7_27_2022__java_Class, Array

Step03_Class

~Step07_Array

클래스

- 객체의 설계도 역할
  - field와 method로 구성되어 있다.
  - new해서 다른 클래스를 불러올 수 있다.
  - 코드 한 class 파일은 설계도일 뿐이다.
  - 객체는 heap 영역에 만들어진다.
  - 불러온 클래스의 명 자체가 타입이 된다.
- static field or static method를 감싸주는 역할
  - static이 붙은 field나 method는 클래스와 함께 static영역에 올라간다.

```
package test.main;
import test.mypac.MyUtil;
public class MainClass04 {
  public static void main(String[] args) {
    /*
     * MyUtil 클래스에 있는 send() 메소드를 호출하고 싶다면?
     * static 메소드 이기 때문에 new 하지 않고
     * import 후에 클래스명에 .을 찍어서 호출 할 수 있다.
     */

    // static 메소드 호출
    MyUtil.send();
    //static 필드 참조
    String v=MyUtil.version;
  }
}
```

- 예제 - 스타크래프트 공격 유닛을 객체로 표현하고 싶다

```
class Zealot{
  public int hp=160;
  public static int attackPower=16;
  //모든 질럿의 공격력은 같고 업해도 같이 업되니까 static
  public void attack(){
    System.out.println(Zealot.attackPower+"의 데미지를 입혔습니다.");
  }
}

Zealot z1=new Zealot();
Zealot z2=new Zealot();
//참조값 두개


z1.attack();
z2.attack();
//16데미지씩

Zealot.attackPower += 2;
//attackPower는  static영역에 저장되어 있어서 다똑같이 반영됨

z1.attack();
z2.attack();
//18데미지씩
```

메소드

- 메소드를 만들 때 고려해야할 것
  - 접근 지정자
  - static or nonstatic
  - return type
  - 메소드 명
  - 메소드에 전달하는 인자의 갯수와 data type

  - public -> 이 메소드는 어디에서든 접근 가능
  - void -> 이 메소드는 어떤 값도 리턴하지 않음
  - walk -> 메소드명

- walk() -> 이 메소드는 어떤 값도 인자로 전달 받지 않는다.

```
package test.mypac;

public class myObject {

  public void walk() {
    System.out.println("걸음을 걸어요");
  }
  //int type을 리턴해주는 메소드
  public int getNumber() {
    return 10;
  }
  //String type 을 리턴해주는 메소드
  public String getGreeting() {
    return "안녕하세요";
  }
  //Car type을 리턴해주는 메소드
  public Car getCar() {
    return new Car(); //같은 패키지 안에 있어서 import 할 필요가 없음.
  }

  //int type을 메소드의 인자로 전달 받는 메소드
  public void setNum(int num) {
    System.out.println("num: :"+num);
  }
  //String type을 메소드의 인자로 전달 받는 메소드
  public void setName(String name) {
    System.out.println("name: "+name);
  }
  //Radio Type 을 메소드의 인자로 전달받는 메소드
  public void useRadio(Radio r) {
    r.listenMusic();
  }
  //Gun type 과 Arrow  type을 메소도의 인자로 전달 받는 메소드
  public void attack(Gun g, Arrow a) {
    System.out.println("총과 활로 공격을 해요!");
  }
}package test.mypac;
public class myObject {

  public void walk() {
    System.out.println("걸음을 걸어요");
  }
  //int type을 리턴해주는 메소드
  public int getNumber() {
    return 10;
  }
  //String type 을 리턴해주는 메소드
  public String getGreeting() {
    return "안녕하세요";
  }
  //Car type을 리턴해주는 메소드
  public Car getCar() {
    return new Car(); //같은 패키지 안에 있어서 import 할 필요가 없음.
  }
```

```
  //int type을 메소드의 인자로 전달 받는 메소드
  public void setNum(int num) {
    System.out.println("num: :"+num);
  }
  //String type을 메소드의 인자로 전달 받는 메소드
  public void setName(String name) {
    System.out.println("name: "+name);
  }
}
```

```
package test.main;

import test.mypac.Arrow;
import test.mypac.Gun;
import test.mypac.Radio;
import test.mypac.myObject;

public class MainClass04 {
  public static void main(String[] args) {
    myObject obj = new myObject();
    obj.setNum(1872);
    obj.setName("Dubu");

    // useRadio() 메소드를 호출해 보세요

    // 메소드의 인자로 전달할 값이 이미 준비 되어 있을 수도 있다
    Radio r = new Radio();
    // 이미 준비 되어 있다면 메소드를 호출하면서 준비된 값을 참조해서 전달하면 된다.
    obj.useRadio(r);
    obj.useRadio(new Radio());
//    r.listenMusic();

    //attack() 메소드를 호출해 보세요
    Gun g = new Gun();
    Arrow a = new Arrow();
    obj.attack(g, a);
  }
}
```

Static메소드

```
package test.mypac;

public class Messenger {
  // String type 을 인자로 전달받는 static 메소드
  public static void sendMessage(String msg) {
    System.out.println(msg + " 를 전송합니다.");
  }
```

```
  // String type 을 리턴해주는 static 메소드
  public static String getMessage() {
    return "Hello";
  }

  public static void useRadio(Radio r) {
    r.listenMusic();
  }
}
```

```
package test.main;

import test.mypac.Messenger;
import test.mypac.Radio;

public class MainClass05 {
  public static void main(String[] args) {
    // Messenger 클래스의 sendMessage(), getMessage(), useRadio() 메소드를 호출해 보세요
    Messenger.sendMessage("두부야 안녕?");
    Messenger.getMessage();
    Messenger.useRadio(new Radio());

  }
}
```

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.)

```
package test.mypac;
public class Student {
  // 생성자
  public Student() {
    System.out.println("Student 클래스의 생성자 호출됨");
  }
}
```

접근 지정자 private

- 다른 프로젝트에서 접근이 안됨

Wrapper Class

```
public class MainClass01 {
  public static void main(String[] args) {
    System.out.println("main 메소드가 시작 되었습니다.");
    // 기본 data type
    int num1 = 10;
    // 참조 data type
    Integer num2 = 10;

    // 참조 데이터 type 이지만 기본 데이터 type 처럼 사용할 수 있다.
    int result1 = num2 + 1;
    int result2 = num1 + num2;
    Integer result3 = num1 + num2;
```

▼ 기본 data type 의 참조 data type

- byte : Byte
- short : Short
- int : Integer
- long : Long

- float : Float
- double : Double

- char : Character
- boolean : Boolean

- 때로는 기본데이터 type의 참조 데이터 type 이 필요할 때가 있다.
- 기본 데이터 type 을 객체에 포장(boxing) 하는 형태
- boxing 과 unboxing 은 자동으로 되기 때문에 프로그래머가 신경을 쓸 필요는 없다.

Array

배열도 참조 데이터 타입

```
package test.main;

public class MainClass01 {
  public static void main(String[] args) {
    System.out.println("메인 메소드가 시작되었습니다.");
    // int type 5개를 저장하고 있는 배열
    int[] nums= {10, 20, 30, 40, 50};
    // double type 5개를 저장하고 있는 배열
    double[] nums2= {10.2, 10.2, 10.3, 10.4, 10.5};
    //boolean type 5개를 저장하고 있는 배열
    boolean[] truth= {true, false, false, true, true};
    //String type(참조 데이터 type) 5개를 저장하고 있는 배열
    String[] names= {"두부", "유키", "소주", "큐디", "하쿠"};
    }
}
```

- int[ ] type
- double[ ] type
- boolean[ ] type
- String[ ] type

```
//배열의 각각의 방 참조하기
int result1=nums[0]; //10
double result2=nums2[1]; // 10.2
boolean result3=truth[2]; // false
String result4=names[3]; //"큐디"

//nums 배열을 복제해서 새로운 배열을 얻어내서 a에 대입하기
int[] a=nums.clone();
// nums 안에 있는 배열의 참조값을 b에 대입하기
int[] b=nums;

//배열의 방의 갯수 참조
int size=nums.length;
```

a = clone → 배열안의 정보만 복사해오고 key값은 다름

b = key값 자체를 복사

다른방법

```
package test.main;
```

```
public class MainClass02 {
  public static void main(String[] args) {
    // 0으로 초기화된 방 3개짜리 int[] 객체 만들어서 참조값을 지역변수 nums에 담기
    int[] nums= { 0, 0, 0};
    nums[0]=10;
    nums[1]=20;
    nums[2]=30;

    // 0 으로 초기화된 방 100개 짜리 int[] 객체 만들어서 참조값을 지역변수 num2 에 담기
    int[] nums2=new int[3];
    nums2[0]=100;
    nums2[1]=200;
    nums2[2]=300;
    //nums2[3]=400; // ??? 없는 방번호를 참조하면 exception 이 발생한다.

    System.out.println("마무리 작업을 하고 app을 종료합니다.");
  }
}
```