ) starList에 HighTemplar로 형변환 할 수 있는 데이터가 있다
// 해당 변수에 phsionicStrom 메소드를 실행하라는 코드
if (starList.get(i) instanceof HighTemplar){
                ((HighTemplar)starList.get(i)).psionicStor
            }


// 2번 ) starList에 "마린"이라는 이름을 가진 개체가 있다면
// 해당 개체에 stimpack이라는 메소드를 실행하라는 코드
if (starList.get(i).getName().equals("마린")){
                ((Marine) starList.get(i)).stimpack();
            }


// 일반적으로 부모 객체는 자식 클래스로 형변환할 수 없다.
        System.out.println( star instanceof Marine);
        System.out.println( star instanceof Zealot);

// 일반적으로 자식 객체는 부모 클래스로 형변환이 가능하다.
        System.out.println( marine instanceof StarUnit);
```

```
        System.out.println( zealot instanceof StarUnit);

// 부모 클래스로 형변환 되었던 자식 객체는
        // 다시 자식 클래스로 형변환이 가능하다.
        StarUnit temp = marine;
        System.out.println(temp instanceof Marine); // ->
```

▼ 추상 클래스

여러 클래스들의 공통적인 필드나 메소드를 추출해서 선언한 클래스를 의미한다

따라서 클래스들의 부모 역할로 주로 쓰인다

▼ 추상 클래스 선언 abstract class명

- new 연산자를 이용해서 객체를 직접 만들지 못하고 상속을 통해 자식 클래스만 만들 수 있다
- 추상 클래스도 필드, 메소드를 선언할 수 있고 자식 각체가 생성될 때 super( )로 추상 클래스의 생성자가 호출되기 때문에 생성자가 반드시 있어야 한다

▼ 추상 메소드와 재정의

abstract 리턴타입 메소드명(매개변수);

- 메소드 선언부(리턴타입, 메소드명, 매개변수)만 동일하고 실행 결과를 쓰임에 따라 클래스마다 다르게 만들고 싶은 경우 사용한다
- 추상 메소드를 호출해서 사용하는 클래스에서 해당 추상 메소드를 오버라이딩하여 실행 부분을 작성해야만 한다