

삼성 청년 SW 아카데미

Java

객체지향 프로그래밍

- 객체지향 프로그래밍
- 클래스
- 생성자

객체지향 프로그래밍

✓ 객체지향 프로그래밍 (OOP, Object Oriented Programming)

- 객체: 의사나 행위가 미치는 대상, 작용의 대상 / 세상의 모든 사물, 개념(유·무형) 등
- 객체(Object): 데이터와 관련된 알고리즘(메서드)를 하나의 단위로 묶어 놓은 것
- 객체지향 프로그래밍(OOP): 객체 단위로 코드를 작성하며, 객체 간의 상호작용으로 프로그램을 설계
- 객체 모델링 : 현실세계의 객체를 SW 객체로 설계하는 것

✓ 클래스 (Class)

- 객체(Object)를 만들기 위한 설계도(Blueprint)

✓ 인스턴스 (Instance)

- 클래스를 통해 생성된 객체
- 객체는 일반적 용어이며, 인스턴스는 특정 클래스를 이용해 생성된 하나의 객체를 지칭
- 인스턴스는 클래스의 한 사례(특정 클래스를 사용해서 객체를 생성하는 맥락)

✓ 객체지향 프로그래밍의 특징 (A PIE)

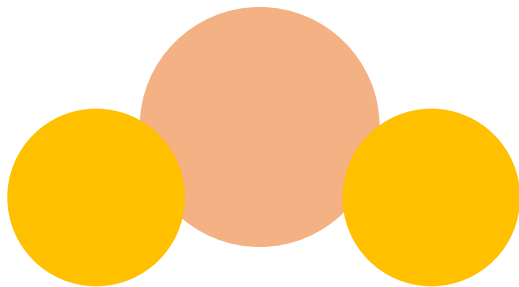
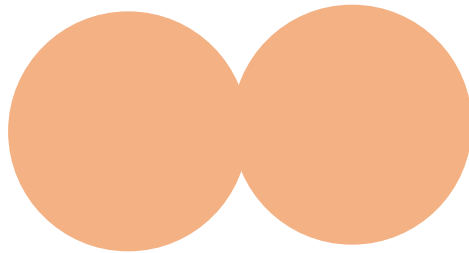
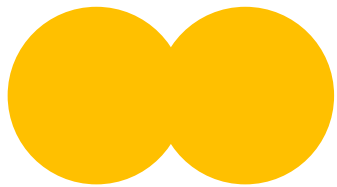
- Abstraction(추상화) : 객체의 불필요한 세부사항을 숨기고, 필요한 인터페이스만을 제공, 구현에 의존하지 않는 설계
- Polymorphism(다형성) : 상속 또는 구현 관계에 있을 때 객체들이 서로 다른 방식으로 동작하는 것
- Inheritance(상속) : 기존의 설계도를 재사용(확장), 하나의 클래스가 다른 클래스의 속성과 메서드를 물려받는 것
- Encapsulation(캡슐화) : 객체의 데이터와 메서드를 하나로 묶고, 외부로부터 객체의 세부 사항을 숨기는 것

✓ 객체지향 프로그래밍의 장점

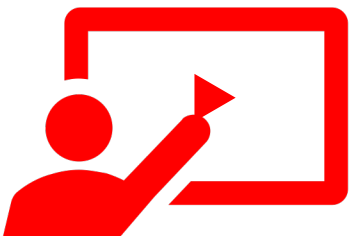
- 코드의 재사용성
- 유지보수성
- 유연성과 확장성

클래스

✓ 생각해 봅시다.



✓ 사람의 정보를 관리하자.



name : Yang
age : 45
hobby : 유튜브



name : Hong
age : 25
hobby : 골프

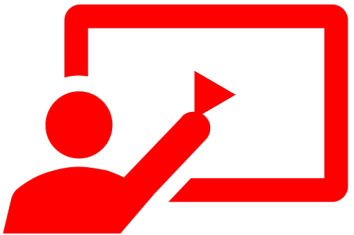
배열을 이용한 관리

```
String[] names = new String[2];  
names[0] = "Yang";  
names[1] = "Hong";
```

```
int[] ages = new int[2];  
age[0] = 45;  
age[1] = 25;
```

```
String[] hobbies = new String[2]  
hobbies[0] = "유튜브"  
hobbies[1] = "골프"
```


- ✓ 사람의 정보를 관리하자.



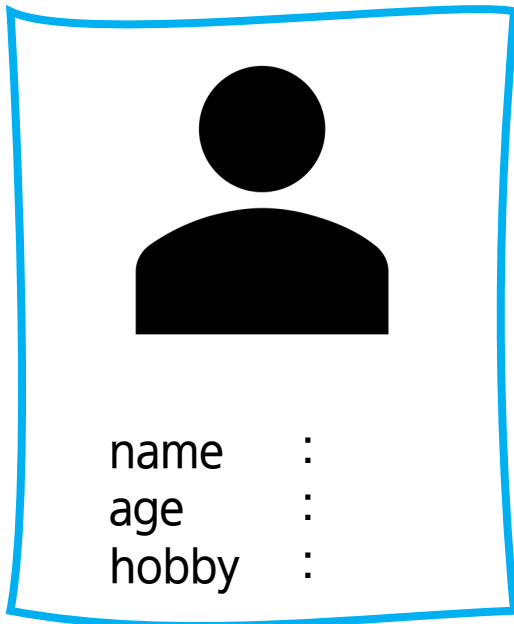
name : Yang
age : 45
hobby : 유튜브



name : Hong
age : 25
hobby : 골프

하나의 변수에 다양한 정보를 묶을 수는 없을까??


✓ 사람의 정보를 관리하자.



```
public class Person {  
    String name;  
    int age;  
    String hobby;  
}
```

✓ SSAFY 생의 하루 일과

1. 아침에 일어난다.
2. 교육장으로 대중교통을 이용하여 이동한다.
3. 오전 수업을 듣는다.
4. 점심을 먹는다.
5. 오후 수업을 듣는다.
6. 집으로 대중교통을 이용하여 이동한다.
7. 과제를 해결한다.
8. 잠을 잔다.



```
교육() {  
    오전 수업을 듣는다.  
    점심을 먹는다.  
    오후 수업을 듣는다.  
}
```

✓ SSAFY 생의 하루 일과

1. 아침에 일어난다.
2. 교육장으로 대중교통을 이용하여 이동한다.
3. 교육()
4. 집으로 대중교통을 이용하여 이동한다.
5. 과제를 해결한다.
6. 잠을 잔다.

000으로 000을 이용하여 이동한다.

```
이동( 장소, 탈것 ){  
    [장소](으)로 [탈것](를)을 이용하여 이동한다.  
}
```

✓ SSAFY 생의 하루 일과

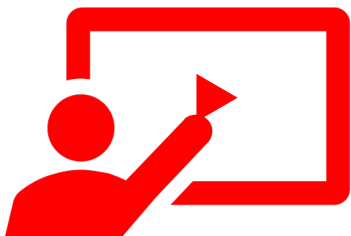
1. 아침에 일어난다.
2. 이동(강의장, 대중교통)
3. 교육()
4. 이동(집, 자동차)
5. 과제를 해결한다.
6. 잠을 잔다.

✓ SSAFY 생의 하루 일과

1. 아침에 일어난다.
2. 이동(강의장, 대중교통)
3. 과제 = 교육()
4. 이동(집, 자동차)
5. 과제를 해결한다. (사실 있을 때만 하는 것)
6. 잠을 잔다.

```
교육() {  
    오전 수업을 듣는다.  
    점심을 먹는다.  
    오후 수업을 듣는다.  
    return 과제유무(true or false)  
}
```

✓ 사람의 정보를 출력하자.



name : Yang
age : 45
hobby : 유튜브



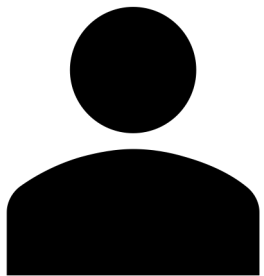
name : Hong
age : 25
hobby : 골프

나의 이름은 Yang 입니다.
나이는 45세, 취미는 유튜브 입니다.

나의 이름은 Hong 입니다.
나이는 25세, 취미는 골프 입니다.

```
info(name, age, hobby){  
    나의 이름은 name 입니다.  
    나이는 age세, 취미는 hobby 입니다.  
}
```

✓ 사람의 정보를 관리하자.



속성
name :
age :
hobby :

동작
info()

```
public class Person {  
    String name;  
    int age;  
    String hobby;  
  
    public void info() {  
        System.out.println("나의 이름은 " + name + "입니다.");  
        System.out.println("나이는 " + age + "세, 취미는 " + hobby + " 입니다.");  
    }  
}
```


✓ 함수(Function)란?

- 특정 작업을 수행하는 문장들의 모임
- 이름을 붙인 것
- 실행 가능한 단위
- 함수의 구성 요소: 반환타입(또는 void), 함수이름, 매개변수, 함수 몸체
- 자바에서는 함수가 클래스의 일부분으로서 존재(메서드: 객체의 멤버 함수)

```
public int add(int a, int b) {  
    return a + b;  
}
```

```
public void printHelloWorld() {  
    System.out.println("Hello, World!");  
}
```

- ✓ 관련 있는 변수와 함수를 묶어서 만든 사용자정의 <데이터타입>
 - member field: 멤버 변수, 객체의 속성, 상태
 - member method: 멤버 메서드, 객체의 동작, 행위(로직)
- ✓ 객체를 생성하는데 사용하는 청사진(Blueprint)
- ✓ 객체를 생성하는 틀
- ✓ 프로그래밍의 목적을 고려하여 클래스를 설계하고 객체를 생성
- ✓ 각 객체들이 어떤 특징(속성과 동작)을 가지고 있을지 결정한다.
- ✓ 클래스를 통해 생성된 객체를 인스턴스라고 한다.
- ✓ 객체들은 메서드를 통해 상호작용한다(서로 메시지를 주고 받는다).
- ✓ 데이터와 메서드를 하나로 묶어 캡슐화한다.
- ✓ 멤버 메서드에서는 멤버 변수에 대한 접근이 자유롭다(매개변수로 넘길 필요 X).

✓ 클래스의 구성 요소

- 멤버 변수(member field) - 속성(Attribute)
- 멤버 메서드(member method) - 동작(Behavior)
- 생성자(Constructor)
- 중첩 클래스(Nested Class)

✓ 클래스 선언 문법

```
[접근제어자] [final|abstract] class 클래스이름 {  
  
    // 멤버 변수, 필드 (속성 정의)  
    [접근제어자] [static] [final] 데이터타입 변수이름 [=초기값];  
  
    // 생성자  
    [접근제어자] 클래스이름([매개변수들]) {  
        생성자 본문  
    }  
  
    // 메서드 (기능 정의)  
    [접근제어자] [static] [final] 반환타입|void 메서드이름([매개변수들]) {  
        메서드 본문  
    }  
}
```

✓ 객체 생성 문법

클래스이름 객체이름 = new 클래스이름([생성자매개변수들]);

✓ 객체 멤버 접근

- . 연산자(...가 가지고 있는)를 사용

- 멤버 변수의 값 접근

객체이름.멤버변수이름

- 멤버 메서드 호출

객체이름.멤버메서드이름([매개변수들]);

✓ 변수의 종류

변수 종류	선언	생성 시기	특징	메모리 영역
클래스 변수 (Class Variable)	클래스에서 멤버 필드 선언 시 static 키워드를 사용	클래스가 메모리에 로드될 때 생성	모든 인스턴스가 공유하는 변수	메서드 영역 (Method Area)
인스턴스 변수 (Instance Variable)	클래스에서 멤버 필드 선언 시 static 키워드 없이 선언	인스턴스가 생성될 때 생성	각 인스턴스마다 별도로 생성	힙 영역 (Heap)
지역 변수 (Local Variable)	메서드, 생성자 또는 초기화 블록 내에서 선언	선언된 블록이 실행될 때 생성	블록이 끝나면 소멸	스택 영역 (Stack)

✓ 메서드 (Method)

- 객체가 할 수 있는 행동을 정의
- 어떤 작업을 수행하는 명령문의 집합에 이름을 붙여 놓은 것
- 메서드의 이름은 소문자로 시작하는 것이 관례

public / protected / (default) / private



static / final / abstract / synchronized



[접근제어자] [활용제어자] 반환타입 메서드이름([매개변수들]) {

메서드 본문(문장들..)

}

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

✓ 메서드 선언

- 선언시 {} 안에 메서드가 해야 할 일을 정의

✓ 메서드 호출

- 객체를 생성한 후 객체의 멤버 메서드를 호출한다.
- 클래스 객체.메서드 이름으로 호출

```
Person p = new Person();  
p.info();
```

- static 이 메서드에 선언되어 있을 때는 클래스이름.메서드 이름으로 호출

```
Person.hello();
```

```
public class Person {  
  
    public void info() {  
        // 메서드 내용 정의  
    }  
  
    public static void hello() {  
        // 메서드 내용 정의  
    }  
}
```


✓ 매개변수(Parameter)

- 메서드에서 사용하는 것

```
public void study(int time) {  
    //int time = ?  
    //파라미터는 해당 위치에 선언한 지역변수  
    System.out.println(time+"시간 공부.");  
}
```

✓ 인자(Argument)

- 호출하는 쪽에서 전달하는 것

```
Person p = new Person();  
p.study(10);
```

✓ 매개변수 생략 가능

✓ 파라미터 전달 시 묵시적 형 변환

```
p.study((byte) 10);    // 0  
p.study((short) 10);   // 0  
p.study(10);           // 0  
p.study(10L);          // X  
p.study(10.0f);        // X  
p.study(10.0);         // X  
p.study(10, 10);       // X
```

- ✓ 리턴 타입은 메서드를 선언할 때 지정, 없다면 void (return 문 생략 가능)
- ✓ 리턴 타입을 작성했다면 반드시 해당 타입의 값을 리턴
- ✓ 리턴 타입은 하나만 적용 가능

```
public int getAge() {  
    return age;  
}
```

```
Person p = new Person();  
p.name = "Yang";  
p.age = 45;  
p.hobby = "유튜브";
```

```
int age = p.getAge();
```

✓ 메서드 오버로딩 (Overloading)

- 이름이 같고 매개변수가 다른 메서드를 여러 개 정의하는 것
- 중복 코드에 대한 효율적 관리 가능
- 파라미터의 개수 또는 순서, 타입이 달라야 할 것 (파라미터 이름만 다른 것은 X)
- 리턴 타입이 다른 것은 의미 X

`System.out.println`

- `println()` : void - PrintStream
- `println(boolean x)` : void - PrintStream
- `println(char x)` : void - PrintStream
- `println(char[] x)` : void - PrintStream
- `println(double x)` : void - PrintStream
- `println(float x)` : void - PrintStream
- `println(int x)` : void - PrintStream
- `println(long x)` : void - PrintStream
- `println(Object x)` : void - PrintStream
- `println(String x)` : void - PrintStream

doc Declaration Console
y at this time.

생성자

✓ 생성자

- new 키워드와 함께 호출하여 객체 생성: 객체를 생성할 때 사용됨
- 클래스명과 동일: 생성자는 클래스와 동일한 이름을 가짐
- 반환 타입이 없음: 생성자는 반환 타입을 가지지 않음
- 객체가 생성될 때 반드시 하나의 생성자 호출: 객체를 생성할 때 항상 하나의 생성자가 호출됨
- 멤버 필드의 초기화: 생성자는 객체의 멤버 필드를 초기화하는 데 주로 사용됨
- 기본 생성자의 자동 제공: 클래스에 생성자가 하나도 정의되지 않으면, 컴파일러가 자동으로 매개변수가 없는 기본 생성자를 추가함
- 기본 생성자: 매개변수가 없고 내용이 없는 생성자
- 생성자 오버로딩: 매개변수의 개수나 타입이 다른 여러 개의 생성자를 정의할 수 있음
- this()를 사용한 생성자 호출: 생성자의 첫 번째 라인에서 this()를 사용하여 같은 클래스의 다른 생성자를 호출할 수 있음

- ✓ 클래스 명과 이름이 동일 (대·소문자)
- ✓ 반환타입이 없다. (void 작성 x)

```
public class Dog {  
    public Dog() {  
        System.out.println("기본 생성자!");  
        System.out.println("클래스 이름과 동일하고 반환타입 x");  
    }  
}
```

✓ 기본(디폴트) 생성자

- 클래스 내에 생성자가 하나도 정의되어 있지 않을 경우 컴파일러가 자동으로 추가하는 생성자
- 접근제어자: 클래스의 접근제어자와 동일
- 형태 : 매개변수와 본문 내용이 없는 형태

[접근제어자] 클래스명() {}

```
public class Dog {
```

```
    public Dog( ) { }
```

```
}
```

```
public class Main {
```

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

```
        // 객체 생성
```

```
        Dog d = new Dog( );
```

```
    }
```

```
}
```

✓ 파라미터가 있는 생성자

- 생성자의 목적이 필드 초기화
- 생성자 호출 시 값을 넘겨주어야 함.
- 해당 생성자를 작성하면 컴파일러가 기본 생성자를 추가하지 않음.

```
public class Dog {  
    String name;  
    int age;
```

```
    public Dog(String n, int a){  
        name = n;  
        age = a;  
    }
```

```
}
```

```
public class Main {  
    public static void main(String [] a) {  
        Dog d1 = new Dog( );  
        d1.name = “짱”;  
        d1.age = 3;
```

```
        Dog d2 = new Dog(“메리”, 4);
```

```
    }
```

```
}
```


✓ 생성자 오버로딩을 지원한다.

- 매개변수의 타입 또는 개수, 순서가 다른 것

```
class Dog {  
    Dog( ) { }  
    Dog(String name) { }  
    Dog(int age) { }  
    Dog(String name, int age) { }  
}
```

```
class Main {  
    public static void main(String [] a) {  
        Dog d = new Dog( );  
        Dog d2 = new Dog("쫘");  
        Dog d3 = new Dog(3);  
        Dog d4 = new Dog("메리", 4);  
    }  
}
```

✓ this

- 참조 변수로써 현재 인스턴스 자기 자신을 가리킴(참조)
- this와 .연산자를 이용하여 자신의 멤버 접근 가능
- 지역변수(매개변수)와 필드의 이름이 동일할 경우 필드임을 식별할 수 있게 함(생성자에서 주로 활용)
- 인스턴스에 대한 참조이므로 static 메서드 또는 static 블록에서 this 사용 불가
- 메서드 체이닝(Method Chaining): 메서드에서 this를 반환하여 메서드 호출을 연쇄적으로 이어갈 수 있음
- this([생성자매개변수들])를 호출하여 생성자 안에서 같은 클래스 내의 다른 생성자를 호출
 - this()를 이용한 생성자 호출은 반드시 생성자 내에서만 가능
 - this() 구문을 생성자 안에서 사용할 때는 반드시 첫번째 줄에 위치

다음 방송에서 만나요!

삼성 청년 SW 아카데미