Java OOP

OOP 특징

- Encapsulation : 하나의 클래스 안에 데이터와 기능을 담아 정의하고, 중요한 데이터나 복잡한 기능 등은 숨기고, 외부에서 사용에 필요한 기능들만 공개하는 것.
- Inheritance : 객체 정의 시 기존에 존재하는 객체의 속성과 기느을 상속받아 정의하는 것.
- Polymorphism, 다형성 : 같은 타입 또는 같은 기능의 호출로 다양한 효과를 가져오는 것
- Abstraction : 현실 세계에 존재하는 객체의 주요특징을 추출하는 과정

OOP의 3대 특징

- Inheritance, Polymorphism, Abstraction

OOP의 4대 특징

- Encapsulation, Inheritance, Polymorphism, Abstraction

Class와 Object

- Object : 시스템의 대상이 되는 모든 것, 구체적인 표현 대상이 있다.
  - 현실 세계에 존재하는 실체
  - 예)학생 object : A학생, B학생 등을 표현
- Class : 구체적인 object 들을 분석해 보고 공통적인 내용들을 추상화 해서 programming 언어로 표현한 것.

Class 만들기

- attribue : 정적인 특성
- behavior : 동적인 특징
- package : 모듈화
- attributs : member variables
- behavior : methods

Constructor, 생성자

```
package com.ssafy;

public class Phone {

    public String name = "Galaxy Note";
    public char color;
```

```
    public int price;

    public int getRealDebt() {
        return 1000;
    }
}

package com.ssafy;

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

        Phone phone = new Phone();

        phone.name = "Galaxy Note";
        phone.color = 'B';
        phone.price = 10000;

        System.out.println( phone.getRealDebt() );
    }
}
```

- Class Type으로 변수를 선언하고, `new` 라는 keyword를 사용하는데, 그 뒤에는 생성자 (`Constructor`)가 온다.
- new와 뒤의 생성자를 보고 그에 맞게 memory allocation을 수행한다.
  - stack 영역에 phone이라는 변수에 주소값 1000 저장
  - heap 영역의 1000이라는 주소에 name, color, price 변수가 들어있음
  - name에는 또 heap영역의 2000이라는 주소가 들어있음
  - heap 영역의 2000이라는 주소에는 이름을 저장.
- 생성자는 `Class이름 + ()` 구조를 가지고 있다.
- Class가 제공하는 생성자가 없을 경우, 즉 Class를 만들 때, 아무런 생성자를 만들지 않으면, Compiler가 기본생성자(`Default Constructor`)를 자동으로 만들어 준다.
- 기본 생성자는 Parameter가 없는, 특별한 작업을 수행하지 않는 가장 단순한 생성자다.
- 만약 우리가 생성자를 만들면 Compiler는 생성자를 만들어주지 않는다.
- 생성자는 Parameter를 가질 수 있다.
- Parameter를 달리하는 여러 개의 생성자를 만들 수 있다.
- 생성자가 여러 개 제공되면, 외부에서는 다양한 방법으로 객체를 만들 수 있다.
- Class는 생성자의 parameter를, 객체를 생성할 때 사용한다.
  - 보통 member variable들의 값을 설정한다.
- 생성자 앞 `public`
  - 외부에서 생성자를 호출할 때, 아무런 제한이 없다는 의미

생성자의 구조

1. 클래스 이름과 동일한 이름의 함수

2. 리턴 타입은 적지 않음
3. 객체의 생성시에 호출됨

```
class Test{
    public Test(){}
}

--------------------

Test t = new Test();
```

4. this() : 생성자 안에서 다른 생성자 호출

```
//case1
Car(){//defalut 생성자
    this(0); // 밑에 int값을 받는 생성자에 0을 매개변수로 전달해서 부탁함. 대신 생성해줘.
}
Car(int num){ // parameter 있는 생성자
    this.num = num;
}



//case2
Car(int num){//defalut 생성자
    this(num, "xyz");
    // 밑에 int값을 받는 생성자에 입력받은 0과
    // "xyz"를 매개변수로 전달해서 부탁함.
    // 대신 생성해줘.
}
Car(int num, String model){ // parameter 있는 생성자
    this.num = num;
    this.model = model;
}
```

Member Variables Set

1. Class를 정의하면서 값을 부여한다.
2. 생성자의 Parameter에 값을 전달한다.
3. 객체를 만든 후에 직접 값을 부여한다.

- Class 안에 선언된 멤버 변수는 new를 통해 객체가 생성될 때 기본 값(`Default Value`)이 자동으로 부여된다.
  - 기본형은 0과 0에 준한 값
  - Object 타입은 null 값

Encapsulation

- Class를 구성할 때, 기본적으로 멤버변수와 메소드를 외부에 노출하지 않도록 가정하고, 필요한 경우에 한해, 외부에 노출하는 것을 의미한다.
- 외부에 노출할 경우, `Setters & Getters` 를 제공하여 외부와 소통한다.
- `접근제한자(Access Modifier)` 를 이용하여 외부에 어느 정도 노출할 것인지를 결정한다.

Member Variables Setters

- 멤버변수 값을 외부에서 마음대로 접근할 수 없도록 하고, 대신 멤버 변수 하나에 대해 값을 Set 할 수 있는 특별한 method를 제공할 수 있다.
    - 예를 들어 잘못된 값이 입력되면 걸러줄 수 있다.
- 이를 통해, 외부에서는 멤버 변수의 값을 변경 요청하게 되고, Class 내부에서는 이 method 안에서 요청에 대한 처리를 수행한다.
- 이러한 method를 `Setters` 라고 부른다.
- By 생성자

```
Phone phone = new Phone("A");
-----------------------------------

public Phone(String name){
    this.name = name;
}
```

- By Setters

```
Phone phone = new Phone("A");
-----------------------------------

public void setName(String name){
    this.name = name;
}
```

- Setters를 제공한 이후에는 멤버 변수 앞에 있는 modifier를 public에서 `private` 으로 변경해 준다.

Member Variables Getters

- Getters : 외부에서 값을 읽고자 할 때, 직접 읽게 하지 않고, 별도로 제공하는 method.
- By Getters

```
public String getName(){
    return name;
```

```
        }
        ----------------------------

        Phone phone = new Phone("A");
        System.out.println(phone.getName());
```

Method

- **public void go () {}**
    - public : modifier
    - void : return type
    - go : method name
    - () : parameter list
    - {} : method body

Generate Getters and Setters : 자동생성

- source - Generate Getters and Setters
- Alt+Shift+S -> R

ObejctArray

```
// 오류
Phone [] phoneArray = new Phone[5]; // 담을 자료형만 만들어 둔 것.

for(int i=0; i<phoneArray.length; i++) {
    phoneArray[i].setPrice(i*200); // 객체가 생성되지 않았음.
}
```

```
// 오류
Phone [] phoneArray = new Phone[5]; // 담을 자료형만 만들어 둔 것.

for( int i=0; i<phoneArray.length; i++ ) {
        phoneArray[i] = new Phone();
        phoneArray[i].setPrice(i*2000);
}
```

JVM Memory Overview

- Java는 High-Level Language로 Memory Management로부터 자유롭다.
- JVM은 주요하게 3개의 메모리 영역을 가지고 있고, 용도는 다음과 같다.
    - Class Area(Method Area) : for Class, Static, Method

    - Heap : for Objects
    - Stack : for Call

- ```
  public class MemoryTest {

      public static void main(String[] args) {

          int i = 10;
          String s1 = "Hello";
          String s2 = new String("World");
          Phone phone = new Phone("S20");

          System.out.println(i);
          System.out.println(s1);
          System.out.println(s2);
          System.out.println(phone);
      }
  }
  ```

    - Class Area
        - MemoryTest(Class)
        - String
        - Phone(Class)
    - Heap
        - 1000 : "Hello"
        - 2000 : "World"
        - 3000 : "S20"
    - Stack
        - i : 10
        - s1 : 1000
        - s2 : 2000
        - phone : 3000

- JVM은 Heap에 만들어진 객체 중 더 이상 참조되지 않은 것들을 대상으로 적절한 시점에 Garbage Collection 작업을 수행한다.

- 3가지 기억하기

    i. Garbage Collection에 직접 관여할 수 없다.
    ii. 자동으로 처리된다는 점은 Coding 관점에서느 장점이지만, 운영 관점에서는 단점이 된다.
    iii. 불필요한 객체 생성을 지양한다.

String Class

- Java에서 문자열은 Phone Class 처럼 Class로 정의하고 기본으로 제공된다.
  ```

    - java.lang package
- Primitive Type 처럼 new 없이 바로 문자열을 줄 수도 있다.
- Reference Type 처럼 new 연산자를 이용하여 값을 줄 수도 있다.
- 코드 실행 결과는?

```
public static void main(String[] args) {
    int i1 = 10;
    int i2 = 10;

    String s1 = "Hello";
    String s2 = "Hello";
    String s3 = new String("Hello");
    String s4 = new String("Hello");

    if( i1 == i2 ) { System.out.println("i1 i2 Same"); }
    if( s1 == s2 ) { System.out.println("s1 s2 Same"); }
    if( s3 == s4 ) { System.out.println("s3 s4 Same"); }
}
```

```
i1 i2 Same
s1 s2 Same
```

    - == 는 두 변수의 memory 값을 비교한다.
    - i1과 i2는 local 변수이므로 Stack에 만들어지고, Primitive Type 이므로 그 변수에 값이 저장된다.
    - s3와 s4는 각각 new에 의해 Heap에 서로 다른 객체를 생성하고 그 주소값을 저장한다.
    - "Hello" : 객체 상수로서 String Constant Pool에 관리(Heap에 만들어 짐) 되고, 재사용 됨
    - `s3 s4 Same` : if(s3.equals(s4))를 사용하면 출력 된다.
- 문자열의 내용이 같은 지 확인하는 법 : equals()

StringBuilder

- 복수 개의 String을 이어서 새로운 String을 만들게 되는 경우
    - + operator를 이용하면 그 만큼 String 객체가 새로 만들어진다.
        - 불필요한 객체가 많이 만들어져서 성능에 영향을 미칠 수 있다.
        - 간단한 + 의 나열은 compiler가 내부적으로 StringBuilder를 사용해 처리한다. 그러나, Loop 등의 코드 안에서는 안한다.
    - StringBuilder 사용 예

```
public static void main(String[] args) {

    String s1 = "Hello";
```

```
        String s2 = "World";
        String s3 = s1 + ", " + s2;

        System.out.println(s3);

        StringBuilder sb = new StringBuilder("");
        sb.append(s1).append(", ").append(s2);

        System.out.println(sb.toString());

        String[] strArray = {"Hello", ", ", "World" };
        String str = "";
        for( String s : strArray ) {
            str += s;
        }
        System.out.println(str);

        sb.setLength(0);
        for( String s : strArray ) {
            sb.append(s);
        }
        System.out.println(sb);

    }
```

- .append로 이어줄 수 있음.

toString()

- 객체의 출력값을 내가 원하는 형식으로 지정한다.

package

- Java 파일을 Module 별로 나누어서 관리하는 것이 일반적이다.
- 파일을 계층적 구조로 관리하기 위해 폴더를 사용하듯이, Java의 Class도 `package` 라는 구조를 통해 계층적으로 관리한다.
- 보통, www를 제외한 도메인의 역순 구조를 많이 사용한다.
  - www.ssafy.com이 도메인이면 com.ssafy가 기본 package가 된다.
  - 그 하위 package는 업무 구분 등으로 구성한다.

import

- 다른 package에 정의된 Java Module을 사용하고자 할때 import keyword를 사용한다.
- import 다음에 package 명을 쓴다.
- import를 사용하지 않으려면 Java Module 앞에 package 명을 붙여서 사용한다.
  - `package com.ssafy.day03.sub;`

```
    public class sub {
        public static void main(String[] args) {
            com.ssafy.day03.Phone p = new com.ssafy.day03.Phone();
    }
}
```

- Java.lang 패키지는 컴파일러가 자동으로 import 해준다.

Access Modifier, 접근 제어자

- public : 어떤 클래스라도 접근 가능
- protected : 동일 패키지내의 클래스 또는 해당 클래스를 상속받은 외부 패키지의 클래스에서 접근이 가능
- default : 해당 패키지 내에서만 접근이 가능
- private : 해당 클래스에서만 접근이 가능
- 개방성 public> protected> defaul> private

final

- 멤버 필드 : 값 변경 불가
- 메소드 : 오버라이딩 불가(부모가 자식에게 물려준 메소드를 재정의하는 것 불가)
- 클래스 : 상속 불가

static

- 객체들 간에 공유되는 유일한 값, 1개
- 객체 생성 없이 `클래스 이름` 으로 사용 가능.
- 멤버필드, 메소드, 블록에 사용
- Heap에 생성된 객체에는 static 붙은 값들은 만들어지지 않음
- Class Area에 생성되어있는 클래스 안에 있는 static 값을 참조해서 사용한다.
- `static 변수` 를 `클래스 변수` 라고도 한다.
- static이 붙지 않은 변수를 `인스턴스 변수` 라고도 한다.
- static 블록 : static{내용}
    - static 블록은 static main문 보다 먼저 실행된다.
    - main에서 본격 작업을 시작하기 전에 준비 작업을 시킨다.
- static method 안에서는 this, super 키워드 사용 불가
    - this와 super가 생성되는 시간이 더 늦음.

Singleton 디자인 패턴

- 객체 생성을 한 번만 수행하고 그 이후부터는 생성된 객체를 공유해서 사용하는 방식
- 조건
  i. field : private, static 수식어 붙어야 함
  ii. 생성자 : private
  iii. 객체를 만들어서 리턴해주는 public static method가 존재해야 함.

```
public class SingletonObject {

    //field : 자기 자신 타입의 변수.
    private static SingletonObject instance;

    // 생성자
    private SingletonObject() {}

     //public method: 외부에서 사용 가능한 메소드
    public static SingletonObject getInstance() {
        if(instance == null)
            instance = new SingletonObject();

        return instance;
    }


}

-----------------------------------------------------

public class SingletonObjectUser {

    public static void main(String[] args) {
        SingletonObject s1 = SingletonObject.getInstance();
        SingletonObject s2 = SingletonObject.getInstance();
        SingletonObject s3 = SingletonObject.getInstance();

        System.out.println(s1 == s2); // 주소값 비교
        System.out.println(s2 == s3); // 주소값 비교

    }

}

-----------------------------------------------------

true
true
```

객체의 종류

- VO(Value Object) : 값을 저장하는게 목적인 객체 ex) Book

    - data는 private
    - getter/setter
- DAO(Data Access Object) : VO같은 데이터에 접근해서 CRUD 작업을 하는 객체