7_28_2022__java_Constructor, Array, Extends

Step05_Constructor

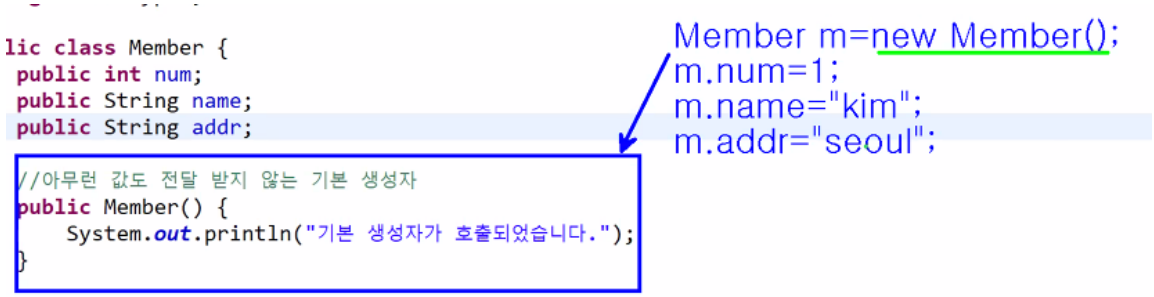
~Step08_Extends

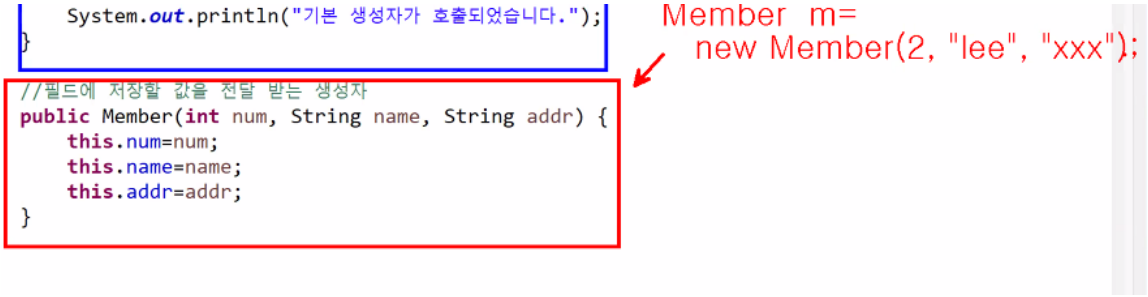
생성자 Costructor

- 클래스 명과 동일하다
- 메소드 모양과 유사 하지만 리턴 type이 없다
- 객체를 생성할 때(new할때) 호출된다.
- 객체를 생성하는 시점에 무언가 준비 작업을 할 때 유용하다.
- 생성자를 명시적으로 정의 하지 않아도 기본 생성자는 있다고 간주된다.
- 여러개 정의할 수 도 있다.(객체를 생성하는 방법이 여러가지 알 수 있다.)

생성자의 다중 정의(over loading)

- 생성자가 여러개가 지정되어 있는 상태





같은 class 안의 두가지 생성자 불러오는 방법이 다르다.

접근 지정자

- public

  어디서나 접근 가능

- private

  동일한 객체 혹은 동일한 class 안에서만 접근 가능

- default

  같은 package 안에서만 접근 가능

- protected

  동일한 package 혹은 상속관계에서 자식에서 접근 가능

접근 지정자를 붙이는 위치

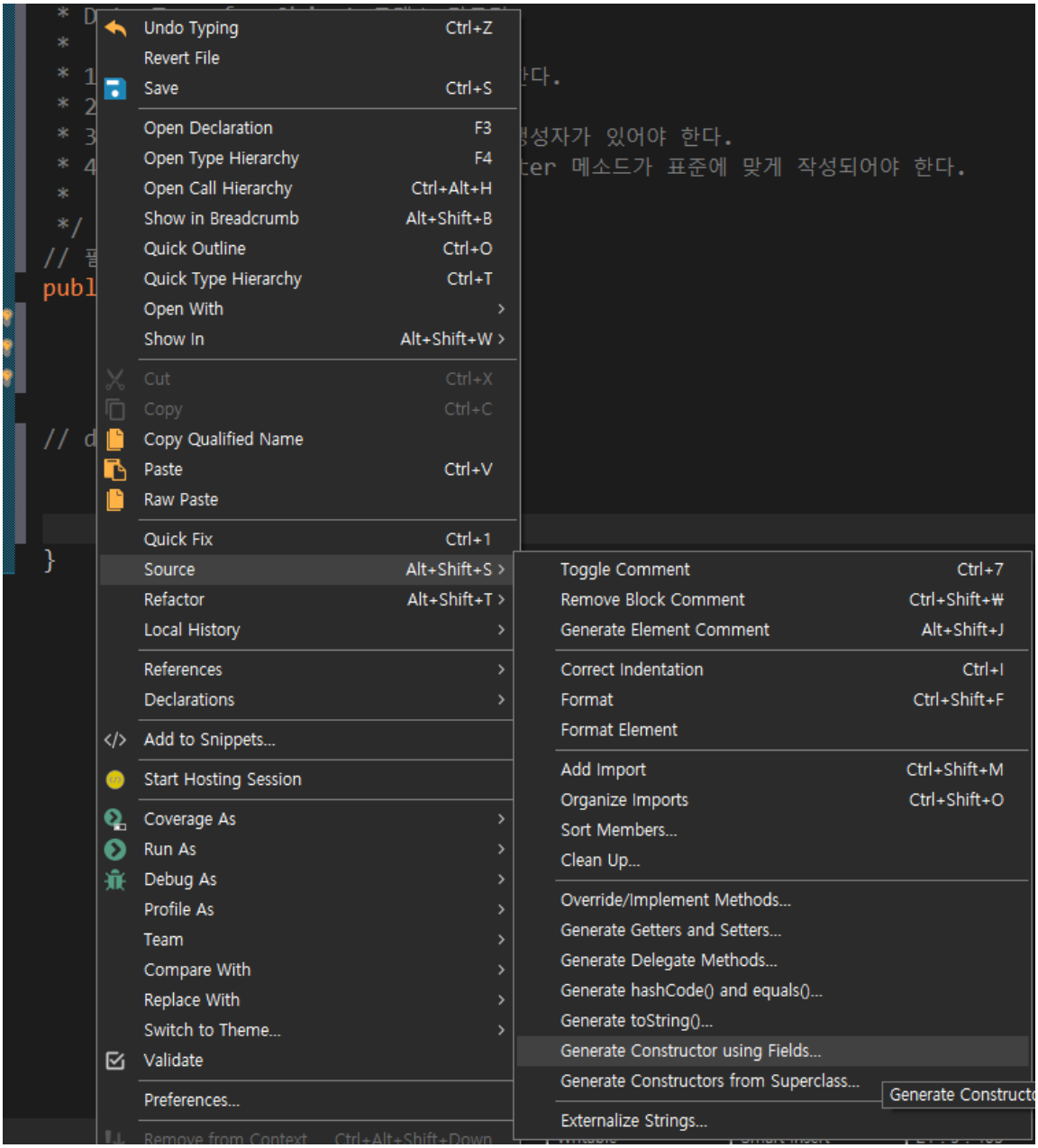
- 클래스를 선언 할때
- 생성자
- 필드
- 메소드
- 클래스는 default 와 public 두가지의 접근 지정자만 지정 가능하다
- 접근 지정자를 생략한것이 default 접근 지정자다

Data Transfer Object

- Data Transfer Object 클래스 만들기
  - 필드의 접근지정자를 private로 설정한다.
  - default 생성자가 있어야 한다.
  - field에 저장할 모든 값을 전달받는 생성자가 있어야 한다.

- 필드에 접근할 수 있는 getter, setter 메소드가 표준에 맞게 작성되어야 한다.

```
package test.mypac;
public class MemberDto {
  private int num;
  private String name;
  private String addr;

// default 생성자 만들기
  public MemberDto() {
  }

// field에 저장할 모든 값을 전달받는 생성자 만들기
  public MemberDto(int num, String name, String addr) {
    this.num = num;
    this.name = name;
    this.addr = addr;
  }
// setter method 만들기
  public void setNum(int num) {
    this.num = num;
  }
  public void setName(String name) {
    this.name = name;
  }
  public void setAddr(String addr) {
    this.addr = addr;
  }
// getter method 만들기
  public int getNum() {
    return num;
  }
  public String getName() {
    return name;
  }
  public String getAddr() {
    return addr;
  }
}
```



우클릭하면 간단하게 만들 수 있음

Extends

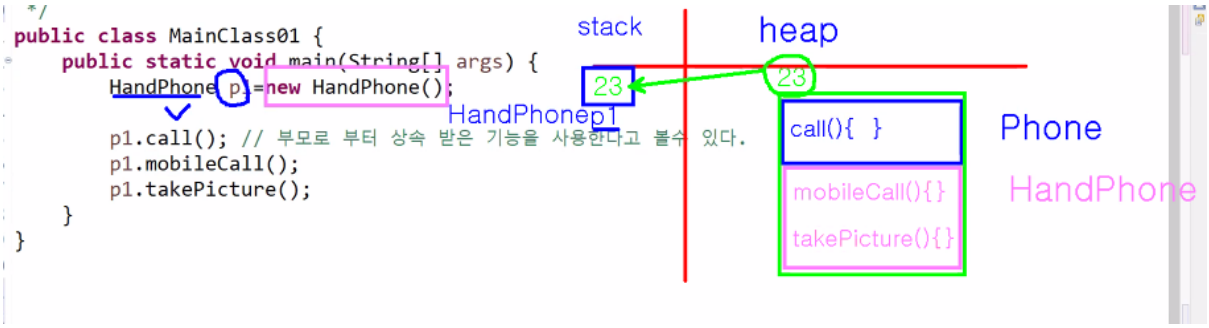
- 잘 만들어진 기능들을 추가해서 사용할 수 있다.

```
package test.mypac;

public class HandPhone extends Phone {
  // 생성자
  public HandPhone() {
    System.out.println("HandPhone 생성자 호출됨");
```

```
  }

  public void mobileCall() {
    System.out.println("이동중에 전화를 걸어요");
  }

  public void takePicture() {
    System.out.println("30만 화소의 사진을 찍어요");
  }

}
```

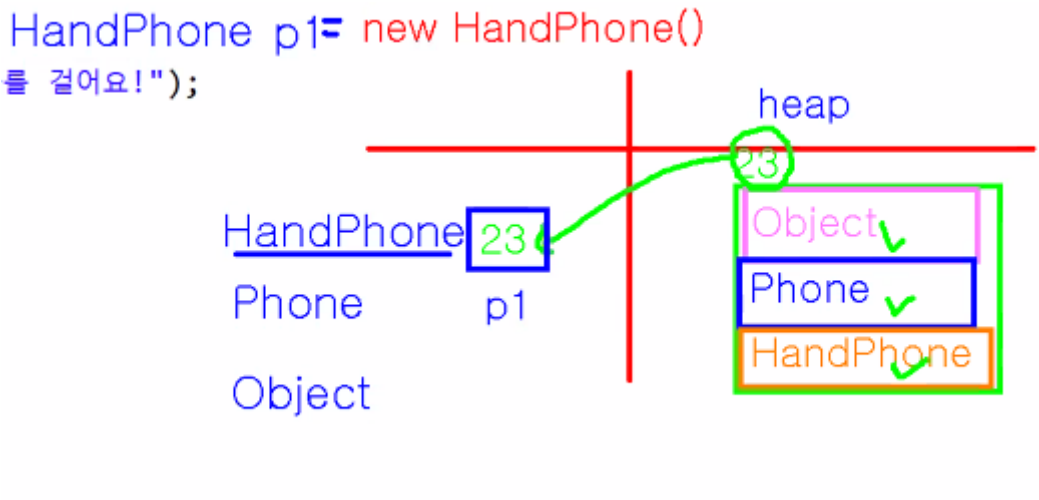
- Phone 클래스를 상속받은 HandPhone 클래스로 객체를 생성하면 Phone 객체가 미리 만들어지고 HandPhone 객체가 만들어진다. 그리고 그 두개의 객체가 동일한 heap 영역에 저장되어서 동일한 참조값으로 관리가 된다.

```
package test.main;
import test.mypac.HandPhone;
public class MainClass01 {
  public static void main(String[] args) {
    HandPhone p1=new HandPhone();
    p1.call(); //부모로부터 상속 받은 기능을 사용한다고 볼 수 있다.
    p1.mobileCall();
    p1.takePicture();
  }
}
```



동일한 사물함에 두 객체가 모두 들어간다.

- 아무것도 상속받지 않으면 자동으로 object class를 상속받음 (extends object)



다양한 타입을 가질 수 있다. (다형성)

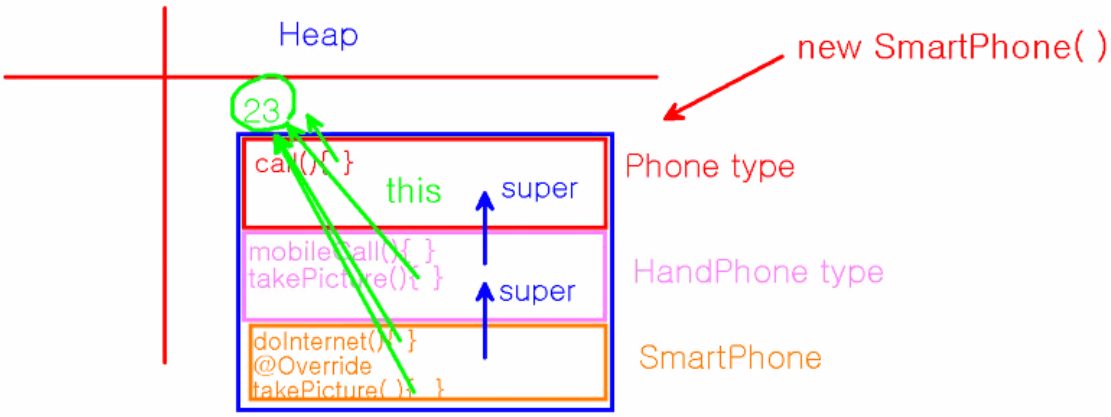
> 💡 다양한 타입을 가질 수 있다 - 다형성

casting 연산자

```
HandPhone p2=(HandPhone)p1;   //  HandPhone type 으로 casting 해서 p2에 담는다.
SmartPhone p3=(SmartPhone)p1; //  SmartPhone type 으로 casting  해서 p3에 담는다.
p3.exploreSafari();
```

부모 클래스를 불러오는 뻘짓을 하는 이유 == super class 를 사용하는게 컴퓨터 입장에서 유연하게 연산 할 수 있다.

- this →자신의 참조값을 가리킴
- super→부모를 가리킴(override했어도 호출해야 할때가 있음)



```
SmartPhone에서

this.takePicture(); 하면  SmartPhone 가리킴
super.takePicture(); 하면 HandPhone 가리킴
```

- super() ← 부모생성자를 호출함(최우선이어야함)
  - 부모생성자가 필요한 값이 있으면 전달해 줄 수 있음

1. 여름이니 Car클래스를 상속받아서 CampingCar를 만들기
2. CampingCar 만의 메소드도 추가
3. 만든 CampingCar 클래스로 객체를 생성하고 메소드도 호출하기
4. 어떻게 테스트 했는지 CampingCar클래스와 테스트한 main 메소드의 내용을 카톡에 올리기

```
package test.auto;

public class CampingCar extends Car {

  public CampingCar(Engine engine) {
    super(engine);
  }

  public void camping() {
    System.out.println("차박도 가능한가?");
  }

  @Override
  public void drive() {
    System.out.println("달리기도 하고!");
  }
}
```

```
package test.main;
import test.auto.Engine;
import test.auto.Haku;
public class MainClass07 {
  public static void main(String[] args) {
    //Haku 객체를 생성해서 참조값을 car1 이라는 지역 변수에 담아보세요
    Haku car1 = new Haku(new Engine());
    //car1 에 들어있는 참조값을 이용해서 달리기도 하고 통통 튀기도 해 보세요
    car1.drive();
    car1.goCartFeeling();
```

```
    car1.camping();
  }
}
```