JAVA 프로그래밍

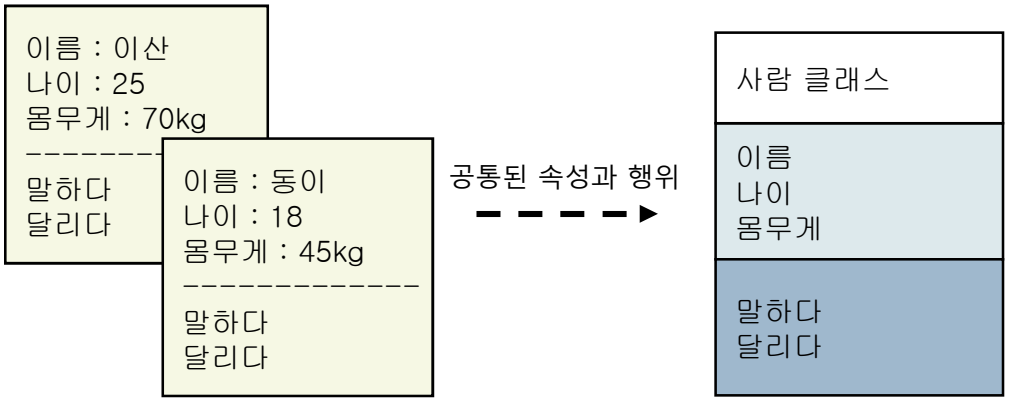
Chapter 08 객체와 클래스

목차

- 객체와 클래스
- 객체의 생성과 사용
- 클래스 선언의 기초문법
- 클래스의 정적 구성요소

객체와 클래스

- ❖ 객체 지향 프로그래밍(OOP, Object Oriented Programming)
  - 객체 지향 프로그밍에서는 모든 데이터를 객체(Object)로 취급하며, 이러한 객체가 프로그램의 중심이 된다.
- ❖ 객체
  - 객체란 어떤 대상이 상태(state)와 행위(behavior)를 갖는 것.
  - 소프트웨어 객체는 상태를 나타내기 위해 변수(variable)를 사용하며, 행위들은 메서드(method)로 표현된다.
- ❖ 클래스
  - 클래스는 상태와 행위가 같은 객체들을 대표하는 것.
  - 객체의 상태를 나타내는 필드(field)와 객체의 행위을 나타내는 메서드(method)로 구성된다.
  - 개념적(추상적)이며 종합적이다.
  - 객체를 정의하는 모체(틀).
  - 객체를 통해서만 구체화된다.
  - 객체의 종류를 나타내는 데이터 타입.



객체의 생성과 사용

❖ 클래스 선언

- [접근제어자] [지정예약어] class 클래스명 [extends 클래스] [implements 인터페이스]

```
public abstract class Child extends Parent implements Serializable{

}
```

❖ 객체 생성

- 자바의 new 연산자를 사용하여 생성한다.
- 클래스명 객체명 = new 생성자(매개변수);
- 예) Person p = new Person("홍길동");

❖ 멤버 참조

- 멤버 참조는 '.' 연산자를 이용한다
- 예) p.name; , p.run();

클래스 선언의 기초문법

```
public class SampleClass {

  public int x;  // 멤버 필드(변수)
  public int y;  // 멤버 필드(변수)

  public int plus() {     // 멤버 메서드
    return x + y;
  }
  public int minus(){     // 멤버 메서드
    return x - y;
  }
  public int divide(){    // 멤버 메서드
    return x / y;
  }
  public int mul(){       // 멤버 메서드
    return x * y;
  }
}
```

```
// 객체 생성
SampleClass obj = new SampleClass();

// 멤버 변수에 값할당
obj.x = 5;
obj.y = 10;

// 멤버 메소드 사용
System.out.println(obj.minus());
System.out.println(obj.plus());
System.out.println(obj.divide());
System.out.println(obj.mul());
```

클래스의 구성요소

- 클래스는 멤버(member)로 속성을 표현하는 필드(field)와 기능을 표현하는 메서드(method)로 구성된다.
- 또한 클래스는 생성된 객체의 필드를 초기화하는 특별한 메서드인 생성자를 가진다.
- 필드(Member Field) 또는 변수(Member Variable)
- 메소드(Member Method)
- 생성자(Constructor)

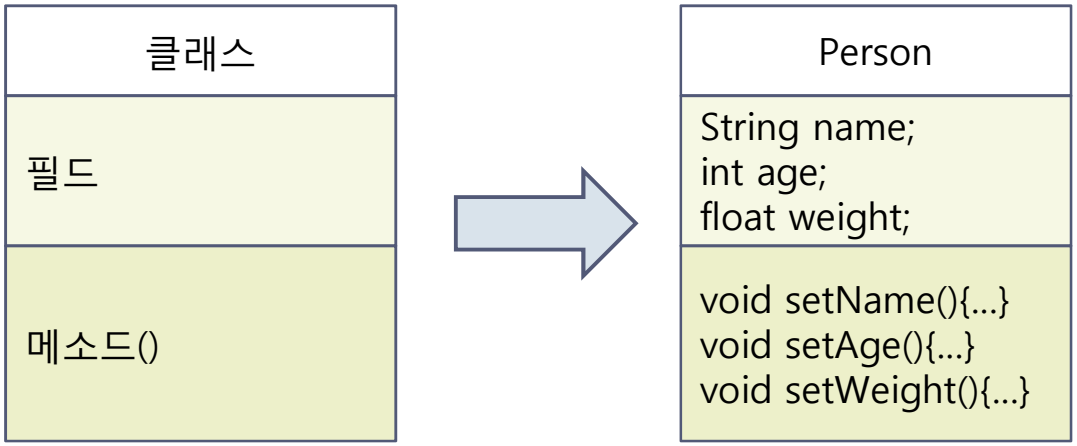


그림. 클래스의 구성요소

생성자

❖ 생성자(Constructor)

- 객체의 생성과 동시에 자동으로 호출되는 메서드.
- 객체 생성시 변수의 초기화 작업을 하는 메서드.
- 생성자의 이름은 클래스 이름과 같아야 하며, 리턴 값을 갖지 않는다.(void형으로 선언하지 않는다.)
- Obj = new Person("홍길동", 25);
- 하나의 클래스가 여러 개의 생성자를 가질 수 있다. (중복 정의 가능).

❖ this

- 객체 내부에서 객체 자신을 참조하는 변수.
- 지역변수와 멤버변수를 구별할 때 사용하는 변수.

❖ this()

- 현재 객체의 생성자를 의미하는 것이다.
- 반드시 생성자의 첫 행에 정의해야 하며, 그렇지 않을 시 Compile 오류가 발생한다.
- this()를 이용 한 클래스내의 특정 생성자에서 Overloading되어 있는 다른 생성자를 호출할 수 있다.
- 그렇게 함으로 해서 생성자에서 코딩 내용이 중복되는 부분을 막을 수 있다.

필드(Field)

❖ 필드(Field)
- 객체의 특징적인 속성을 정의한 것으로, 클래스에 포함된 변수를 의미한다.
- 변수와 상수 즉, 데이터를 표현
- 필드의 형태가 클래스(static field) 와 인스턴스(instance)로 필드 개념이 달라진다.

❖ 캡슐화(encapsulation)
- 캡슐화란 데이터(속성)와 데이터를 처리하는 함수를 하나로 묶는 것.
- 외부에서 객체 내부를 직접 접근하거나 조작할 수 없고, 외부에서 접근할 수 있도록 정의된 오퍼레이션을 통해서만 관련 데이터를 접근할 수 있다.

❖ 정보은닉을 위한 private 접근 제한자
- 정보은닉(Information hiding) : 객체의 구성 요소를 외부로부터 감추는 기술
- 객체에 포함된 정보의 손상과 오용을 막을 수 있다.

❖ 상수 필드의 정의 : final
- final int a = 100;

필드의 구분

- ❖ 인스턴스 변수(필드)
  - 각 인스턴스의 개별적인 저장공간. 인스턴스마다 다른 값 저장가능
  - 인스턴스 생성 후, '객체변수.인스턴스변수명'으로 접근
  - 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지 컬렉터에 의해 자동 제거됨
  - 인스턴스(instance) : 어떤 클래스로 부터 만들어진 객체를 그 클래스의 인스턴스라고 한다.

- ❖ 클래스 변수(필드)(static field)
  - 같은 클래스의 모든 인스턴스들이 공유하는 변수
  - 인스턴스 생성없이 '클래스이름.클래스변수명'으로 접근
  - 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸

- ❖ 지역 변수
  - 메서드 내에 선언되며, 메서드의 종료와 함께 소멸
  - 조건문, 반복문의 블럭{} 내에 선언된 지역변수는 블럭을 벗어나면 소멸

메소드(Method)

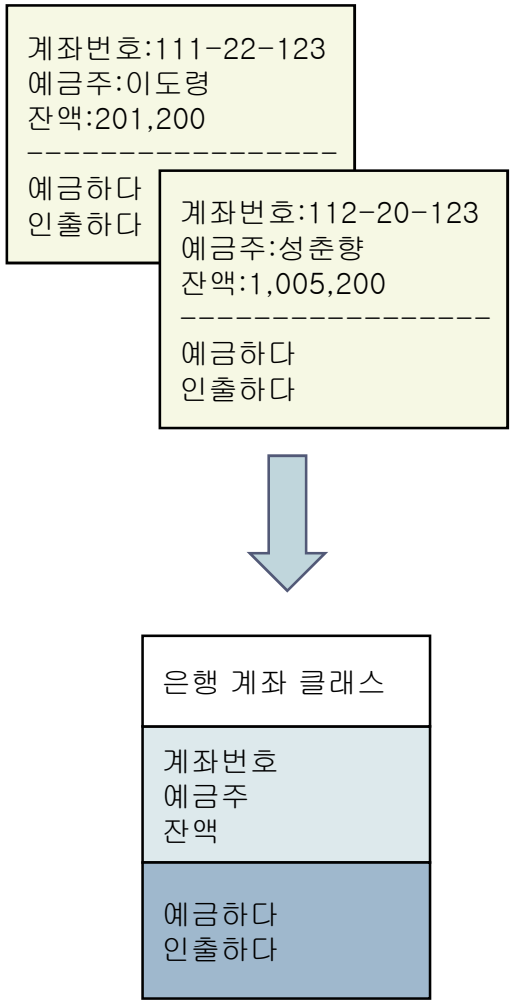
❖ 메소드(Method)

- 메소드는 특정한 일을 수행하는 행위, 다시 말해 동작을 의미하는 것이다.
- 클래스의 기능을 정의한다.
- 타 언어의 함수와 같다.
- 반복적으로 수행되어야 하는 여러 문장을 하나의 메소드로 정의해 놓으면 좋다.
- 관련된 여러 문장을 하나의 메소드로 만들어 놓는 것이 좋다.
- return 문을 만나면 실행을 종료한다.

❖ 매개변수 전달기법

- Call by Value : 기본형 매개변수이며 값을 전달한다.
- Call by Reference : 참조형 매개변수를 전달하며 데이터의 참조값(주소)를 전달한다.

메소드 예제 클래스




```
class Account{
  String accountNo;   // 계좌번호
  String ownerName;   // 예금주 이름
  int balance;        // 잔액

  // 생성자 선언
  Account(String accountNo, String ownerName,
int balance){
    this.accountNo = accountNo;
    this.ownerName = ownerName;
    this.balance = balance;
  }
  // 입금
  void deposit(int amount){
    balance += amount;
  }
  // 출금
  int withdraw(int amount){
    if(balance < amount)
      return 0;
    balance -= amount;
    return amount;
  }
}
```

메소드 오버로딩

❖ 메소드 오버로딩(Overloading : 중복정의)

- 한 클래스 내에 같은 이름의 메소드를 중복하여 정의하는 것.
- 메소드의 이름은 같으나 패러미터 변수의 수와 타입이 다르면 중복 정의가 가능하다
- 메소드 오버로딩은 반환 타입(return type)과는 관계가 없다.

예시)

- void println()
- void println(boolean x)
- void println(char x)
- void println(int x)
- void println(String x)
- void println(Object x)

클래스 정적 구성요소

- 필드와 메소드를 클래스 자체에 속하는 구성 요소로 선언 시 필요
- 이런 구성요소를 클래스의 정적(static)구성요소라 한다.
  - 정적 필드(static field)
  - 정적 초기화 블럭
  - 정적 상수(static final)
  - 정적 메소드(static method)

클래스 정적 구성요소

❖ 정적 필드 (static field : 클래스 필드, 클래스 변수)

- 클래스 자체에 속하는 데이터를 저장할 변수 선언을 목적으로 만든다.
- 필드의 선언문 앞에 static 키워드를 사용하여 선언.
- 인스턴스 필드는 객체를 생성한 후에 사용할 수 있으나 클래스 필드는 객체의 생성 없이도 사용이 가능하다.
- 인스턴스 필드는 객체마다 생성되지만 클래스 필드는 단 하나만 생성된다.

```
class Accumulator{
  int total = 0;                  // 인스턴스 변수
  static int grandTotal = 0;      // 클래스 변수

  void accumulate(int amount){
    total += amount;
    grandTotal += amount;
  }
}
```

클래스 정적 구성요소

❖ 정적 초기화 블럭

- 정적 필드의 초기화를 위해 사용한다.
- 정적 필드는 생성자를 통해 초기값을 대입 할 수 없다.
- 생성자는 객체의 생성시에 호출되기 때문이다.

```
class StaticBlock{
    static int a;
    static int b;
    static float c;

    static{
            a = 100;
            b = 200;
            c = 3.14f;
    }
}
```

클래스 정적 구성요소

❖ 정적 메소드(클래스 메소드)

- static 키워드를 붙여서 선언한 메소드.
- 정적 필드와 같은 특성을 갖는다.
- 객체의 생성 없이 클래스 이름으로 바로 접근이 가능하다.
- 특정 객체에 종속되지 않고 기능을 수행하는 기능적 메소드에 유용.
- 클래스 메소드에서 인스턴스 변수나 인스턴스 메소드를 사용할 수 없다.

```
class Accumulator{
  int total = 0;                  // 인스턴스 변수
  static int grandTotal = 0;      // 클래스 변수

  void accumulate(int amount){
    total += amount;
    grandTotal += amount;
  }

  static int getGrandTotal(){     // 클래스 메소드
     return grandTotal;
  }
}
```

클래스 정적 구성요소

❖ 정적 상수(static final)

- 정적 필드에 final 키워드를 붙이면 프로그램 수행 중에 값을 변경 할 수 없는 상수변수가 된다.

❖ 형태

- static final int MAX_VALUE = 100;
- final static int MIN_VALUE = 10;

```
class LimitedValue {

  final static int UPPER_LIMIT = 100000;

  int value;
  void setValue(int value) {
    if (value < UPPER_LIMIT)
      this.value = value;
    else
      this.value = UPPER_LIMIT;
  }
}
```

클래스의 데이터 타입으로써의 의미

- 변수
  - 하나의 데이터를 저장할 수 있는 공간.
- 배열
  - 같은 종류의 여러 데이터를 하나의 집합으로 저장할 수 있는 공간.
- 구조체
  - 서로 관련된 여러 데이터를 종류에 관계없이 하나의 집합으로 저장할 수 있는 공간.
- 클래스
  - 데이터와 함수의 결합(구조체 + 함수)

클래스의 데이터 타입으로써의 의미

- ❖ 기본 데이터 타입의 한계
  - 기본 데이터 타입으로 변수를 생성했을 때 생성된 변수에는 하나의 데이터만 보관할 수 있다.
  - 정해진 데이터 타입만을 사용해야 한다.
- ❖ 구조체
  - 동시에 여러 개의 데이터를 담을 수 있는 데이터 타입을 사용자가 직접 만들어서 사용하는 사용자 정의 데이터 타입
- ❖ 클래스
  - 자바에서는 구조체가 발전해서 클래스가 된다.