yonggyo1981 객체지향프로그래밍1 동영상 강의 URL 업데이트 3536ecb · 2 years ago History

Name	Name	Last commit date
..		
day05_07	객체지향 프로그래밍1	2 years ago
images	객체지향 프로그래밍1	2 years ago
README.md	객체지향프로그래밍1 동영상 ...	2 years ago

README.md

강의 동영상 링크

동영상 링크

객체지향 프로그래밍1

객체 지향 프로그래밍과 클래스

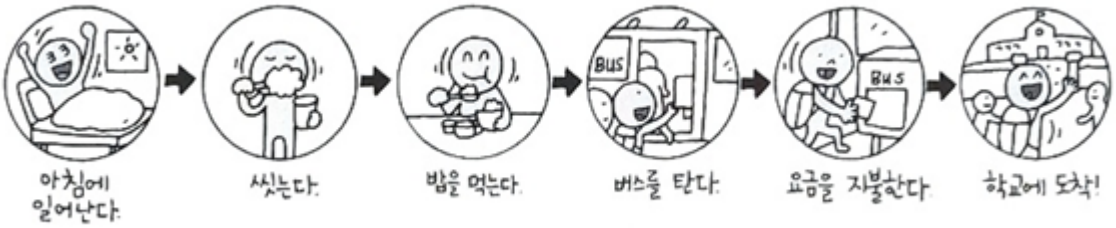
객체와 객체 지향 프로그래밍

- 국어사전에서 객체의 뜻을 찾아보면 '의사나 행위가 미치는 대상'이라고 설명합니다.
- 우리 주위에 있는 객체를 생각해 보면 사람, 자동차, 건물 등이 있습니다. 즉, 눈에 보이는 사물은 모두 객체라고 할 수 있습니다.
- 눈에 안보이는 것도 객체가 될 수 있습니다. 주문, 생산, 관리등 어떤 행동을 나타내는 단어도 객체가 될 수 있습니다.

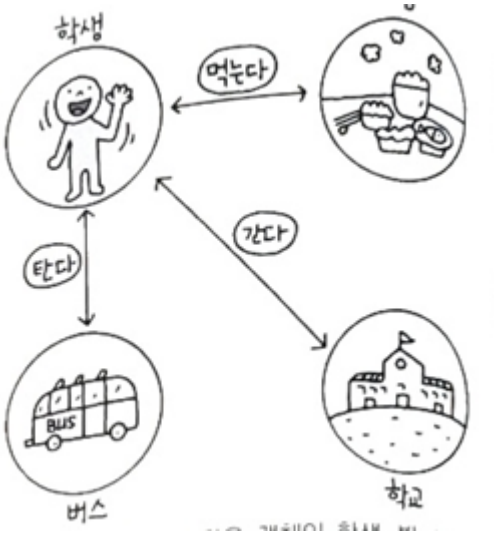
- 자바 객체 지향 프로그래밍(Object-Oriented Programming: OOP)은 객체를 기반으로 하는 프로그램입니다.
- 객체지향 프로그램이란 '**어떤 대상(객체)**'을 가지고 프로그래밍한다는 것

생활 속에서 객체 찾아보기

- 절차 지향 프로그래밍 : 순서대로 일어나는 일을 시간순으로 프로그래밍 하는 것



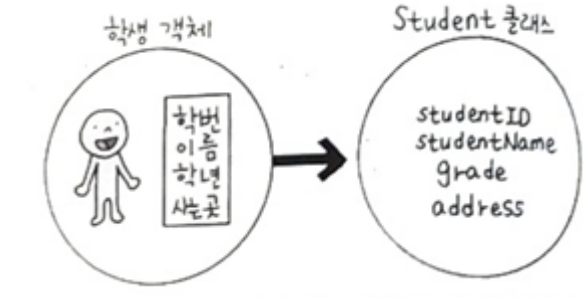
- 객체 지향 프로그래밍 : 객체를 정의하고 객체 간 협력을 프로그래밍 하는 것
- 객체지향 프로그램은 먼저 객체를 만들고 객체 사이에 일어나는 일을 구현하는 것
- 객체지향 프로그래밍을 할 때는 객체를 먼저 정의하고 각 객체가 어떤 기능을 제공하고 객체 간 협력을 어떻게 구현할 것인지를 고민해야 한다.



> 객체지향 프로그래밍은 객체인 학생, 밥, 버스, 학교 사이에 일어나는 일을 프로그램으로 구현하는 것입니다.

클래스 살펴보기

- 객체지향프로그램은 클래스를 기반으로 프로그래밍합니다.
- 클래스는 객체의 속성과 기능을 코드로 구현한 것입니다.
- 객체를 클래스로 구현한다는 것을 '**클래스를 정의한다**'라고 합니다.
- 클래스를 정의하려면 우선 클래스의 이름과 클래스가 가지는 속성 또는 특성이 필요합니다.



- 학생 객체를 생각해 보면, 먼저 객체를 표현할 클래스의 '이름'이 필요합니다. 클래스명을 Student라고 정합니다.
- 학생 객체가 가지는 일반적인 속성은 학번, 이름, 학년, 사는 곳 등으로 생각해볼 수 있습니다.
- 이런 클래스의 속성은 특성이라고 하고 클래스 내부에 변수로 선언합니다.
- 이렇게 선언하는 클래스의 속성을 **멤버변수**라고 합니다.

클래스를 정의하는 문법

```
(접근제어자) class 클래스 이름 {
        멤버변수;
        메서드;
}
```

> 클래스 이름 앞에 있는 public 예약어는 **접근 제어자**라고 합니다. 정보은닉 참고

day05_07/Student.java

```
package day05_07;

public class Student {
        int studentID; // 학번
        String studentName; // 학생이름
        int grade; // 학년
        String address; // 사는 곳
}
```

- 클래스를 만들때는 class 예약어를 사용합니다.
- 클래스 이름은 Student라고 정했습니다.
- {}안에는 클래스의 내용을 구현합니다.
- 프로그램에서 사용할 객체의 속성을 클래스의 변수로 선언합니다.
- 변수를 선언할 때는 각 변수의 속성에 맞는 자료형을 사용해야 합니다.
- 자바 프로그램은 모든 요소가 클래스 내부에 있어야 합니다.
- 클래스 외부에는 package 선언과 import문장 외에 아무것도 선언하지 않습니다.

클래스 이름을 짓는 규칙

- 클래스 이름은 대문자로 시작합니다.(파스칼 케이스 - 첫 시작 단어는 대문자로 예) OrderInfo, OrderItems)

- 소문자로 시작한다고 해서 오류가 발생하는 것은 아니나 개발자들 사이에서 관습처럼 사용되는 규칙입니다.
- 관습을 따르면 각 명칭의 용도를 쉽게 유추할 수 있습니다.

클래스의 속성을 구현하는 멤버변수

- 클래스에 선언하여 객체 속성을 나타내는 변수가 멤버 변수(member variable)입니다.

```
public class Student {
        int studentID; // 학번
        String studentName; // 학생이름
        int grade; // 학년
        String address; // 사는 곳
}
```

- 멤버변수 : int studentID, String studentName, int grade, String address
- 클래스에 선언하는 멤버변수는 다른말로 **속성(property)**, **특성(attribute)** 등으로 표현하기도 합니다.
- 멤버변수는 속성이 무엇이냐에 따라 알맞는 자료형을 선언해 주어야 합니다.

Person 클래스 만들기

day05_07/Person.java

```
package day05_07;

public class Person {
        String name; // 이름
        int height; // 키
        double weight; // 몸무게
        char gender; // 성별
        boolean married; // 결혼여부
}
```

- Person 클래스의 멤버 변수로 이름, 키, 몸무게, 성별, 결혼여부를 선언했습니다.
- 이와 같은 멤버변수를 선언할 때 int, double형 같은 **기본자료형**(primitive data type)으로 선언할 수도 있고 또 다른 클래스형으로 선언할 수도 있습니다.
- 클래스형이란 다른 말로 **참조자료형**이라고 합니다.
- 참조자료형으로 사용하는 클래스는 String, Date와 같이 이미 JDK에서 제공하는 것일 수도 있고, 개발자가 직접 만든 Student나 Person같은 클래스가 다른 클래스에서 사용하는 멤버변수의 자료형이 될 수 있습니다.

변수의 자료형

기본자료형	참조자료형
int, long, float, double등	String, Date, Student 등

클래스와 인스턴스

- 클래스에서는 학생 객체가 가지고 있는 속성르 사용해 학생과 관련된 기능을 구현할 수 있습니다.
  (예 : '학생에게 이름을 부여한다.', '학생이 사는 곳을 출력한다.')
- 클래스 내부에서 멤버 변수를 사용하여 클래스의 기능을 구현한 것을 '**멤버 함수**(member function)' 또는 '**메서드**(method)'라고 합니다.
  (메서드로 용어를 통일하여 진행)

학생 이름과 주소를 출력하는 메서드 만들기

day05_07/Student .java

```
package day05_07;

public class Student {
	int studentID; // 학번
	String studentName; // 학생이름
	int grade; // 학년
	String address; // 사는 곳

	public void showStudentInfo() {
		System.out.println(studentName + "," + address); // 이름, 주소 출력
	}
}
```

패키지란?

- 클래스파일의 묶음
- 패키지를 만들면 프로젝트 하위에 물리적으로 디렉토리가 생성됩니다.
- 패키지는 계층구조를 가지고 있습니다.
- 패키지가 단순히 클래스 묶음이 아닌 프로젝트 전체 소스 코드를 구성하는 계층구조가 되고, 이 계층구조를 잘 구성해야 소스 코드 관리와 유지보수가 편리합니다.

패키지 선언하기

- 자바 소스 코드에서 클래스의 패키지 선언은 다음처음 맨 위에서 합니다.

```
package domain.student.view

public class StudentView {
	...
}
```

- 클래스 이름은 StudentView이지만, 클래스의 전체 이름(class full name)은 domain.student.view.StudentView입니다.

- 클래스 이름이 같다고 해도 패키지 이름이 다르면 클래스 전체 이름이 다른 것이므로 다른 클래스가 됩니다.
- 같은 이름의 클래스라도 다른 패키지에 속해 있으면 서로 연관이 없습니다.

메서드

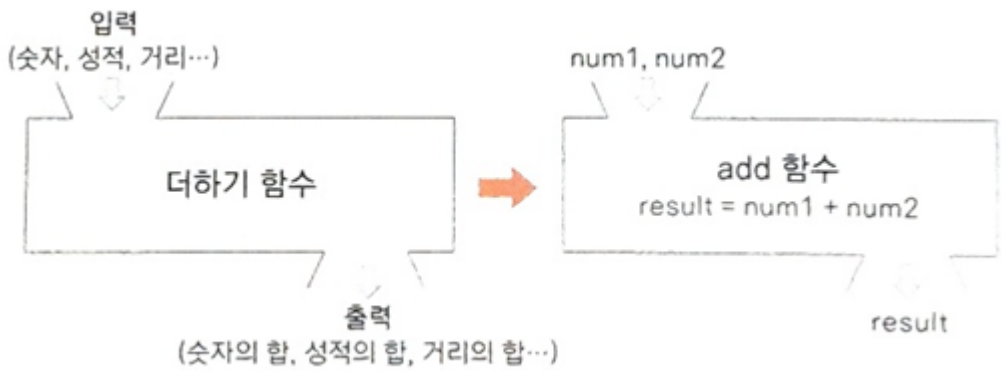
메서드는 함수(function)의 한 종류 입니다.

함수란?

- 하나의 기능을 수행하는 일련의 코드
- 함수는 어떤 기능을 수행하도록 미리 구현해 놓고 필요할 때마다 호출하여 사용할 수 있습니다.

함수의 입력과 반환

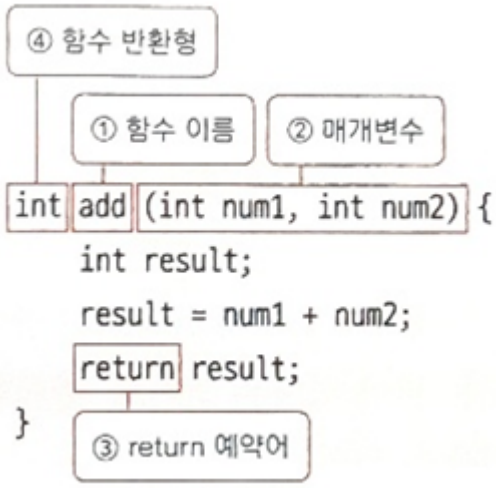
- 함수는 이름이 있고, 입력된 값과 결과 값을 갖습니다.
- 두 수를 더하는 함수를 예로 들면 두 수를 입력받아서 '더하기 함수'를 거치면 두 수의 합을 반환합니다.



- 함수에 이름을 붙일 때는 의미를 알 수 있는 단어를 사용하는 것이 좋습니다. 더하는 함수 이므로 **add**라고 이름을 짓습니다.
- 더하는 두 수는 각각 num1, num2라고 정합니다.
- num1, num2와 같이 함수의 입력을 받는 변수를 **매개변수**라고 합니다.
- 두 수를 더한 결과 값을 result 변수에 저장하여 돌려 줍니다. 이를 '**결과를 반환한다**'고 합니다.
- 이렇게 함수를 수행한 후, 결과로 되돌려 주는 값인 result를 **반환값**이라고 부릅니다.

함수 정의하기

함수가 하는 일을 코드로 구현하는 것을 **함수를 정의 한다** 라고 합니다.



함수이름

- 함수 이름은 add라고 썼습니다.
- (1) 위치가 함수 이름을 적는 부분입니다.
- 함수 이름은 변수 이름처럼 프로그래머가 임의로 만들면 되는데, 함수 기능과 관련있게 만들어야 나중에 호출하거나 이해하기 좋습니다.

매개변수

- add 함수는 두 값을 더하는 일을 합니다.
- 덧셈을 수행하기 위해서는 먼저 함수에 두 값이 입력되어야 합니다.
- 함수는 넘겨받은 값으로 덧셈을 수행합니다. 이렇게 함수 내부에서 사용할 괄호 안의 변수를 **매개변수**라고 합니다.
- (2) num1과 num2가 매개변수 입니다.
- 경우에 따라서는 **매개변수**가 필요없는 함수도 있습니다.

```
int getTenTotal() {
        int i;
        int total = 0;
        for(i = 1; i <= 10; i++) {
                total += i;
        }

        return total;  // 1부터 10까지 더한 값을 반환
}
```

return 예약어와 반환형

- add()함수를 수행한 후 결과 값은 변수 result에 저장됩니다.
- result에 저장된 결과 값은 함수를 호출했을 때 반환되는 값이므로 **반환값**이라고 부릅니다.
- '**이 함수의 결과 값을 반환 합니다.**를 뜻하는 예약어가 **return**입니다.

- (3) return 예약어를 사용하여 result 값을 반환하는 것 입니다.
- 반환 값의 자료형을 반환형이라고 하는데 (4) 위치에 써 줍니다.
- 이 함수에서 변수 **result**의 반환형은 정수형이므로 (4)위치에 **int**라고 적습니다.
- 경우레 따라서는 **반환값이 없는 함수**도 있습니다.
- 반환값이 없다고 해서 반환형을 쓰는 (4)위치를 비워두면 오류가 발생합니다. 이 때에는 (4)위치에 **void**라고 씁니다.
- **void**는 비어있다는 의미로 '**반환할 값이 없다**'는 뜻의 예약어입니다.

```
void printGreeting(String name) {
        System.out.println(name + "님 안녕하세요");
        return; // 반환값 없음 (생략 가능)
}
```

- return 예약어는 함수 수향을 끝내고 프로그램 흐름 중에서 호출한 곳으로 다시 되돌아갈 때도 사용할 수 있습니다.

```
 void divide(int num1, int num2) {
        if (num2 == 0) {
                System.out.println("나누는 수는 0이 될 수 없습니다.");
                return; // 함수 수행 종료
        } else {
                int result = num1 / num2;
                System.out.println(num1 + "/" + num2 + "=" + result + "입니다.");
        }
 }
```

- 나누는 수가 0이라면 함수 수행을 하면 안되므로 함수 수향을 종료하는 예약어 return을 사용합니다.
- **함수 수행을 종료하는 목적**이므로 return 뒤에 반환값을 적지 않아도 됩니다.

함수 호출하고 값 반환하기

함수를 사용하는 것을 **함수를 호출한다**라고 합니다.

day05_07/FunctionTest.java

```
package day05_07;

public class FunctionTest {
        public static void main(String[] args) {
                int num1 = 10;
                int num2 = 20;

                int sum = add(num1, num2); // add() 함수 호출
                System.out.println(num1 + " + " + num2 + " = " + sum + "입니다.");
        }
```

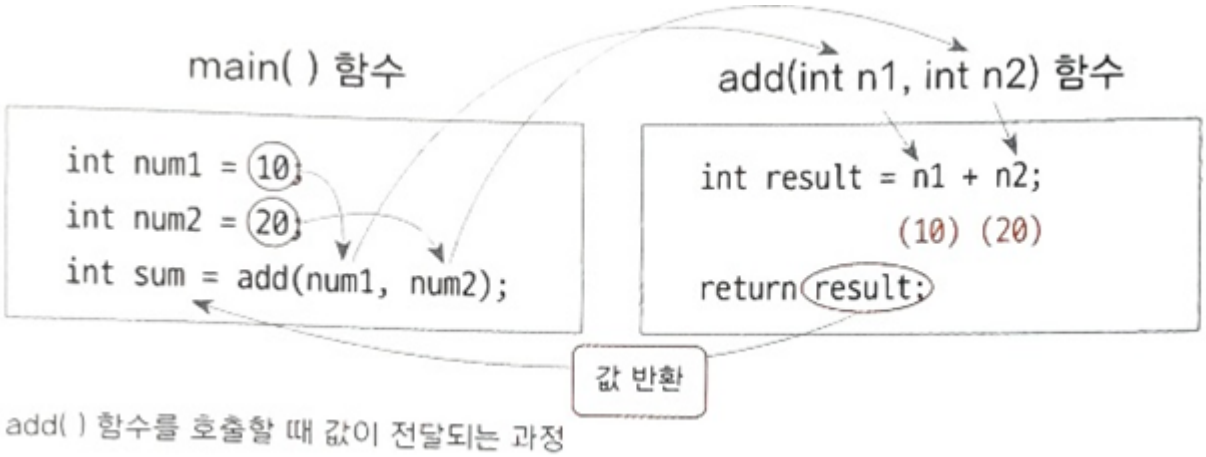
```
        // add() 함수
        public static int add(int n1, int n2) {
                int result = n1 + n2;
                return result; // 결과 값 반환
        }
}
```

실행결과

```
10 + 20 = 30입니다.
```

매개변수 살펴보기

- add() 함수를 호출할 때 num1, num2 두개의 변수를 넘겼습니다.
- 함수를 구현하는 부분에서는 add(int n1, int n2)와 같이 n1, n2를 사용했습니다.
- num1과 n1, num2와 n2는 전혀 상관이 없습니다.



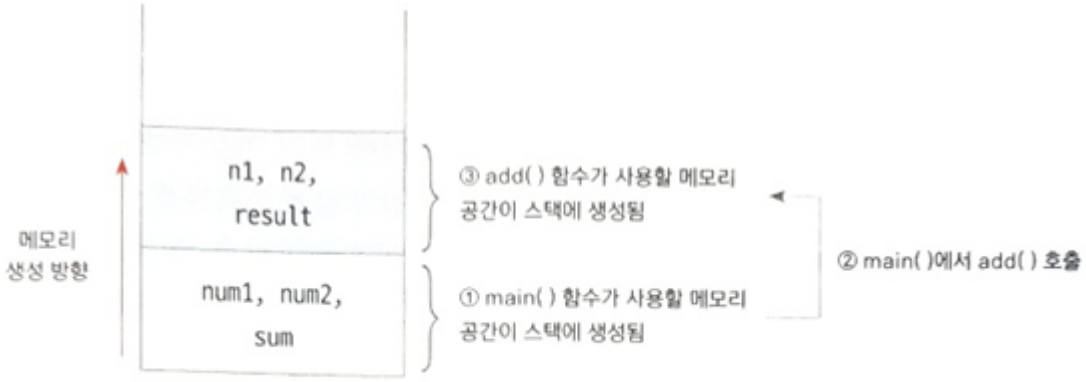
add( ) 함수를 호출할 때 값이 전달되는 과정

- num1, num2는 main()함수의 변수입니다. add(num1, num2)로 사용하면 add()함수에 두 값을 넘겨줄 수 있습니다.
- 매개변수 n1, n2는 실제 함수로 넘어온 두 값을 받아 주는 역할입니다. 다시 말해 n1, n2는 add() 함수에서 선언한 새로운 변수입니다.
- 따라서 함수를 호출할 때 사용하는 변수이름과 호출되는 함수에서 사용하는 변수는 서로 다른 변수이므로 이름이 같아도 되고 달라도 상관이 없습니다.

함수 호출과 스택 메모리

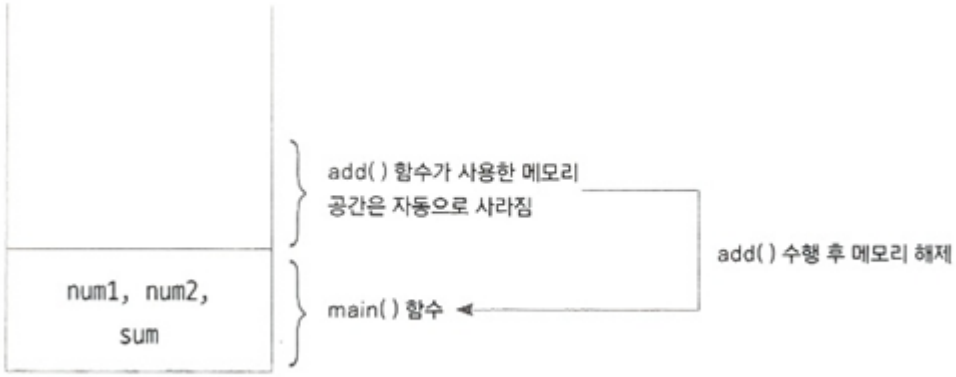
- 함수를 호출하면 그 함수만을 위한 메모리 공간이 할당되는데, 이 메모리 공간을 스택(stack)이라고 부릅니다.

add()함수를 호출하면 메모리를 생성하는 과정



- 함수가 호출되면 그 함수가 사용할 메모리 공간이 스택에 생성됩니다.
- 프로그램을 시작할 때 main() 함수부터 호출하기 때문에 가장 먼저 main()함수에 포함된 변수 num1, num2, sum을 저장할 메모리 공간이 생성됩니다.
- main()함수에서 add()함수를 호출하면 add() 함수를 저장할 메모리 공간이 스택에 새롭게 생성됩니다.

add()함수 수행이 끝나고 함수에 할당했던 메모리 공간을 해제하는 과정



- add() 함수 수행이 끝나고 결과 값이 반환되면 add()함수가 사용하던 메모리 공간은 자동으로 사라집니다.
- 프로그램에서 여러 함수를 사용하는 경우에 함수를 호출한 순서대로 메모리 공간에서 만들어지고 맨 마지막에 호출한 함수부터 반환됩니다.
- 따라서 메모리 공간도 맨 마지막에 추가된 것부터 스택에서 사라집니다.
- 즉 A() -> B() -> C() 함수의 순서로 호출했다면 C() -> B() -> A() 순서로 반환되고 스택 메모리도 이 순서대로 소멸됩니다.
- 그림에서 보듯 main() 함수에서 사용하는 num1, num2 변수의 add()함수에서 사용하는 n1, n2 변수는 서로 다른 메모리 공간을 사용하므로 이름이 같든 다르든 상관이 없습니다.
- 이렇게 함수 내부에서만 사용하는 변수를 **지역변수**라고 합니다.
- **지역변수는 스택메모리에서 생성**됩니다. 변수의 유효범위 참조

함수의 장점

- 기능을 나누어 코드를 효율적으로 구현할 수 있다.
- 기능을 분리해서 구현하면 프로그램 코드의 가독성이 좋아진다.
- 기능별로 함수를 구현해 놓고 같은 기능을 매번 코드로 만들지 않고 그 기능의 함수를 편리하게 호출할 수 있다.(중복되는 코드를 막을 수 있다.)
- 디버깅 작업을 할 때도 편리하다.(하나의 기능이 하나의 함수로 구현되어 있기 때문에 오류난 기능만 찾아서 수정하면 된다.)

클래스 기능을 구현하는 메서드

- 메서드(method)는 멤버변수를 사용하여 클래스의 기능을 구현한다.
- 즉, 메서드는 함수에 객체 지향개념이 포함된 용어로 이해하면 된다.
- 함수의 스택 메모리 사용방법과 함수의 장점등도 모두 메서드에 동일하게 적용된다.

day05_07/Student.java

```
package day05_07;

public class Student {
	int studentID; // 학번
	String studentName; // 학생이름
	int grade; // 학년
	String address; // 사는 곳

	// 학생의 이름을 반환하는 메서드
	public String getStudentName() {
		return studentName;
	}

	// 학생의 이름을 부여하는 메서드
	public void setStudentName(String name) {
		studentName = name;
	}

	public void showStudentInfo() {
		System.out.println(studentName + "," + address); // 이름, 주소 출력
	}
}
```

자바의 이름 짓기 규약

- **클래스 이름** : 파스칼케이스 - 단어의 시작 문자는 모두 대문자, 예) OrderInfo, OrderItems
- **메서드 및 멤버변수** : 카멜케이스 - 첫 단어를 제외 단어의 시작문자는 대문자, 예) showStudentInfo
- **패키지 이름** : 모두 소문자
- **상수명** : 모두 대문자, 단어와 단어 사이는 언더바(_)로 구분, 예) NO_OF_STUDENT

클래스와 인스턴스

클래스 사용과 main(), 함수

```
public class Student {
	int studentID;
	String studentName;
	int grade;
	String address;

	public String getStudentName() {
		return studentName;
	}

	public void setStudentName(String name) {
		studentName = name;
	}
}
```

- 멤버변수는 클래스 속성을 나타냅니다.
- 메서드는 멤버변수를 이용하여 클래스 기능을 구현합니다.

프로그램을 시작하는 main()함수

- main() 함수는 자바 가상 머신(Java Virtual Machine:JVM)이 프로그램을 시작하기 위해 호출하는 함수입니다.
- 클래스 내부에 만들지만, 클래스의 메서드드는 아닙니다.

main()함수에서 클래스를 사용하는 방법

- 클래스 내부에 main함수를 만드는 것

```
package day05_07;

public class Student {
	int studentID;
	String studentName;
	int grade;
	String address;


	public String getStudentName() {
		return studentName;
	}

	public void setStudentName(String name) {
		studentName = name;
	}

	// main() 함수
	public static void main(String[] args) {
		Student studentAhn = new Student(); // Student 클래스 생성
```

```
                studentAhn.studentName = "안연수";

                System.out.println(studentAhn.studentName);
                System.out.println(studentAhn.getStudentName());
        }
}
```

- 외부에 테스트용 클래스를 만들어 사용하는 것

```
package day05_07;

public class StudentTest {
        public static void main(String[] args) {
                Student studentAhn = new Student(); // Student 클래스 생성
                studentAhn.studentName = "안승연";

                System.out.println(studentAhn.studentName);
                System.out.println(studentAhn.getStudentName());
        }
}
```

- 클래스 내부에 main()함수를 만들면 이 클래스가 프로그램의 시작 클래스가 됩니다.

> 만약 패키지가 다르다면 import문을 사용해서 함께 사용하기를 원하는 클래스를 불러와야 한다. **클래스 이름이 같아도 패키지가 다르면 다른클래스이다**.
> Student라는 같은 이름의 두 클래스를 하나의 패키지에 구현하면 같은 이름의 클래스가 존재한다고 오류가 날 것입니다. 하지만 패키지가 다르면 문제가 되지 않습니다. 왜냐하면 aaa패키지 하위의 Student 클래스의 실제 이름은 aaa.Student이고, bbb패키지 하위의 Student 클래스의 실제 이름은 bbb.Student이기 때문입니다. 이를 클래스 전체 이름(class full name)이라고 합니다. 따라서 패키지가 다르면 클래스 이름이 같아도 다른 클래스입니다.

new 예약어로 클래스 생성하기

- 클래스를 사용하려면 먼저 클래스를 생성해야 합니다.

```
클래스형 변수이름 = new 생성자;
```

생성자 참조

- 클래스를 생성할 때는 new 예약어를 사용하고 이어서 생성자를 써 줍니다.
- 클래스 자료형 변수에 변수를 선언하고 new 예약어로 생성자를 호출하여 대입하면 새로운 클래스가 생성됩니다.
- 클래스가 생성된다는 것은 클래스를 실제 사용할 수 있도록 메모리 공간(힙 메모리)을 할당 받는다는 뜻입니다.
- 이렇게 실제로 사용할 수 있도록 생성된 클래스를 **인스턴스**라고 합니다.
- 그리고 인스턴스를 가리키는 클래스형 변수를 **참조변수**라고 합니다.

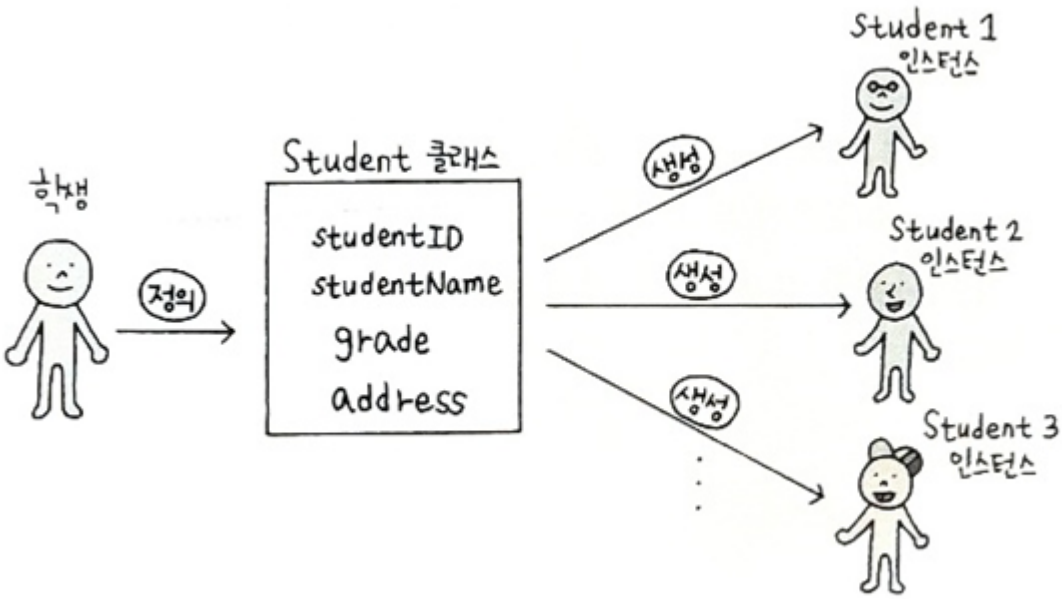
```
Student studentAhn = new Student();
```

- Student 클래스 자료형으로 studentAhn 변수를 선언하고 new Student();로 Student클래스를 생성하여 studentAhn에 대입한다는 뜻
- studentAhn을 참조변수라고 하고, 이 변수가 생성된 인스턴스를 가리킵니다.

인스턴스와 참조 변수

객체, 클래스, 인스턴스

- **객체**란 '의사나 행위가 미치는 대상'이며 이를 코드로 구현한 것이 **클래스**입니다.
- 클래스가 메모리 공간에 생성된 상태를 **인스턴스**라고 합니다.
- 또한 **생성된 클래스의 인스턴스**를 **객체**라고도 합니다.



- 클래스의 생성자를 호출하면 인스턴스가 만들어집니다.
- **클래스는 하나**이지만, 이 클래스로부터 **여러 개의 각기 다른 인스턴스를 생성**할 수 있습니다. (예 : 어떤 학교에서 학생이라는 추상어는 뜻이 하나이지만, 그 학교에 다니느 학생 개개인은 여러 명인 것과 마찬가지)

day05_07/StudentTest1.java - 인스턴스 여러개 생성하기

```
package day05_07;

public class StudentTest1 {
	public static void main(String[] args) {

		Student student1 = new Student();  // 첫 번째 학생 생성
		student1.studentName = "안연수";
		System.out.println(student1.getStudentName());

		Student student2 = new Student();  // 두 번째 학생 생성
		student2.studentName = "안승연";
		System.out.println(student2.getStudentName());
```

```
        }
}
```

실행결과

```
안연수
안승연
```

참조변수 사용하기

- 참조변수를 사용하면 인스턴스의 멤버변수와 메서드를 참조하여 사용할 수 있는데 이때 마침표(.)연산자를 사용합니다.

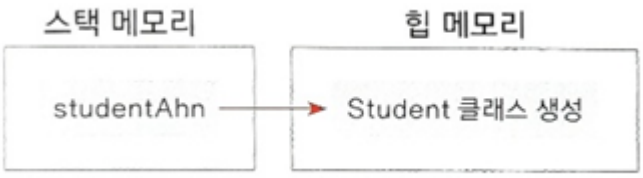
```
참조변수.멤버변수
참조변수.메서드

studentAhn.studentName = "안연수";  // 멤버변수 사용
System.out.println(studentAhn.getStudentName());  // 메서드 사용
```

인스턴스와 힙 메모리

- new Student()를 선언하면 Student하나가 생성되는데 각 Student는 StudentID, StudentName 등의 멤버변수를 가지고 있습니다.
- 그런데 이들 변수를 저장할 공간이 있어야 합니다. 이때 사용하는 메모리가 **힙 메모리**(heap memory)입니다.
- 클래스 생성자를 하나 호출하면 인스턴스가 힙 메모리에 생성되는 것입니다.

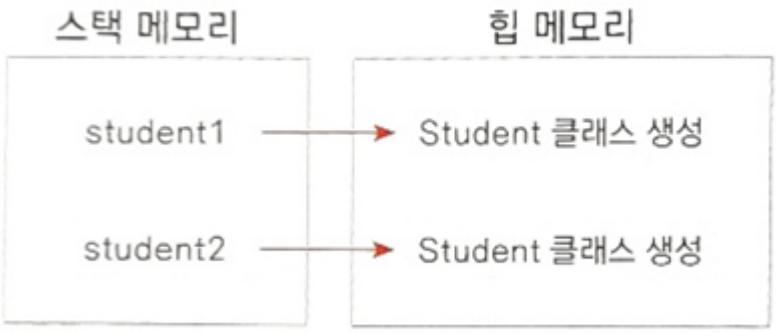
```
Student studentAhn = new Student();
```



- studentAhn 변수는 **지역변수** 입니다. 지역변수는 **스택 메모리**에 생성됩니다.
- 인스턴스는 **힙 메모리**에 생성됩니다.
- 지역변수 studentAhn에 생성된 인스턴스를 대입하는 것은 **studentAhn에 인스턴스가 생성된 힙 메모리의 주소를 대입한다는 것**과 같은 의미입니다.
- 다음과 같이 두 개의 인스턴스를 생성

```
Student student1 = new Student();
Student student2 = new Student();
```

- 생성된 두 인스턴스는 당연히 각각 다른 메모리 공간을 차지합니다. 따라서 student1.studentName과 student2.studentName은 서로 다른 값을 가지게 됩니다.



- 클래스가 생성될 때마다 인스턴스는 다른 메모리 공간을 차지합니다.
- **멤버변수를 저장하는 공간이 매번 따로 생긴다는 의미**입니다.
- 이런 이유 때문에 클래스에 선언한 멤버변수를 다른말로 **인스턴스 변수**라고 합니다.

힙 메모리란?

힙(heap)은 프로그램에서 사용하는 동적 메모리(dynamic memory) 공간을 말합니다. 일반적으로 프로그램은 **스택, 힙, 데이터** 이렇게 세 영역을 사용해야 하는데, **객체가 생성될 때 사용하는 공간이 힙**입니다. 힙은 동적으로 할당되며 사용이 끝나면 메모리를 해제해 주어야 합니다. C나 C++ 언어에서는 프로그래머가 직접 메모리를 해제해야 하지만 자바에서는 **가비지 콜렉터(garbage collector)**가 자동으로 메모리를 해제해줍니다.

참조 변수와 참조 값

- 참조 변수는 힙 메모리에 생성된 인스턴스를 가리킵니다. 그러면 참조 변수에 실제로 어떤 내용이 들어가 있는지 출력해 보겠습니다.
- 힙 메모리에 생성된 인스턴스의 메모리 주소는 참조 변수에 저장됩니다.
- 출력 내용은 **클래스 이름@주소 값** 입니다.
- 여기에 나오는 주소 값은 다른 말로 **해시 코드(hash code) 값**이라고도 합니다.
- 이 값은 자바 가상 머신에서 객체가 생성되었을 때 생성된 객체에 할당하는 **가상 주소 값**입니다.
- 따라서 student1변수를 사용하여 student1 인스턴스를 참조할 수 있습니다.
- 이때 student1을 **참조 변수**, 주소 값을 **참조 값**이라고 합니다.

용어 정리

용어	설명
객체	객체 지향 프로그램의 대상, 생성된 인스턴스
클래스	객체를 프로그래밍 하기 위해 코드로 만든 상태

용어	설명
인스턴스	클래스가 메모리에 생성된 상태
멤버변수	클래스의 속성, 특징
메서드	멤버 변수를 이용하여 클래스의 기능을 구현
참조 변수	메모리에 생성된 인스턴스를 가리키는 변수
참조 값	생성된 인스턴스의 메모리 주소 값

생성자

day05_07/constructor/Person.java

```
package day05_07.constructor;

public class Person {
        String name;
        float height;
        float weight;
}
```

day05_07/constructor/PersonTest.java

```
package day05_07.constructor;

public class PersonTest {
        public static void main(String[] args) {
                Person personLee = new Person(); // Person()  - 생성자
        }
}
```

- 클래스를 생성할 때 사용하는 Person()과 같은 함수를 **생성자**라고 합니다.
- 생성자가 하는 역할은 인스턴스의 생성과 인스턴스를 생성할 때 멤버 변수나 상수를 초기화하는 것입니다.

디폴트 생성자

- 생성자는 클래스를 생성할 때만 호출합니다.
- 생성자 이름은 클래스 이름과 같습니다.
- 생성자는 반환값이 없습니다.
- 상기 코드에서는 Person() 생성자가 따로 없는데 생성자가 없는 클래스는 클래스파일을 컴파일 할때 자바 컴파일러에서 자동으로 생성자를 만들어 줍니다.
- 이렇게 자동으로 만들어 주는 생성자를 **디폴트 생성자**(default constructor)라고 합니다.
- **디폴트 생성자**는 매개변수가 없고 구현코드도 없습니다.

```
package day05_07.constructor;

public class Person {
	String name;
	float height;
	float weight;

	// 자바 컴파일러가 자동으로 제공하는 디폴트 생성자
	public Person() {}
}
```

생성자 만들기

- 생성자는 주로 멤버변수에 대한 값들을 매개변수로 받아서 인스턴스가 새로 생성될 때 멤버 변수 값들을 초기화하는 역할을 합니다.
- 즉, 인스턴스가 생성됨과 동시에 멤버변수의 값을 지정하고 인스턴스를 초기화하기 위해 생성자를 직접 구현하여 사용합니다.

day05_07/constructor/Person.java

```
package day05_07.constructor;

public class Person {
	String name;
	float height;
	float weight;

	/**
	 * 사람 이름을 매개변수로 입력받아서
	 * Person 클래스를 생성하는 생성자
	 */
	public Person(String pname) {
		name = pname;
	}
}
```

- 새로 만든 생성자는 문자열 String형 매개변수를 하나 입력 받아서 이름을 지정합니다.
- 그러나 PersonTest.java코드에서 오류가 발생합니다.

day05_07/constructor/PersonTest.java

```
package day05_07.constructor;

public class PersonTest {
	public static void main(String[] args) {
		Person personLee = new Person(); // 오류 발생
	}
}
```

- 자바 컴파일러는 생성자가 하나도 없는 경우에만 디폴트 생성자를 제공합니다.
- 프로그래머가 생성자를 직접 추가하면 디폴트 생성자는 만들어지지 않습니다.
- 따라서 PersonTest커드는 디폴트 생성자가 없어서 오류가 난 것 입니다.
- 오류를 없애려면 **매개변수가 있는 생성자로 호출**하거나 프로그래머가 **디폴트 생성자를 추가**로 직접 구현하면 됩니다.

day05_07/constructor/PersonTest.java

```
package day05_07.constructor;

public class Person {
        String name;
        float height;
        float weight;

        public Person() {} // 디폴트 생성자 직접 추가

        /**
         * 사람 이름을 매개변수로 입력받아서
         * Person 클래스를 생성하는 생성자
         */
        public Person(String pname) {
                name = pname;
        }
}
```

생성자 오버로드

- 클래스에서 생성자가 두개 이상 제공되는 경우를 **생성자 오버로드**(constructor overload) 합니다.
- 필요에 따라 매개변수가 다른 생성자를 여러 개를 만들 수 있습니다.
- 클래스에 생성자를 여러 개 제공하면 이 클래스를 사용하는 코드에서는 원하는 생성자를 선택해 사용할 수 있습니다.
- 경우에 따라서는 클래스에서 일부러 디폴트 생성자를 제공하지 않기도 합니다.

> 객체지향 프로그램에서 메서드 이름이 같고 매개변수만 다른 경우를 **오버로드**라고 합니다.

day05_07/constructor/Person.java

```
package day05_07.constructor;

public class Person {
        String name;
        float height;
        float weight;

        public Person() {} // 디폴트 생성자 직접 추가

        /**
```

```
         * 사람 이름을 매개변수로 입력받아서
         * Person 클래스를 생성하는 생성자
         */
        public Person(String pname) {
                name = pname;
        }

        //이름 키, 몸무게를 매개변수로 입력받는 생성자
        public Person(String pname, float pheight, float pweight) {
                name = pname;
                height = pheight;
                weight = pweight;
        }
}
```

day05_07/constructor/PersonTest.java

```
package day05_07.constructor;

public class PersonTest2 {
        public static void main(String[] args) {
                Person personKim = new Person();
                personKim.name = "김유신";
                personKim.weight = 85.5f;
                personKim.height = 180.0f;

                Person personLee = new Person("이순신", 175, 75);
        }
}
```

참조 자료형

크기가 정해진 기본 자료형(int, char, float, double 등)으로 선언하는 변수가 있고, 클래스형으로 선언하는 참조 자료형 변수가 있다.\

day05_07/reference/Subject.java

```
package day05_07.reference;

public class Subject {
        String subjectName;
        int scorePoint;
}
```

day05_07/reference/Student3.java

```
package day05_07.reference;

public class Subject3 {
        int studentID;
```

```
        String studentName;

        // Subect 참조 자료형을 사용하여 선언
        Subject korean;
        Subject math;
}
```

- 기본자료형 : int studentID
- 참조자료형 : String studentgName, Subject korean, Subject math

정보 은닉

- 객체 지향 프로그램에서는 예약어를 사용해 클래스 내부의 변수나 메서드, 생성자에 대한 접근 권한을 지정할 수 있습니다.
- 이러한 예약어를 '**접근 제어자**(access modifier)'라고 합니다.

접근제어자 정리

접근 제어자	설명
public	외부 클래스 어디에서나 접근할 수 있습니다.
protected	같은 패키지 내부와 상속 관계의 클래스에서만 접근할 수 있고 그 외 클래스에서는 접근할 수 없습니다.
아무것도 없는 경우	default이며 같은 패키지 내부에서만 접근할 수 있습니다.
private	같은 클래스 내부에서만 접근할 수 있습니다.

day05_07/hiding/Student.java - private 사용하기

```
package day05_07.hiding;

public class Student {
        int studentID;
        // studentName 변수를 private으로 선언
        private String studentName;
        int grade;
        String address;
}
```

day05_07/hiding/StudentTest.java - private 사용하기

```
package day05_07.hiding;

public class StudentTest {
        public static void main(String[] args) {
                Student studentLee = new Student();
                studentLee.studentName = "이상원"; // 오류발생
```

```
        }
}
```

- StudentTest.java파일에 오류가 발생합니다.
- studentName 변수의 접근 제어자가 public일 때는 외부 클래스인 StudentTest.java 클래스에서 이 변수에 접근할 수 있었지만, private으로 바뀌면서 외부 클래스의 접근이 허용되지 않기 때문입니다.

get(), set() 메서드

- private으로 선언한 studentName 변수를 외부 코드에서 사용하려면 public 메서드를 제공해야 한다.
- public 메서드가 제공되지 않는다면 studentName 변수에 접근할 수 있는 방법은 없습니다.
- 이때 사용할 수 있는 것이 get(), set()메서드 입니다.

> 값을 받는 get() 메서드를 getter, 값을 지정하는 set()메서드를 setter라고도 부릅니다.

day05_07/hiding/Student.java

```
package day05_07.hiding;

public class Student {
        int studentID;
        // studentName 변수를 private으로 선언
        private String studentName;
        int grade;
        String address;

        public String getStudentName() {
                return studentName;
        }

        public void setStudentName(String studentName) {
                this.studentName = studentName;
        }
}
```

- 학생 이름을 받아오거나 지정할 수 있도록 getStudentName()메서드와 setStudentName()메서드를 추가했습니다.

day05_07/hiding/StudentTest.java

```
package day05_07.hiding;

public class StudentTest {
        public static void main(String[] args) {
                Student studentLee = new Student();
                //studentLee.studentName = "이상원"; // 오류발생
```

```
                // setStudentName() 메서드 활용해 private 변수에 접근 가능
                studentLee.setStudentName("이상원");

                System.out.println(studentLee.getStudentName());
        }
}
```

- studentName 멤버 변수에 이름 값을 직접 대입하는 것이 아니고 setStudentName()메서드를 활용하여 값을 대입할 수 있습니다.
- 즉 외부 클래스에서 private 변수에 직접 접근할 수는 없지만, public 메서드를 통하면 private변수에 접근할 수 있습니다.

정보 은닉이란?

- 클래스의 멤버 변수를 public으로 선언하면 접근이 제한되지 않으므로 정보의 오류가 발생할 수 있습니다.
- 이런 경우 오류가 나더라도 그 값이 해당 변수에 대입되지 못하도록 다음과 같이 변수를 private으로 바꾸고 public 메서드를 별도로 제공해야 합니다.

day05_07/hiding/MyDate.java

```
package day05_07.hiding;

public class MyDate {
        public int day;
        public int month;
        public int year;
}
```

day05_07/hiding/MyDateTest.java

```
package day05_07.hiding;

public class MyDateTest {
        public static void main(String[] args) {
                MyDate date = new MyDate();
                date.month = 2;
                date.day = 31;
                date.year = 2018;
        }
}
```

- 이처럼 클래스 내부에서 사용할 변수나 메서서드는 private으로 선언해서 외부에서 접근하지 못하도록 하는 것을 객체 지향에서는 '**정보은닉**(information hiding)'이라고 합니다.

day05_07/hiding/MyDate2.java

```
package day05_07.hiding;
```

```
public class MyDate2 {
	private int day;
	private int month;
	private int year;

	public void setDay(int day) {
		if (month == 2) {
			if (day < 1 || day > 28) {
				System.out.println("오류입니다.");
			} else {
				this.day = day;
			}
		}
	}
}
```

day05_07/hiding/MyDateTest2.java

```
package day05_07.hiding;

public class MyDateTest2 {
	public static void main(String[] args) {
		MyDate2 date = new MyDate2();

		date.setYear(2018);
		date.setMonth(2);
		date.setDay(31);
	}
}
```

- 정보은닉은 객체지향 프로그래밍의 특징 중에 하나이며 자바에서는 접근 제어자를 사용하여 정보은닉을 구현합니다.
- 모든 변수를 private으로 선언해야 하는 것은 아니지만, 필요한 경우에는 private으로 선언하여 오류를 막을 수 있습니다.

this 예약어

자신의 메모리를 가리키는 this

- this는 생성된 인스턴스 스스로를 가리키는 예약어입니다.

day05_07/thisex/ThisExample.java

```
package day05_07.thisex;

class BirthDay {
	int day;
	int month;
	int year;

	public void setYear(int year) {
```

```
                this.year = year;  // bDay.year = year;와 같음
        }

        public void printThis( ) {
                System.out.println(this); // System.out.println(bDay);와 같음
        }
}

public class ThisExample {
        public static void main(String[] args) {
                BirthDay bDay = new BirthDay();
                bDay.setYear(2000);
                System.out.println(bDay);
                bDay.printThis();
        }
}
```

실행결과

```
day05_07.thisex.BirthDay@5e91993f
day05_07.thisex.BirthDay@5e91993f
```

- 인스턴스를 가리키는 변수가 참조변수 이며, 참조변수를 출력하면 '**클래스 이름@메모리 주소**' 문자열 값이 나옵니다.
- 출력 결과를 보면 bDay.printThis()메서드를 호출하여 출력한 this 값이 참조 변수 bDay를 출력한 값과 같습니다.
- 즉, 클래스 코드에서 사용하는 **this**는 **생성된 인스턴스 자신**을 가리키는 역할을 합니다.
- this.year = year; 문장으로 참조하면 **동적메모리에서 생성된 인스턴스 year 변수 위치를 가리키고 그 위치에 매개변수 값을 넣어 주는 것**입니다.

- 위 그림에서 main() 함수는 ThisExample 클래스의 시작 메서드 입니다.
- 그림을 보면 main() 함수에서 bDay 변수가 가리키는 인스턴스와 BirthDay 클래스의 setYear()메서드에서 this가 가리키는 인스턴스가 같은 곳에 있음을 알 수 있습니다.

생성자에서 다른 생성자를 호출하는 this

- this를 사용해 클래스의 생성자에서 다른 생성자를 호출할 수 있습니다.

day05_07/thisex/CallAnotherConst.java

```
package day05_07.thisex;

class Person {
	String name;
	int age;

	Person() {
		this("이름 없음", 1);  // this를 사용해 Person(String, int) 생성자
호출
	}

	Person(String name, int age) {
		this.name = name;
		this.age = age;
	}
}

public class CallAnotherConst {
	public static void main(String[] args) {
		Person noName = new Person();
		System.out.println(noName.name);
		System.out.println(noName.age);
	}
}

실행결과

이름 없음
1
```

- this를 사용하여 생성자를 호출하는 코드 이전에 다른 코드를 넣을 수 없습니다. 만약 다른 코드를 넣으면 오류가 발생합니다.
- 생성자는 클래스가 생성될 때 호출되므로 클래스 생성이 완료되지 않은 시점에 다른 코드가 있다면 오류가 발생할 수 있습니다.
- 즉, 디폴드 생성자에서 생성이 완료되는 것이 아니라 this를 사용해 다른 생성자를 호출하므로 이때는 this를 활용한 문장이 가장 먼저 와야 합니다.

자신의 주소를 반환하는 this

- this를 사용하여 생성된 클래스 자신의 주소 값을 반환할 수 있습니다.
- 인스턴스 주소 값을 반환할 때는 this를 사용하고 반환형은 클래스 자료형을 사용합니다.

day05_07/thisex/CallAnotherConst.java

```
package day05_07.thisex;

class Person {
	String name;
	int age;

	Person() {
		this("이름 없음", 1);  // this를 사용해 Person(String, int) 생성자
```

```
호출
        }

        Person(String name, int age) {
                this.name = name;
                this.age = age;
        }

        // 반환받은 클래스형
        Person returnItSelf() {
                return this; // this 반환
        }
}

public class CallAnotherConst {
        public static void main(String[] args) {
                Person noName = new Person();
                System.out.println(noName.name);
                System.out.println(noName.age);

                Person p = noName.returnItSelf(); // this 값을 클래스 변수에 대입
                System.out.println(p);  // noName.returnItSelf()의 반환 값 출력
                System.out.println(noName); // 참조변수 출력
        }
}


실행결과

이름 없음
1
day05_07.thisex.Person@5e91993f
day05_07.thisex.Person@5e91993f
```

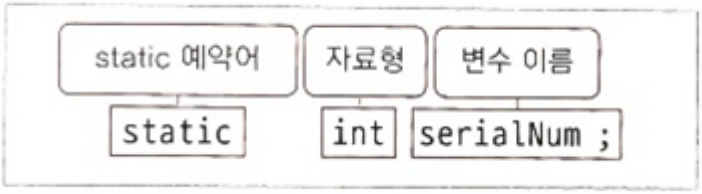
static 변수

변수를 여러 클래스에서 공통으로 사용하려면?

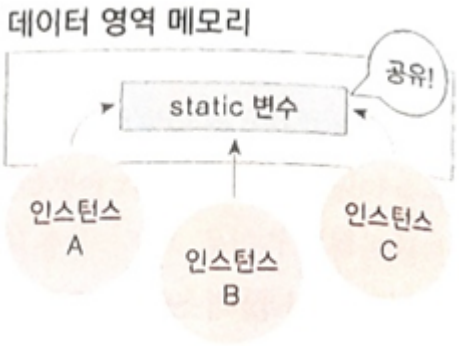
클래스에서 공통적으로 사용하는 변수를 'static 변수'로 선언합니다.

static 변수의 정의와 사용방법

- static 변수는 다른 용어로 **정적변수**라고도 합니다.
- static 변수는 자바 뿐만 아니라 다른 언어에서도 비슷한 개념으로 사용하고 있는 변수로서 자바에서는 다른 멤버 변수처럼 클래스 내부에 선언합니다.
- 변수를 선언할때 다음과 같이 자료형 앞에 static 예약어를 사용합니다.



- static 변수는 클래스 내부에 선언하지만, 다른 멤버 변수처럼 인스턴스가 생성될 때마다 새로 생성되는 변수가 아닙니다.
- static 변수는 프로그램이 실행되어 메모리에 올라갔을 때 딱 한번 메모리 공간에 할당됩니다.
- 그리고 그 값은 모든 인스턴스가 공유합니다.



- 일반 멤버변수는 인스턴스가 생성될 때 마다 새로 생성되어 각각 다른 studentName을 가지게 되지만
- static으로 선언한 변수는 인스턴스 생성과 상관없이 먼저 생성되고 그 값을 모든 인스턴스가 공유하게 되는 것입니다.
- 이런 이유 때문에 static 변수를 클래스에 기반한 변수라고 해서 **클래스 변수**(class variable)라고도 합니다.

day05_07/staticex/Student.java

```
package day05_07.staticex;

public class Student {
	// static 변수는 인스턴스 생성과 상관없이 먼저 생성됨
	public static int serialNum = 1000;
	public int studentID;
	public String studentName;
	public int grade;
	public String address;

	public String getStudentName() {
		return studentName;
	}

	public void setStudentName(String name) {
		studentName = name;
	}
}
```

day05_07/staticex/StudentTest1.java

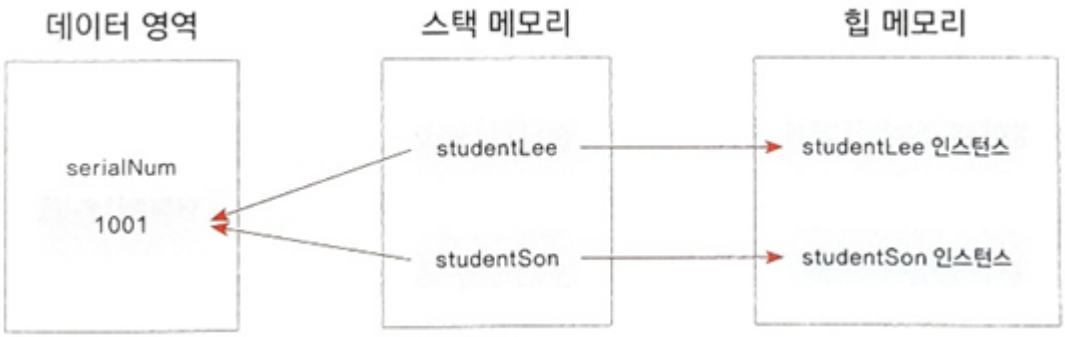
```
package day05_07.staticex;

public class StudentTest1 {
	public static void main(String[] args) {
		Student studentLee = new Student();
		studentLee.setStudentName("이지원");
		System.out.println(studentLee.serialNum); // serialNum 변수의 초깃
값 출력
		studentLee.serialNum++; // static 변수 값 증가

		Student studentSon = new Student();
		studentSon.setStudentName("손수경");
		System.out.println(studentSon.serialNum);  //증가된 값 출력
		System.out.println(studentLee.serialNum); // 증가된 값 출력
	}
}


실행결과
1000
1001
1001
```

- studentLee를 먼저 생성하고 이 참조 변수를 사용하여 전체 인스턴스에서 공통으로 사용하는 serialNum 변수 값을 1씩 증가 시킵니다.
- 그리고 studentSon을 생성합니다. 생성된 studentSon으로는 아무 연산도 수행하지 않습니다.
- 그 다음에 studentSon과 studentLee로 serialNum 변수 값을 출력해 보면 둘다 1001로 증가된 serialNum 값이 출력되는 것을 알 수 있습니다.
- static으로 선언한 serialNum 변수는 모든 인스턴스가 공유하기 때문입니다.
- 즉, 두 개의 참조 변수가 동일한 변수의 메모리를 가리키고 있다는 것을 알 수 있습니다.



- static변수는 모든 인스턴스가 공유하는 변수이므로 이 변수를 바로 학번으로 사용하면 모든 학생이 동일한 학번 값을 가지게 되므로 학생 고유 번호로써 학번으로 활용할 수 없습니다.
- 학번은 학생의 고유 번호 이므로 학생의 멤버변수로 선언해 주고, 학생이 한 명 생성될 때마다 증가한 serialNum 값을 studentID에 대입해 주면 이 문제를 해결할 수 있습니다.

day05_07/staticex/Student1.java

```
package day05_07.staticex;

public class Student1 {
```

```
        public static int serialNum = 1000;
        public int studentID;
        public String studentName;
        public int grade;
        public String address;

        public Student1() {
                serialNum++; // 학생이 생성될 때마다 증가
                studentID = serialNum;  // 증가된 값을 학번 인스턴스 변수에 부여
        }

        public String getStudentName() {
                return studentName;
        }

        public void setStudentName(String name) {
                studentName = name;
        }
}
```

day05_07/staticex/StudentTest2.java

```
package day05_07.staticex;

public class StudentTest2 {
        public static void main(String[] args) {
                Student1 studentLee = new Student1();
                studentLee.setStudentName("이지원");
                System.out.println(studentLee.serialNum);
                System.out.println(studentLee.studentName + " 학번:" +
studentLee.studentID);

                Student1 studentSon = new Student1();
                studentSon.setStudentName("손수경");
                System.out.println(studentSon.serialNum);
                System.out.println(studentSon.studentName + " 학번:" +
studentSon.studentID);
        }
}
```

실행결과

```
1001
이지원 학번:1001
1002
손수경 학번:1002
```

- 학생 인스턴스를 생성할 때 마다 serialNum 변수의 값은 증가합니다.
- 그리고 새로 생성되는 학생마다 가지는 studentID 변수에 증가한 serialNum 값을 복사해 주었으므로, 두 학생의 학번은 다릅니다.
- 이처럼 static 변수는 같은 클래스에서 생성된 인스턴스들이 같은 값을 공유할 수 있으므로, 인스턴스 간에 공통으로 사용할 값이 필요한 경우 사용할 수 있습니다.

클래스 변수

- static 변수는 인스턴스를 생성할 때마다 만들어지는 것이 아니고 클래스를 선언할 때 특정 메모리에 저장되어 모든 인스턴스가 공유하는 변수입니다.
- static 변수는 인스턴스 생성과는 별개이므로 인스턴스보다 먼저 생성됩니다. 그러므로 인스턴스가 아닌 **클래스 이름으로도 참조하여 사용할 수 있습니다**.
- 따라서 자바에서는 static 변수를 **클래스 변수**라고도 합니다.

day05_07/staticex/StudentTest3.java

```
package day05_07.staticex;

public class StudentTest3 {
	public static void main(String[] args) {
		Student1 studentLee = new Student1();
		studentLee.setStudentName("이지원");
		System.out.println(Student1.serialNum); // serialNum 변수를 직접 클
래스 이름으로 참조
		System.out.println(studentLee.studentName + " 학번:" +
studentLee.studentID);

		Student1 studentSon = new Student1();
		studentSon.setStudentName("손수경");
		System.out.println(Student1.serialNum);  // serialNum 변수를 직접
클래스 이름으로 참조
		System.out.println(studentSon.studentName + " 학번:" +
studentSon.studentID);
	}
}
```

- static 변수 serialNum을 studentLee.serialNum과 같이 인스턴스 변수로 참조할 수 도 있습니다.
- 하지만 static변수는 인스턴스가 생성되지 않아도 사용할 수 있기 때문에 보통은 Student.serialNum과 같이 클래스 이름과 함께 사용합니다.
- static변수, 정적변수, 클래스변수라는 세가지 용어가 있지만 모두 static변수로 같은 의미이다.
- 자바에서 static변수를 **클래스 변수**라고 하는 이유는 인스턴스마다 생성되는 변수가 아니라 클래스에 속해 한 번만 생성되는 변수이고 이를 여러 인스턴스가 공유하기 때문입니다.

클래스 메서드

- static 변수를 위한 메서드
- 이런 메서드를 **static메서드** 또는 **클래스 메서드**라고 합니다.

day05_07/staticex/Student2.java

```
package day05_07.staticex;

public class Student2 {
	private static int serialNum = 1000; // private 변수로 변경
```

```
        public int studentID;
        public String studentName;
        public int grade;
        public String address;

        public Student2() {
                serialNum++;
                studentID = serialNum;
        }

        public String getStudentName() {
                return studentName;
        }

        public void setStudentName(String name) {
                studentName = name;
        }

        // serialNum의 get()메서드
        public static int getSerialNum() {
                int i = 10;
                return serialNum;
        }

        // serialNum의 set()메서드
        public static void setSerialNum(int serialNum) {
                Student2.serialNum = serialNum;
        }
}
```

day05_07/staticex/StudentTest4.java

```
package day05_07.staticex;

public class StudentTest4 {
        public static void main(String[] args) {
                Student2 studentLee = new Student2();
                studentLee.setStudentName("이지원");
                // serialNum 값을 가져오기 위해 get()메서드를 클래스 이름으로 직접 호
출
                System.out.println(Student2.getSerialNum());
                System.out.println(studentLee.studentName + " 학번:" +
studentLee.studentID);

                Student2 studentSon = new Student2();
                studentSon.setStudentName("손수경");
                // serialNum 값을 가져오기 위해 get()메서드를 클래스 이름으로 직접 호
출
                System.out.println(Student2.getSerialNum());
                System.out.println(studentSon.studentName + " 학번:" +
studentSon.studentID);
        }
}
```

클래스 메서드와 인스턴스 변수

- 클래스 메서드 내부에서는 인스턴스 변수를 사용할 수 없습니다.
- 마찬가지로 클래스 메서드 내부에서는 인스턴스 메서드 역시 사용할 수 없습니다.
- **클래스 변수**, 클래스 메서드는 인스턴스가 만들어지기 이전에 존해하므로 인스턴스에 접근할 수 없습니다.
- **클래스 변수**, 클래스 메서드는 인스턴스가 생성되지 않아도 사용할 수 있습니다.

```
public class Student2 {
        private static int serialNum = 1000; // private 변수로 변경
        public int studentID;
        public String studentName;
        public int grade;
        public String address;

        ...
        public static int getSerialNum() {
                int i = 10;
                studentName = "이지원"; // 오류발생
                return serialNum;
        }
        ...
```

day05_07/staticex/StudentTest5.java

```
package day05_07.staticex;

public class StudentTest5 {
        public static void main(String[] args) {
                // 인스턴스 생성 없이 호출 가능
                System.out.println(Student2.getSerialNum());
        }
}
```

실행결과

```
1000
```

변수의 유효범위

지역변수의 유효범위

- 지역변수는 함수나 메서드 내부에 선언하기 때문에 함수 밖에서는 사용할 수 없습니다.
- 스택에서 생성되는 지역변수는 함수가 호출될 때 생성되었다가 함수가 반환되면 할당되었던 메모리 공간이 해제되면서 함께 없어집니다.

멤버변수의 유효범위

- 멤버변수는 인스턴스 변수라고도 합니다.
- 클래스가 생성될 때 힙(heap)메모리에 생성되는 변수입니다.
- 멤버변수는 클래스의 어느 메서드에서나 사용할 수 있습니다.
- 힙에 생성된 인스턴스가 가비지 컬렉터(garbage collector)에 의해 수거되면 메모리에서 사라집니다.

static 변수의 유효범위

- 사용자가 프로그램을 실행하면 메모리 프로그램이 상주합니다.
- 이때 프로그램 영역 중에 데이터 영역이 있습니다. 이 영역에는 상수나 문자열, static 변수가 생성됩니다.
- static 변수는 클래스 생성과 상관 없이 데이터 영역 메모리에 생성됩니다.
- 따라서 인스턴스 변수와 static 변수는 사용하는 메모리가 다릅니다.
- static 변수는 private이 아니라면 클래스 외부에서도 객체생성과는 무관하게 사용할 수 있습니다.
- 프로그램 실행이 끝난 뒤에 메모리에서 내려가면 static 변수도 소멸됩니다.
- static 변수는 프로그램이 시작할 때 부터 끝날 때 까지 메모리에 상주하므로 크기가 너무 큰 변수를 static으로 선언하는것은 좋지 않습니다.

static 응용 - 싱글톤 패턴

- 프로그램을 구현하다 보면 여러개 인스턴스가 필요한 경우도 있고 단 하나의 인스턴스만 필요한 경우도 있습니다.
- 객체 지향 프로그램에서 인스턴스를 단 하나만 생성하는 디자인 패턴을 **싱글톤 패턴**(singleton pattern)이라고 합니다.

싱글톤 패턴으로 회사 클래스 구현하기

1. 생성자를 private으로 만들기
    - 생성자가 public 이면 외부에서 인스턴스를 여러개 생성할 수 있습니다.
    - 따라서 싱글톤 패턴에서는 생성자를 반드시 명시적으로 만들고 그 접근 제어자를 private으로 지정해야 합니다.
    - 그러면 생성자가 있으므로 컴파일러가 디폴트 생성자를 만들지 않고, 접근 제어자가 private이므로 외부 클래스에서 마음대로 Company 인스턴스를 생성할 수 없게 됩니다.
    - 즉, Company 클래스 내부에서만 클래스의 생성을 제어할 수 있습니다.
2. 클래스 내부에 static으로 유일한 인스턴스 생성하기
    - private으로 선언하여 외부에서 이 인스턴스에 접근하지 못하도록 제한해야 합니다.
3. 외부에서 참조 할 수 있는 public 메서드 만들기
    - private으로 선언한 유일한 인스턴스를 외부에서도 사용할 수 있도록 설정해야 합니다.
    - 이를 위해 public 메서드를 생성합니다. 그리고 유일하게 생성한 인스턴스를 반환해 줍니다.

- 이 때 인스턴스를 반환하는 메서드는 반드시 static으로 선언해야 합니다.
- 왜냐하면 인스턴스 생성과 상관없이 호출할 수 있어야 하기 때문입니다.

day05_07/singleton/Company.java

```
package day05_07.singleton;

public class Company {

	private static Company instance = new Company();

	private Company() {}

	public static Company getInstance() {
		if (instance == null) {
			instance = new Company();
		}

		return instance;
	}
}
```

day05_07/singleton/CompanyTest.java

```
package day05_07.singleton;

public class CompanyTest {
	public static void main(String[] args) {
		// 클래스 이름으로 getInstance() 호출하여 참조 변수에 대입
		Company myCompany1 = Company.getInstance();
		Company myCompany2 = Company.getInstance();

		System.out.println(myCompany1 == myCompany1); // 두 변수가 같은 주
소인지 확인
	}
}

실행결과
true
```