

# 객체 지향 입문



Written By. 신채린

## 1. 객체지향 프로그래밍과 클래스

[객체란 무엇인가?](#)

[객체지향 프로그래밍이란?](#)

[클래스](#)

[멤버 변수](#)

[메서드](#)

[Etc.](#)

## 2. 클래스 사용하기

[클래스 생성하기](#)

[클래스의 속성, 메서드 참조하기](#)

[예시](#)

## 3. 함수와 메서드

[함수란?](#)

[함수 호출과 스택 메모리](#)

## 4. 메서드

## 5. 인스턴스 생성과 힙 메모리

[인스턴스](#)

[힙 메모리](#)

[멤버변수](#)

[참조변수와 참조값](#)

## 6. 생성자

[생성자 \(constructor\)](#)

[기본 생성자 \(default constructor\)](#)

[예시](#)

## 7. 생성자 오버로딩

[private](#)

## 9. 참조 자료형 (reference data type)

[변수의 자료형](#)

[참조 자료형이란?](#)

[참조 자료형 직접 만들어 사용하기](#)

## 10. 정보 은닉

[접근 제어 지시자 \(access modifier\)](#)

[get\(\)/set\(\) 메서드](#)

[정보 은닉 \(information hiding\)](#)

## 11. 캡슐화 (encapsulation)

[정보 은닉을 활용한 캡슐화](#)

[레포트 만들기 예제](#)

## 12. this에 대하여

[this의 역할](#)

[생성된 인스턴스 메모리의 주소를 가짐](#)

[생성자에서 다른 생성자를 호출하는 this](#)

[자신의 주소를 반환하는 this](#)

[예시](#)

## 13. 객체 간의 협력 (collaboration)

## 16. static 변수, 메서드

[static 변수](#)

[static 변수 선언과 사용하기](#)

[Static 변수 test 하기](#)

[static 변수와 인스턴스 변수](#)

[static 메서드](#)

[예시](#)

#### 17. static 메서드의 구현과 활용, 변수의 유효 범위

[static 메서드 만들기](#)

[static 메서드 \(클래스 메서드\)에서는 인스턴스 변수를 사용할 수 없다.](#)

[변수의 유효범위와 메모리](#)

#### 18. static 응용 - singleton pattern

[singleton pattern](#)

[예시 - 싱글톤 패턴으로 회사 객체 구현하기](#)

#### 20. 배열이란

[배열이란?](#)

[배열 선언과 초기화](#)

[배열 사용하기](#)

[배열의 길이와 요소의 개수는 동일하지 않습니다.](#)

[문자 배열을 만들어 A-Z 까지 배열에 저장하고 이를 다시 출력하기](#)

[항상된 for문 사용하기](#)

#### 21. 객체 배열

[기본 자료형 배열과 참조 자료형 배열 \(객체 배열\)](#)

[객체 배열 구현](#)

[객체 배열 복사](#)

[얕은 복사](#)

[깊은 복사](#)

[항상된 for 문](#)

#### 22. 다차원 배열

#### 23. ArrayList

[java.util 패키지에서 제공되는 ArrayList](#)

[주요 메서드](#)

[ArrayList를 활용한 간단한 예제](#)

#### 24. ArrayList를 활용한 간단한 성적 산출 프로그램

[예제 시나리오](#)

[Student 클래스](#)

[Subject 클래스](#)

[실행하기](#)

[결과](#)

## 1. 객체지향 프로그래밍과 클래스

### 객체란 무엇인가?

- 객체 (Object)
  - 의사나 행위가 미치는 대상 (사전적 의미)
  - 구체적, 추상적 데이터의 단위

### 객체지향 프로그래밍이란?

- 객체지향 프로그래밍 (Object Oriented Programming: OOP)
  - 객체를 기반으로 하는 프로그래밍
  - 객체를 정의하고, 객체의 기능을 구현하며, 객체 간의 협력(cooperation)을 구현
- 절차지향 프로그래밍 (Procedural Programming)
  - 시간이나 사건의 흐름에 따른 구현, 일련의 과정을 구현
  - C 언어

### 클래스

- 객체를 코드로 구현한 것
- 객체 지향 프로그래밍의 가장 기본적인 요소

## 멤버 변수

- 객체가 가지는 속성을 변수로 표현
- 클래스의 멤버변수
- member variable, property, attribute

## 메서드

- 객체의 기능을 구현
- 함수의 일종으로 객체 안에서 사용하는 함수이다.
- method, member function

## Etc.

- 파일 이름과 public class에 적혀있는 이름은 동일해야 한다.

## 2. 클래스 사용하기

### 클래스 생성하기

- new 키워드를 사용하여 생성자로 생성
- 예시) Student studentLee = new Student();

### 클래스의 속성, 메서드 참조하기

- 생성에 사용한 변수(참조변수)로 클래스의 속성, 메서드 참조
- 예시)
  - studentName = "Lee";
  - showStudentInfo();
- 

## 예시

```
package Chapter5;

public class Student {
    // 아래 public은 접근제어자이다.
    public int studentID;
    public String studentName;
    public String address;

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

```
package Chapter5;
```

```
public class StudentTest {
    public static void main(String[] args) {
        // new는 생성자 - 메모리에 생성이 됨!
        Student studentLee = new Student();
        studentLee.studentName = "이순신";
        studentLee.address = "서울";
        studentLee.showStudentInfo();
    }
}
```

### 3. 함수와 메서드

#### 함수란?

- 함수 (function)
  - 하나의 기능을 수행하는 일련의 코드
  - 함수는 호출하여 사용하고 기능이 수행된 후 값을 반환할 수 있음
  - 함수로 구현된 기능은 여러 곳에서 호출되어 사용될 수 있음



- 함수의 입력과 반환
  - 3가지 요소 존재 (입력, 연산, 출력)
    - 매개변수 (입력), 반환값 (출력)은 없을 수도 있다.
- 함수 정의하기
  - 함수는 이름, 매개변수, 반환 값, 함수 몸체(body)로 구성됨

```
int add(int num1, int num2) {
    int result;
    result = num1 + num2;
    return result;
}
```

- 함수에 반환하는 값이 없다면 **void**를 추가한다.
- 예시)

```
package Chapter5;

public class FunctionTest {
    // 매개 변수 2개를 받아서 반환해준다.
    public static int addNum(int num1, int num2) {
        int result;
        result = num1 + num2;
    }
}
```

```

        return result;
    }

    // return 값이 없을 때 void
    public static void sayHello(String greeting) {
        System.out.println(greeting);
    }

    public static int calcSum() {
        int sum = 0;
        int i;

        for (i = 0; i <= 100; i++) {
            sum += i;
        }
        return sum;
    }

    public static void main(String[] args) {
        int n1 = 10;
        int n2 = 20;

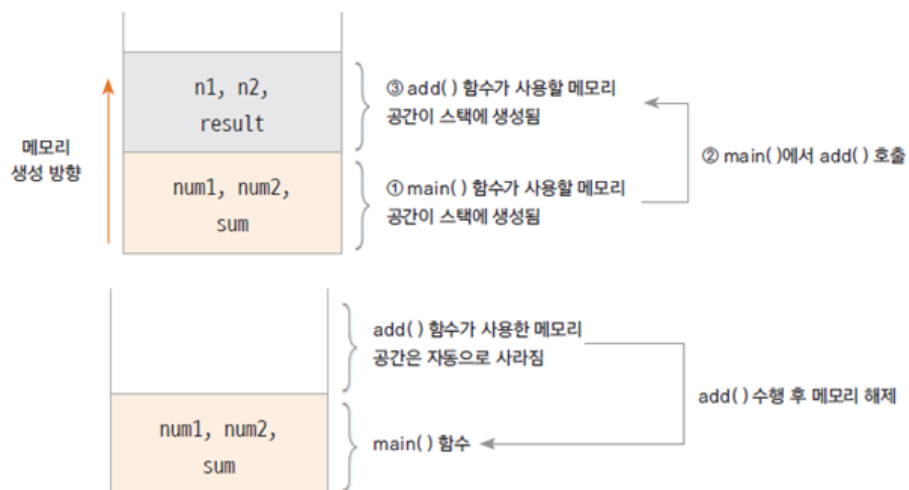
        int total = addNum(n1, n2);

        sayHello("안녕하세요");
        int num = calcSum();

        System.out.println(total); //30
        System.out.println(num); //5050
    }
}

```

## 함수 호출과 스택 메모리



- 함수에서는 스택 메모리를 사용한다.
  - **스택**: 함수가 호출될 때 지역 변수들이 사용하는 메모리이다.
  - 함수의 수행이 끝나면 자동으로 메모리가 반환된다 (사라진다)

## 4. 메서드

- 객체의 기능을 구현하기 위해 클래스 내부에 구현되는 함수

- 메서드를 구현함으로써 객체의 기능이 구현 됨
- 메서드의 이름은 사용하는 쪽 (클라이언트 코드)에 맞게 명명하는 것이 좋음
  - 예시) `getStudentName`
- 멤버 변수를 이용하여 클래스의 기능을 구현한 함수
- 메서드 예시

```
package Chapter5;

public class Student {
    // 아래 public은 접근제어자라고 한다.
    public int studentID;
    public String studentName;
    public String address;

    public void showStudentInfo() {
        System.out.println(studentID + "학번의 이름은 " + studentName + "이고, 주소는 " + address + "입니다.");
    }

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

```
package Chapter 5;

public class StudentTest {
    public static void main(String[] args) {
        // 이렇게 생성되는 객체는 인스턴스이다. Student 클래스 기반!
        Student studentLee = new Student();

        studentLee.studentID = 12345;
        studentLee.setStudentName("Lee");
        studentLee.address = "서울 강남구";

        studentLee.showStudentInfo();

        Student studentKim = new Student();
        studentKim.studentID = 54321;
        studentKim.studentName = "Kim";
        studentKim.address = "경기도 성남시";

        studentKim.showStudentInfo();
    }
}
```

## 5. 인스턴스 생성과 힙 메모리

### 인스턴스

- 클래스 객체의 속성을 정의하고, 기능을 구현하며 만들어 놓은 코드 상태
- 실제 클래스를 기반으로 생성된 객체 (인스턴스)는 각각 다른 멤버 변수를 갖게 된다.
- 클래스로부터 생성된 객체 (클래스가 메모리에 생성된 상태)
- **new** 키워드를 이용하여 여러 개의 인스턴스를 생성

## 힙 메모리

- 객체가 생성되면 인스턴스는 **heap 메모리**에 멤버 변수의 크기에 따라 메모리가 생성
  - heap은 동적 메모리이다.
- 자바에서 Garbage Collector가 주기적으로 사용하지 않는 메모리를 수거한다.
- 하나의 클래스로 부터 여러개의 인스턴스가 생성되고 각각 다른 메모리 주소를 가지게 됨

## 멤버변수

- 클래스의 속성 및 특성
- 멤버변수는 만들어지면, 자동으로 초기화 된다.
  - int의 경우는 0
  - string의 경우는 null
  - boolean의 경우는 false

## 참조변수와 참조값

- 참조변수
  - 메모리에 생성된 인스턴스를 가리키는 변수
- 참조값
  - 생성된 인스턴스의 메모리 주소 값
- `package 이름.class 이름@16진수` 로 참조 변수가 가리키는 메모리 참조값

```
package Chapter5;

public class StudentTest {
    public static void main(String[] args) {
        // new는 생성자 - 메모리에 생성이 됨!
        Student studentLee = new Student();
        studentLee.studentName = "이순신";
        studentLee.address = "서울";

        studentLee.showStudentInfo();

        Student studentKim = new Student();
        studentKim.studentName = "김유신";
        studentKim.address = "경주";

        studentKim.showStudentInfo();

        // 참조변수 출력하기
        // 앞은 package 이름.class 이름@16진수로 참조 변수가 가리키는 메모리 참조값
        // jvm이 주는 가상 address이다.
        System.out.println(studentLee); //Chapter5.Student@3d494fbf
        System.out.println(studentKim); //Chapter5.Student@1ddc4ec2
    }
}
```

## 6. 생성자

## 생성자 (constructor)

- 객체를 생성할 때 `new` 키워드와 함께 호출 (객체 생성 외에는 호출할 수 없음)
- 생성자는 일반 함수처럼 기능을 호출하는 것이 아니고, **객체를 생성**하기 위해 `new` 와 함께 호출됨
- 객체가 생성될 때, 변수나 상수를 초기화하거나 다른 초기화 기능을 수행하는 메서드를 호출함
  - 인스턴스를 초기화하는 코드가 구현됨 (주로 멤버 변수 초기화)
- 생성자는 반환 값이 없음, 상속되지 않음
- 생성자는 클래스 이름과 동일

## 기본 생성자 (default constructor)

- 하나의 클래스에는 반드시 하나 이상의 생성자가 존재해야 함
- 프로그래머가 생성자를 하나도 구현하지 않으면, Java compiler가 코드가 컴파일 되기 전에 pre-compile 단계에서 default 생성자가 추가됨
  - default 생성자는 생성자가 하나도 없을 때 생성된다.
- 기본 생성자는 매개 변수가 없고, 구현부가 없음
  - `public Student() {}`
- 만약 클래스에 다른 생성자가 있는 경우, 디폴트 생성자는 제공되지 않음

## 예시

- `Student.java`

```
public class Student {  
  
    public int studentNumber;  
    public String studentName;  
    public int grade;  
  
    public Student(int studentNumber, String studentName, int grade) {  
        this.studentNumber = studentNumber;  
        this.studentName = studentName;  
        this.grade = grade;  
    }  
  
    public String showStudentInfo() {  
        return studentName + "학생의 학번은 " + studentNumber + "이고, " + grade + "학년 입니다.";  
    }  
}
```

- `this` 키워드를 사용하여 멤버 변수를 가리킨다.
- `StudentTest.java`

```
public class StudentTest {  
  
    public static void main(String[] args) {  
  
        //Student studentLee = new Student();  
  
        Student studentLee = new Student(12345, "Lee", 3);  
  
        String data = studentLee.showStudentInfo();  
        System.out.println(data);  
    }  
}
```



## 7. 생성자 오버로딩

- 생성자를 2개 이상 구현하는 경우
- 클래스에 생성자를 따로 구현하면 기본 생성자 (default constructor)는 제공되지 않음
- 생성자를 호출하는 코드에서 여러 생성자 중 선택하여 필요에 따라 호출해서 사용할 수 있음
- private 변수도 생성자를 이용하여 초기화를 할 수 있음
- Method Overloading 이란?
  - method이름은 같지만, 매개변수가 다르다.
  - Overloading을 제공하는 이유는 사용하는 쪽의 편의성을 위해서이다.
- 예시)

- `Student.java`

```
// 생성자 직접 구현하기
public Student(String name) {
    studentName = name;
}

// 생성자 오버로딩
public Student(int id, String name) {
    studentID = id;
    studentName = name;
    address = "주소 없음";
}
```

- `UserInfo.java`

```
public class UserInfo {

    public String userId;
    public String userPassWord;
    public String userName;
    public String userAddress;
    public String phoneNumber;

    public UserInfo(){}

    public UserInfo(String userId, String userPassWord, String userName) {
        this.userId = userId;
        this.userPassWord = userPassWord;
        this.userName = userName;
    }

    public String showUserInfo() {
        return "고객님의 아이디는 " + userId + "이고, 등록된 이름은 " + userName + "입니다.";
    }
}
```

- 생성자가 2개이다. 하나는 매개변수가 존재하며, 하나는 default constructor이다.

- `UserInfoTest.java`

```
public class UserInfoTest {

    public static void main(String[] args) {

        UserInfo userLee = new UserInfo();
    }
}
```

```

userLee.userId = "a12345";
userLee.userPassWord = "zxcvbn12345";
userLee.userName = "Lee";
userLee.phoneNumber = "01034556699";
userLee.userAddress = "Seoul, Korea";

System.out.println(userLee.showUserInfo());

UserInfo userKim = new UserInfo("b12345", "09876mnbvc", "Kim");
System.out.println(userKim.showUserInfo());
}
}

```

- default constructor 외부에서 제공하면 문제가 될 수 있는 경우에는, 굳이 따로 제공하지 않아도 된다.

## private

- 클래스 내부에서만 사용되는 변수를 말한다.
- 외부에서는 참조할 수 있다.
- 정보 은닉할 때 활용한다.

## 9. 참조 자료형 (reference data type)

### 변수의 자료형



- 기본 자료형 (int, long, float, double 등)
- 참조 자료형 (String, Date, Student 등)

### 참조 자료형이란?

- 클래스형으로 변수를 선언함
- 기본 자료형은 사용하는 메모리가 정해져 있지만, 참조 자료형은 클래스에 따라 다름
- 참조 자료형을 사용할 때는 해당 변수에 대해 생성하여야 한다. (String 클래스는 예외적으로 생성하지 않고 사용할 수 있다)

### 참조 자료형 직접 만들어 사용하기



- 학생 클래스 (Student)에 있는 과목 이름, 과목 성적 속성을 과목 클래스 (Subject)로 분리하고, Subject 참조 자료형 멤버 변수를 Student에 정의하여 사용함
  - 학생과 과목에 대한 클래스를 분리하여 사용하고, Subject 클래스를 활용하여 수강한 과목들의 변수의 타입으로 선언한다.
- 선언된 Subject 변수는 생성된 인스턴스가 아니므로 Student의 생성자에서 생성하여 사용한다.

- `Student.java`

```

package ch09;

public class Student {

    int studentID;
    String studentName;

    Subject korea;
    Subject math;

    public Student(int id, String name) {
        studentID = id;
        studentName = name;

        korea = new Subject();
        math = new Subject();
    }

    public void setKoreaSubject(String name, int score) {
        korea.subjectName = name;
        korea.score = score;
    }

    public void setMathSubject(String name, int score) {
        math.subjectName = name;
        math.score = score;
    }

    public void showStudentScore() {
        int total = korea.score + math.score;
        System.out.println(studentName + " 학생의 총점은 " + total + "점 입니다." );
    }
}

```

- `Subject.java`

```

package ch09;

public class Subject {
    String subjectName;
    int score;
    int subjectID;
}

```

- StudentTest.java

```
package ch09;

public class StudentTest {

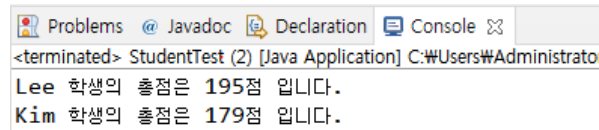
    public static void main(String[] args) {

        Student studentLee = new Student(100, "Lee");
        studentLee.setKoreaSubject("국어", 100);
        studentLee.setMathSubject("수학", 95);

        Student studentKim = new Student(101, "Kim");
        studentKim.setKoreaSubject("국어", 80);
        studentKim.setMathSubject("수학", 99);

        studentLee.showStudentScore();
        studentKim.showStudentScore();
    }
}
```

- 결과



```
<terminated> StudentTest (2) [Java Application] C:\Users\Administrato
Lee 학생의 총점은 195점 입니다.
Kim 학생의 총점은 179점 입니다.
```

## 10. 정보 은닉

### 접근 제어 지시자 (access modifier)

- 클래스 외부에서 클래스의 멤버 변수, 메서드, 생성자를 사용할 수 있는지 여부를 지정하는 키워드
- 변수, 메서드, 생성자에 대한 접근 권한 지정
- public, private, protected, 아무것도 안 쓰는 경우 (기본 접근 제어자/default)
  - public 외부에 모두 open, 패키지가 다른 경우 public 사용. 클래스 외부 어디서나 접근 할 수 있음
  - private 같은 클래스 내부에서만 접근 가능 (외부 클래스, 상속 관계의 클래스에서도 접근 불가)
  - protected 같은 패키지나 상속관계의 클래스에서 접근 가능하고, 그 외 외부에서는 접근할 수 없음
    - 클래스 간의 상속 관계가 발생할 때, 상위 클래스가 가진 private 변수나 method를 하위 클래스에 public 하게 open하고 싶을 때 protected 사용
  - 기본 접근 제어자 - 같은 **package**에서만 접근이 가능 (상속 관계라도 패키지가 다르면 접근 불가)

### get()/set() 메서드

- **private** 으로 선언된 멤버 변수 (속성/필드)에 대해 접근, 수정할 수 있는 메서드를 public으로 제공
- get() 메서드만 제공 되는 경우 read-only 필드
- 이클립스에서 자동으로 생성됨

## 정보 은닉 (information hiding)

- 외부에서 클래스 내부의 정보에 접근하지 못하도록 함
  - `private` 키워드를 활용
- `private` 변수를 외부에서 접근하게 하려면 `public` 메서드를 제공함
- 클래스 내부 데이터를 잘못 사용하는 오류를 방지할 수 있음
- `private`로 제어한 멤버 변수도 `public` 메서드가 제공되면, 접근가능하지만 변수가 `public`으로 공개되었을 때보다 `private`일 때 **각 변수에 대한 제한을 `public`메서드에서 제어할 수 있다.**

```
public void setMonth(int month) {  
  
    if ( month < 1 || month > 12) {  
        isValid = false;  
    }  
    else {  
        this.month = month;  
    }  
}
```

- 객체 지향 프로그램에서 정보 은닉이 필요한 외부에서 접근 가능한 최소한의 정보를 오픈함으로써 객체의 오류를 방지하여 클라이언트가 객체가 더 효율적으로 객체를 활용할 수 있도록 해준다.
- 예시)
  - MyDate

```
package Chapter5;  
  
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    private boolean isValid;  
  
    public void setDay(int day) {  
        this.day = day;  
    }  
  
    public int getDay() {  
        return day;  
    }  
  
    public void setMonth(int month) {  
        if ( month < 1 || month > 12) {  
            isValid = false;  
        }  
        else {  
            this.month = month;  
        }  
    }  
  
    public int getMonth() {  
        return month;  
    }  
  
    public void setYear(int year) {  
        this.year =year;  
    }  
  
    public int getYear() {  
        return year;  
    }  
  
    public void showDate() {  
        if (isValid){
```

```

        System.out.println( year + "년 " + month + "월 " + day + "일 입니다.");
    }
    else {
        System.out.println("")
    }
}
}
}

```

- MyDateTest

```

package Chapter5;

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

        date.setYear(2019);
        date.setMonth(7);
        date.setDay(30);

        date.showDate();
    }
}

```

## 11. 캡슐화 (encapsulation)

- private과 protect 키워드를 이용해 정보를 hiding하고 객체를 감싸서 외부에서 사용할 꼭 필요한 메서드 한두개만 오픈한다.

### 정보 은닉을 활용한 캡슐화

- 꼭 필요한 정보와 기능만 외부에 오픈함
- 대부분의 **멤버 변수와 메서드를 감추고** 외부에 통합된 인터페이스만은 제공하여 일관된 기능을 구현 하게 함
- 각각의 메서드나 멤버 변수를 접근함으로써 발생하는 오류를 최소화 한다.

### 레포트 만들기 예제

```

public class MakeReport {

    StringBuffer buffer = new StringBuffer();

    private String line = "=====\n";
    private String title = "  이름\t  주소 \t\t 전화번호  \n";
    private void makeHeader()
    {
        buffer.append(line);
        buffer.append(title);
        buffer.append(line);
    }

    private void generateBody()
    {
        buffer.append("James \t");
        buffer.append("Seoul Korea \t");
        buffer.append("010-2222-3333\n");

        buffer.append("Tomas \t");
        buffer.append("NewYork US \t");
        buffer.append("010-7777-0987\n");
    }
}

```

```

private void makeFooter()
{
    buffer.append(line);
}

public String getReport()
{
    makeHeader();
    generateBody();
    makeFooter();
    return buffer.toString();
}
}

```

```

public class TestReport {

    public static void main(String[] args) {

        MakeReport report = new MakeReport();
        String builder = report.getReport();

        System.out.println(builder);
    }

}

```

#### • 결과

```

<terminated> TestReprt [Java Application] C:\Users\Administrator\p2\pool
이름      주소      전화번호
=====
James    Seoul Korea    010-2222-3333
Tomas    NewYork US     010-7777-0987
=====

```

## 12. this에 대하여

### this의 역할

- 인스턴스 자신의 메모리를 가리킴
- 생성자에서 또 다른 생성자를 호출함
  - 생성자 overloading 때문에 한 클래스에 여러 개의 생성자 존재 가능
  - 이 때, 하나의 생성자에서 다른 생성자를 호출하는 경우가 있음

```

public Person() {
    this("이름 없음", 1);
}

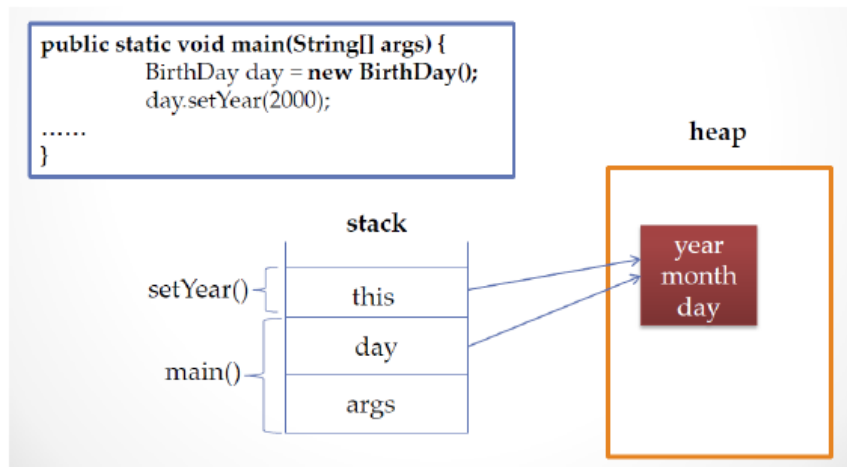
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}

```

- 인스턴스 자신의 주소(참조값)를 반환할 때 this를 반환하면 된다.

## 생성된 인스턴스 메모리의 주소를 가짐

- 클래스 내에서 참조 변수가 가지는 주소 값과 동일한 주소 값을 가지는 키워드
  - 지역 변수 메모리 공간은 stack에 잡힌다.



## 생성자에서 다른 생성자를 호출하는 this

- 클래스에 생성자가 여러개인 경우, this를 이용하여 생성자에서 다른 생성자를 호출할 수 있음
- 생성자에서 다른 생성자를 호출하는 경우, 인스턴스의 생성이 완전하지 않은 상태이므로 `this() statement` 이전에 다른 statement를 쓸 수 없음
  - this를 호출할 때 String과 Integer의 매개변수를 가지고 있으며, 해당 데이터타입을 갖는 매개변수를 찾는데, `public Person(String name, int age)` 가 호출되어야 끝날 수 있다.
    - 호출되기 전에, 다른 코드를 집어 넣으면 오류가 날 수 있다.

```
public class Person {

    String name;
    int age;

    public Person() {
        this("이름없음", 1); // 생성자에서 다른 생성자를 호출한다.
    }

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

- 생성자의 역할은 instance를 초기화하는 역할

## 자신의 주소를 반환하는 this

```
public class Person {

    String name;
    int age;

    public Person() {
        this("이름없음", 1);
    }
}
```



```

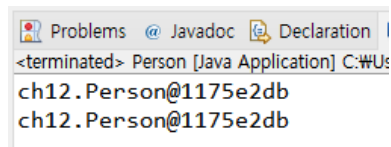
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}

public Person getPerson() {
    return this; //자신의 주소를 반환하는 this
}

public static void main(String[] args)
{
    Person p = new Person();
    p.name = "James";
    p.age = 37;

    Person p2 = p.getPerson();
    System.out.println(p);
    System.out.println(p2);
}
}

```



## 예시

- Person 클래스

```

package Chapter5.thisex;

public class Person {
    String name;
    int age;

    public Person() {
        this("이름 없음", 1);
    }

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

    public void showInfo() {
        System.out.println(name + ", " + age);
    }

    // 반환 타입은 클래스 자신이면 된다.
    public Person getSelf() {
        return this;
    }
}

```

- PersonTest

```

package Chapter5.thisex;

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

```

```

    Person personNoname = new Person();
    personNoname.showInfo(); //이름 없음, 1

    Person personLee = new Person("Lee", 20);
    personLee.showInfo(); // Lee, 20
    System.out.println(personLee); // Chapter5.thisex.Person@b1bc7ed

    Person p = personLee.getSelf();
    System.out.println(p); //Chapter5.thisex.Person@b1bc7ed
}

```

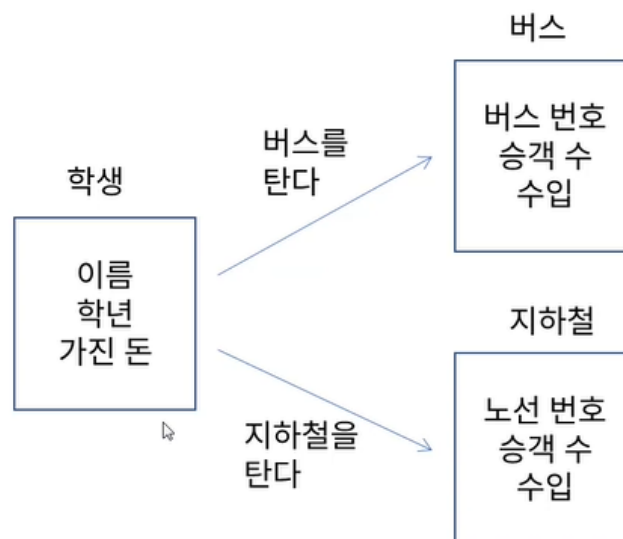
### 13. 객체 간의 협력 (collaboration)

- 객체 지향 프로그램에서 객체 간에는 협력이 이루어진다.
- 협력을 위해서는 필요한 메시지를 전송하고, 이를 처리하는 기능이 구현되어야 한다.
  - 객체가 유기적으로 연관이 되고, 정보를 주고 받는데, 이를 메시지가 전송된다고 한다.
  - 또한, 매개변수로 객체 자체가 전달되는 경우가 있다.
- 객체 지향 프로그램은 객체를 정의하고 객체간의 협력을 구현한 프로그램
- 객체 협력의 예시
  - 학생이 지하철이나 버스를 타고 학교가는 과정에서 일어나는 협력

James와 Thomas는 각각 버스와 지하철을 타고 학교에 갑니다.  
 James는 5000원을 가지고 있었고, 100번 버스를 타면서 1000원을 지불합니다.  
 Tomas는 10000원을 가지고 있었고, 초록색 지하철을 타면서 1200원을 지불합니다.

두 학생이 버스와 지하철을 타는 상황을 구현해 봅시다.

- 매개 변수로 객체 자체가 전달되는 경우 → 내가 어떤 버스를 탔는지 지정해야 한다.



- Student

```

package Chapter5.cooperation;

public class Student {
    String studentName;
    int grade;
    int money;

    public Student(String studentName, int money) {
        this.studentName = studentName;
        this.money = money;
    }

    // 학생이 어떤 버스를 탔는지 매개 변수에 버스 클래스를 넘긴다.
    public void takeBus(Bus bus) {
        bus.take(money);
        this.money -= 1000; // 나의 돈은 1000원이 줄어든다.
    }

    public void takeSubway(Subway subway) {
        subway.take(1200);
        this.money -= 1200;
    }

    public void showInfo() {
        System.out.println(studentName + "님의 남은 돈은 " + money + "원 입니다.");
    }
}

```

- Bus

```

package Chapter5.cooperation;

public class Bus {
    int busNumber;
    int passengerCount;
    int money;

    public Bus(int busNumber) {
        this.busNumber = busNumber;
    }

    public void take(int money) { //승차
        this.money += money;
        passengerCount++;
    }

    public void showBusInfo() {
        System.out.println(busNumber + "번 버스의 승객은 " + passengerCount + "명 이고, 수입은 "+ money + "입니다.");
    }
}

```

- Subway

```

package Chapter5.cooperation;

public class Subway {
    int lineNumber;
    int passengerCount;
    int money;

    public Subway(int lineNumber) {
        this.lineNumber = lineNumber;
    }

    public void take(int money) { //승차
        this.money += money;
        passengerCount++;
    }
}

```

```

    public void showSubwayInfo() {
        System.out.println(lineNumber + "노선의 지하철 승객은 " + passengerCount + "명 이고, 수입은 " + money + "입니다.");
    }
}

```

- TransTest

```

package Chapter5.cooperation;

public class TakeTransTest {
    public static void main(String[] args) {
        Student studentJ = new Student("James", 5000);
        Student studentT = new Student("Thomas", 10000);

        Bus bus100 = new Bus(100);
        Bus bus500 = new Bus(500);
        Subway subwayGreen = new Subway(2);
        Subway subwayBlue = new Subway(4);

        studentJ.takeBus(bus100);
        studentT.takeSubway(subwayGreen);

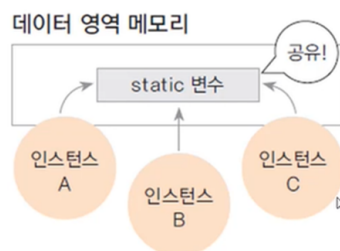
        studentJ.showInfo(); // James님의 남은 돈은 4000원 입니다.
        studentT.showInfo(); // Thomas님의 남은 돈은 8800원 입니다.

        bus100.showBusInfo(); //100번 버스의 승객은 1명 이고, 수입은 5000입니다.
        bus500.showBusInfo(); //500번 버스의 승객은 0명 이고, 수입은 0입니다.
    }
}

```

## 16. static 변수, 메서드

### static 변수



- 여러 인스턴스가 하나의 값을 공유할 필요가 있음
- 예시
  - 학생마다 새로운 학번 생성
  - 카드회사에서 카드를 새로 발급할 때마다 새로운 카드 번호를 부여
  - 회사에 사원이 입사할 때마다 새로운 사번이 필요한 경우

### static 변수 선언과 사용하기

- `static int serialNum;`

- static 변수는 처음 프로그램이 로드될 때 데이터 영역 메모리에 생성됨
  - 이렇게 메모리에 올라갔을 때를 process라고 함.
  - 메모리에 올라갈 때, 2가지 영역을 갖게 된다.
    - code 영역
    - data 영역은 상수 영역, static 영역이라고도 함
  - 인스턴스의 메모리는 heap에 저장!
- 인스턴스의 생성과 상관없이 사용할 수 있으므로 클래스 이름으로 직접 참조
  - `Student.serialNum = 100`
- 클래스 변수, 정적 변수라고도 함

## Static 변수 test 하기

- `Employee.java`

```
public class Employee {

    public static int serialNum = 1000;

    private int employeeId;
    private String employeeName;
    private String department;

    public int getEmployeeId() {
        return employeeId;
    }
    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }
    public String getEmployeeName() {
        return employeeName;
    }
    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
}
```

- `EmployeeTest.java`

```
public class EmployeeTest {

    public static void main(String[] args) {
        Employee employeeLee = new Employee();
        employeeLee.setEmployeeName("이순신");
        System.out.println(employeeLee.serialNum);

        Employee employeeKim = new Employee();
        employeeKim.setEmployeeName("김유신");
        employeeKim.serialNum++;

        System.out.println(employeeLee.serialNum);
        System.out.println(employeeKim.serialNum);
    }
}
```

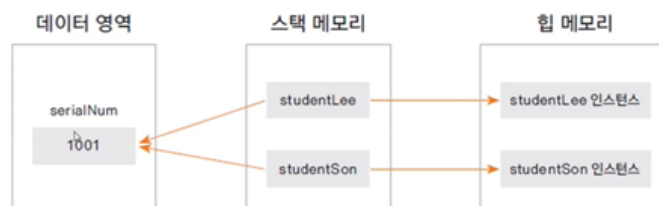
- 결과

```

Problems @ Javadoc Declaration Console
<terminated> EmployeeTest [Java Application] C:\Users\Administrat
1000
1001
1001

```

## static 변수와 인스턴스 변수



- 3가지 영역의 메모리
  - 힙 메모리/ 동적 메모리 - 필요할 때 allocate하고, 다 쓰고 나면 free시키거나 release 시킨다.
  - 스택 메모리 - function이 호출될 때 마다 function 안에서 (함수나 메소드) 지역변수가 쓰는 메모리
  - 데이터 영역 - 처음에 메모리에 로드될 때부터 자리 잡고, 공유되어서 쓰이는 영역
- 데이터 영역(static 영역)에 위치한 동일한 메모리를 참조
- static은 큰 메모리를 사용하면 안된다.
- 위 그림에서처럼, 학번을 1000번부터 1씩 추가해서 부여하고 싶다면,
  - static 변수인 serialNum을 생성하고,
  - 생성한 모든 constructor에 serialNum을 1씩 추가한다.
  - 이후, studentID에 serialNum을 부여한다.

## static 메서드

- static 변수를 위한 기능을 제공하는 static 메서드
- **static 메서드에서는 인스턴스 변수를 사용할 수 없음**
  - static 메서드는 인스턴스의 생성과 상관없이 호출될 수 있기 때문이다.
  - 인스턴스 변수는 인스턴스의 생성이 되어야 호출할 수 있으므로 인스턴스 변수는 사용할 수 없다.
  - 또한 static 메서드는 프로그램을 생성할 때 생성되므로, 큰 메모리를 차지하면 안된다.
- 클래스 이름으로 참조하여 사용하는 메서드
  - `Student.getSerialNum();`
- 클래스 메서드, 정적 메서드라고 함

## 예시

- 회사가원이 입사할 때마다 새로운 사번 부여하기
- `Employee.java`

```
package Chapter5.staticex;

public class Employee {

    public static int serialNum = 1000;

    private int employeeId;
    private String employeeName;
    private String department;

    public Employee() {
        serialNum++;
        employeeId = serialNum;
    }

    public int getEmployeeId() {
        return employeeId;
    }
    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }
    public String getEmployeeName() {
        return employeeName;
    }
    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
}
```

- `EmployeeTest.java`

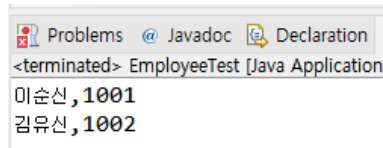
```
package Chapter5.staticex;

public class EmployeeTest {

    public static void main(String[] args) {
        Employee employeeLee = new Employee();
        employeeLee.setEmployeeName("이순신");
        System.out.println(employeeLee.serialNum);

        Employee employeeKim = new Employee();
        employeeKim.setEmployeeName("김유신");

        System.out.println(employeeLee.getEmployeeName() + "," + employeeLee.getEmployeeId());
        System.out.println(employeeKim.getEmployeeName() + "," + employeeKim.getEmployeeId());
    }
}
```



★ static 변수와 메서드는 인스턴스 변수, 메서드가 아니므로, 클래스 이름으로 직접 참조합니다.

## 17. static 메서드의 구현과 활용, 변수의 유효 범위

### static 메서드 만들기

- serialNum의 변수를 private으로 선언하고, getter/setter을 구현한다.
- 예시

• Employee.java

```
private static int serialNum = 1000;

...

public static int getSerialNum() {
    return serialNum;
}

public static void setSerialNum(int serialNum) {
    Employee.serialNum = serialNum;
}
```

- 클래스 이름으로 호출 가능 (클래스 메서드, 정적 메서드)

```
System.out.println(Employee.getSerialNum());
```

### static 메서드 (클래스 메서드)에서는 인스턴스 변수를 사용할 수 없다.

- static 메서드는 인스턴스 생성과 무관하게 클래스 이름으로 호출 될 수 있음
- 인스턴스 생성 전에 호출 될 수 있으므로 static 메서드 내부에서는 인스턴스 변수를 사용할 수 없음
- 예시

• Employee.java

```
public static void setSerialNum(int serialNum) {
    int i = 0;

    employeeName = "Lee"; //오류발생
    Employee.serialNum = serialNum;
}
```

- `int i=0` 은 함수 내부에 선언된 지역변수이다.
  - setSerialNum 함수가 끝나면 없어질 변수이다.
- `employeeName = "Lee"` 가 오류가 발생하는 이유는, 이 메서드가 불러질 시점에 employeeName이 없을 수 있기 때문이다.



- `EmployeeTest2.java`

```
public class EmployeeTest2 {

    public static void main(String[] args) {

        System.out.println(Employee.getSerialNum());
        Employee.setSerialNum(1003);
        System.out.println(Employee.getSerialNum());
    }
}
```

## 변수의 유효범위와 메모리

- 변수의 생성과 소멸 시기 중요
- 변수의 유효 범위(scope)와 생성과 소멸(life cycle)은 각 변수의 종류마다 다름
- 지역변수, 멤버 변수, 클래스 변수는 유효범위와 life cycle, 사용하는 메모리도 다름

변수 유형	선언 위치	사용 범위	메모리	생성과 소멸
지역 변수 (로컬 변수)	함수 내부에 선언	함수 내부에서만 사용	스택	함수가 호출될 때 생성되고 함수가 끝나면 소멸함
멤버 변수 (인스턴스 변수)	클래스 멤버 변수로 선언	클래스 내부에서 사용하고 <code>private</code> 이 아니면 참조 변수로 다른 클래스에서 사용 가능	힙	인스턴스가 생성될 때 힙에 생성되고, 가비지 컬렉터가 메모리를 수거할 때 소멸됨
static 변수 (클래스 변수)	static 예약어를 사용하여 클래스 내부에 선언	클래스 내부에서 사용하고 <code>private</code> 이 아니면 클래스 이름으로 다른 클래스에서 사용 가능	데이터 영역	프로그램이 처음 시작할 때 상수와 함께 데이터 영역에 생성되고 프로그램이 끝나고 메모리를 해제할 때 소멸됨

- **static 변수**는 프로그램이 메모리에 있는 동안 계속 그 영역을 차지하므로 너무 큰 메모리를 할당하는 것은 좋지 않음
- 클래스 내부의 여러 메서드에서 사용하는 변수는 멤버 변수로 선언하는 것이 좋음
- 멤버 변수가 너무 많으면 인스턴스 생성 시 쓸데없는 메모리가 할당됨
- 상황에 적절하게 변수를 사용해야 함

## 18. static 응용 - singleton pattern

### singleton pattern

- 프로그램에서 인스턴스가 단 한 개만 생성되어야 하는 경우 사용하는 디자인 패턴
  - 단 하나만 존재해야 하는 인스턴스가 있음
    - 예시. 학교에 학생이 여러명 있을 수 있지만 학교는 하나만 있어야 한다.
- static 변수, 메서드를 활용하여 구현 할 수 있음
- 생성자는 **private** 으로 선언
  - `private`이기 때문에 외부에서는 해당 생성자를 호출할 수 없다.
  - `private`으로 생성자를 선언하기 때문에 컴파일러는 생성자를 제공해주지 않는다.

```
private Company() {}
```

- 클래스 내부에 **유일한 private 인스턴스 객체 생성**

```
private static Company instance = new Company();
```

- 외부에서 유일한 객체를 참조할 수 있는 public static get() 메서드 구현

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

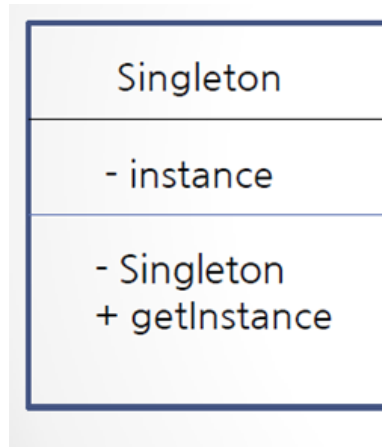
## 예시 - 싱글톤 패턴으로 회사 객체 구현하기

- Company.java

```
package Chapter5.staticex;  
  
public class Company {  
    private static Company instance = new Company();  
  
    // 내부에서 사용할 수 있는 생성자  
    private Company() {}  
  
    public static Company getInstance() {  
        if (instance == null) {  
            instance = new Company();  
        }  
        return instance;  
    }  
}
```

- CompanyTest.java

```
package Chapter5.staticex;  
  
import java.util.Calendar;  
  
public class CompanyTest {  
    public static void main(String[] args) {  
        Company company1 = Company.getInstance();  
        Company company2 = Company.getInstance();  
  
        System.out.println(company1); //Chapter5.staticex.Company@b4c966a  
        System.out.println(company2); //Chapter5.staticex.Company@b4c966a  
  
        Calendar calendar = Calendar.getInstance();  
    }  
}
```



- 마이너스 → Private이라는 뜻
- Singleton → 생성자가 Private이다.

## 20. 배열이란

### 배열이란?

- 동일한 자료형의 순차적 자료 구조
- 인덱스 연산자[]를 이용하여 빠른 참조가 가능
  - 물리적으로 연결되어 있기 때문에 **index 연산이 가능하다**.
  - index 연산이 빠르다. byte를 몇개를 건너뛰면 되는지 바로 산술적으로 계산할 수 있다.
- 물리적 위치와 논리적 위치가 동일
- 배열의 순서는 0부터 시작
- 자바에서는 객체 배열을 구현한 ArrayList를 많이 활용함
- 데이터가 중간에 비면 안된다.
  - 연속된 자료구조이므로 중간에 element 하나를 제외시킨다면, 해당 element 뒤에 있는 element들을 한 칸씩 앞으로 움직여야 한다. 마찬가지로 element를 추가해도, element를 한 칸씩 뒤로 움직여야 한다.

### 배열 선언과 초기화

- 배열 선언하기
  - `int[] arr = new int[10];`
  - `int arr[] = new int[10];`



- **fixed length**로 시작하므로 length 설정 필요

- 메모리 구조
  - int는 4바이트
- 배열 초기화하기
  - 배열은 선언과 동시에 자료형에 따라 초기화 됨 ( 정수는 0, 실수는 0.0, 객체는 null)
  - 필요에 따라 초기값을 지정할 수 있음

```
int[] numbers = new int[] {10, 20, 30}; //개수 생각해야 함

int[] numbers = {10, 20, 30};           // new int[] 생각 가능

int[] ids;
ids = new int[] {10, 20, 30};           // 선언후 배열을 생성하는 경우는 new int[] 생략할 수 없음
```

## 배열 사용하기

- [] 인덱스 연산자 활용 - 배열 요소가 저장된 메모리의 위치를 연산하여 찾아 줌
- 배열을 이용하여 합을 구하기

```
int[] arr = new int[10];
int total = 0;

for(int i=0, num=1; i< arr.length; i++, num++) {
    arr[i] = num;
}

for( int i =0; i<arr.length; i++) {
    total += arr[i];
}
System.out.println(total);
```

## 배열의 길이와 요소의 개수는 동일하지 않습니다.

- 배열을 선언하면 개수만큼 메모리가 할당되지만, 실제 요소(데이터)가 없는 경우도 있음
- 배열의 length 속성은 배열의 개수를 반환해주기 때문에 요소의 개수와는 다름
- length를 활용하여 오류가 나는 경우

```
double[] dArr = new double[5];

dArr[0] = 1.1;
dArr[1] = 2.1;
dArr[2] = 3.1;

double mttotal = 1;
for(int i = 0; i< dArr.length; i++) {
    mttotal *= dArr[i];
}

System.out.println(mttotal);
```

- 요소의 개수에 대한 변수(count)를 따로 유지

```
double[] dArr = new double[5];
int count = 0;
dArr[0] = 1.1; count++;
dArr[1] = 2.1; count++;
dArr[2] = 3.1; count++;

double mttotal = 1;
for(int i = 0; i< count; i++) {
    mttotal *= dArr[i];
}

System.out.println(mttotal);
```

## 문자 배열을 만들어 A-Z 까지 배열에 저장하고 이를 다시 출력하기

```
public class CharArrayTest {

    public static void main(String[] args) {

        char[] alphabets = new char[26];
        char ch = 'A';

        for(int i = 0; i<alphabets.length; i++) {

            alphabets[i] = ch++;
        }

        for(int i = 0; i<alphabets.length; i++) {
            System.out.println(alphabets[i] + "," + (int)alphabets[i]);
        }
    }
}
```

## 향상된 for문 사용하기

배열의 n개 요소를 0 부터 n-1까지 순차적으로 순회할 때 간단하게 사용할 수 있음

```
for( 변수 : 배열) {

}
```

```
public class CharArrayTest {

    public static void main(String[] args) {

        char[] alphabets = new char[26];
        char ch = 'A';

        for(int i = 0; i<alphabets.length; i++) {

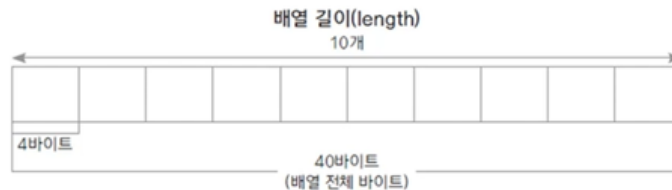
            alphabets[i] = ch++;
        }

        for(char alpha : alphabets) {
            System.out.println(alpha + "," + (int)alpha);
        }
    }
}
```

## 21. 객체 배열

### 기본 자료형 배열과 참조 자료형 배열 (객체 배열)

- 기본 자료형 배열은 선언과 동시에 배열의 크기만큼의 메모리가 할당되지만, 객체 배열의 경우엔 요소가 되는 객체의 주소가 들어갈(4바이트, 8바이트) 메모리만 할당되고(null) 각 요소 객체는 생성하여 저장해야 함
- 기본 자료형 배열
  - `int[] arr = new int[10];`



- 참조 자료형 배열
  - `Book[] library = new Book[5];`



- 배열로 선언을 한 상태에서는 null 값이 들어간다.
- 나중에 생성할 객체의 주소를 담는다.

예시

- `Book.java`

```
public class Book {  
  
    private String title;  
    private String author;  
  
    public Book() {}  
  
    public Book(String title, String author) {  
        this.title = title;  
        this.author = author;  
    }  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    public String getAuthor() {  
        return author;  
    }  
  
    public void setAuthor(String author) {  
        this.author = author;  
    }  
  
    public void showBookInfo() {  
        System.out.println(title + ", " + author);  
    }  
}
```

```
}
}
```

- `BookArrayTest.java`

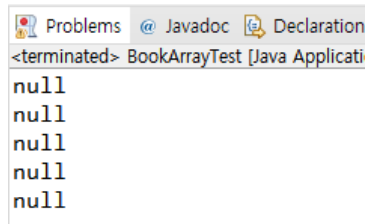
```
public class BookArrayTest {

    public static void main(String[] args) {

        Book[] library = new Book[5];

        for(int i =0; i<library.length; i++) {
            System.out.println(library[i]);
        }
    }
}
```

- 결과



```
<terminated> BookArrayTest [Java Applicati
null
null
null
null
null
```

## 객체 배열 구현

- 객체를 생성하여 각 배열의 요소로 저장하기

```
public class BookArrayTest {

    public static void main(String[] args) {

        Book[] library = new Book[5];

        library[0] = new Book("태백산맥1", "조정래");
        library[1] = new Book("태백산맥2", "조정래");
        library[2] = new Book("태백산맥3", "조정래");
        library[3] = new Book("태백산맥4", "조정래");
        library[4] = new Book("태백산맥5", "조정래");

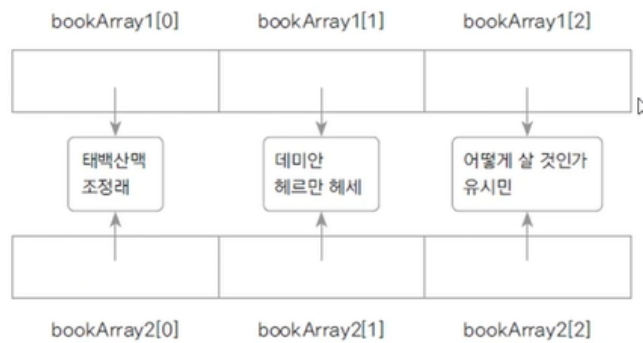
        for(int i =0; i<library.length; i++) {
            System.out.println(library[i]);
            library[i].showBookInfo();
        }
    }
}
```

## 객체 배열 복사

- `System.arraycopy(src, srcPos, dest, destPos, length)` 자바에서 제공되는 배열 복사 메서드

### 얕은 복사

- 주소 값만 복사가 된다.
- 객체 주소만 복사되어 한쪽 배열의 요소를 수정하면 같이 수정 될
- 즉, 두 배열이 같은 객체를 가리킴



- 예시

```
public class ObjectCopy {

    public static void main(String[] args) {

        Book[] library = new Book[5];
        Book[] copyLibaray = new Book[5];

        library[0] = new Book("태백산맥1", "조정래");
        library[1] = new Book("태백산맥2", "조정래");
        library[2] = new Book("태백산맥3", "조정래");
        library[3] = new Book("태백산맥4", "조정래");
        library[4] = new Book("태백산맥5", "조정래");

        System.arraycopy(library, 0, copyLibaray, 0, 5);

        System.out.println("=====copy library=====");
        for( Book book : copyLibaray ) {
            book.showBookInfo();
        }

        library[0].setTitle("나목");
        library[0].setAuthor("박완서");

        System.out.println("=====library=====");
        for( Book book : library) {
            book.showBookInfo();
        }

        System.out.println("=====copy library=====");

        for( Book book : copyLibaray) {
            book.showBookInfo();
        }
    }
}
```

- 결과



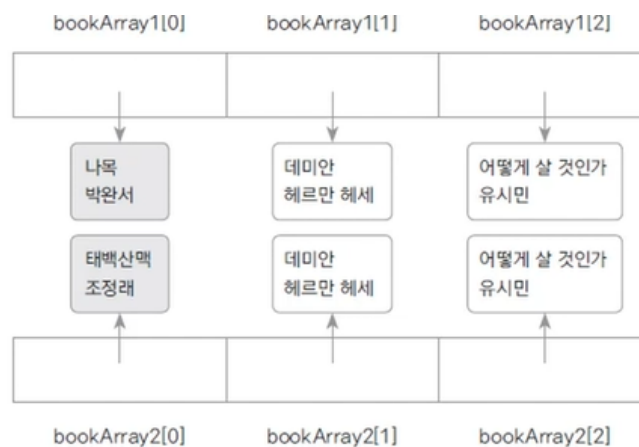
```

Problems @ Javadoc Declaration Co
<terminated> ObjectCopy [Java Application] C:\Usu
=====copy library=====
태백산맥1, 조정래
태백산맥2, 조정래
태백산맥3, 조정래
태백산맥4, 조정래
태백산맥5, 조정래
=====library=====
나목, 박완서
태백산맥2, 조정래
태백산맥3, 조정래
태백산맥4, 조정래
태백산맥5, 조정래
=====copy library=====
나목, 박완서
태백산맥2, 조정래
태백산맥3, 조정래
태백산맥4, 조정래
태백산맥5, 조정래

```

## 깊은 복사

- 일일이 객체를 새로 만들어서 값을 대입할 필요가 있다.
- 각각의 객체를 생성하여 그 객체의 값을 복사하여 배열이 서로 다른 객체를 가리키도록 함
- 이 때는 인스턴스가 구별이 되어서 bookArray1을 변경한다고 해서 bookArray2가 변경되지 않는다.



- 예시

```

public class ObjectCopy2 {

    public static void main(String[] args) {

        Book[] library = new Book[5];
        Book[] copyLibrary = new Book[5];

        library[0] = new Book("태백산맥1", "조정래");
        library[1] = new Book("태백산맥2", "조정래");
        library[2] = new Book("태백산맥3", "조정래");
        library[3] = new Book("태백산맥4", "조정래");
        library[4] = new Book("태백산맥5", "조정래");

        copyLibrary[0] = new Book();
        copyLibrary[1] = new Book();
    }
}

```

```

copyLibrary[2] = new Book();
copyLibrary[3] = new Book();
copyLibrary[4] = new Book();

for(int i = 0; i < library.length; i++) {
    copyLibrary[i].setTitle(library[i].getTitle());
    copyLibrary[i].setAuthor(library[i].getAuthor());
}

library[0].setTitle("나목");
library[0].setAuthor("박완서");

System.out.println("=====library=====");
for( Book book : library) {
    book.showBookInfo();
}

System.out.println("=====copy library=====");
for( Book book : copyLibrary) {
    book.showBookInfo();
}
}
}

```

## • 결과

```

=====library=====
나목, 박완서
태백산맥2, 조정래
태백산맥3, 조정래
태백산맥4, 조정래
태백산맥5, 조정래
=====copy library=====
태백산맥1, 조정래
태백산맥2, 조정래
태백산맥3, 조정래
태백산맥4, 조정래
태백산맥5, 조정래

```

## 향상된 for 문

- 배열 요소의 처음부터 끝까지 모든 요소를 참조할 때 편리한 반복문

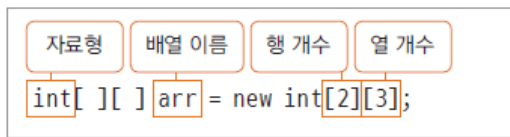
```

for (변수: 배열) {
    반복실행문
}

```

## 22. 다차원 배열

- 2차원 이상으로 구현 된 배열
- 평면 (이차원 배열) 이나 공간(삼차원 배열)을 활용한 프로그램 구현
- 이차원 배열의 예시



arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]

- `자료형[ ][ ] 배열 이름 = new 자료형[행 개수][열 개수];`
- `int[ ][ ] arr = new int[2][3];`

```
package Chapter6.array;

public class TwoDimension {
    public static void main(String[] args) {
        int[][] arr = { {1, 2, 3}, {4, 5, 6} };

        System.out.println(arr.length); // 2 행의 개수
        System.out.println(arr[0].length); //3 열의 개수
        System.out.println(arr[1].length);

        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++) {
                System.out.print(arr[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

## 23. ArrayList

### java.util 패키지에서 제공되는 ArrayList

- 기존의 배열 선언과 사용 방식은 배열의 길이를 정하고 요소의 개수가 배열의 길이보다 커지면 배열을 재할당하고 복사해야 했음
  - 배열의 요소를 추가하거나 삭제하면 다른 요소들의 이동에 대한 구현을 해야 함
- ArrayList는 객체 배열을 좀더 효율적으로 관리하기 위해 자바에서 제공해 주는 클래스
  - ArrayList 클래스란 자바에서 제공되는 객체 배열이 구현된 클래스이다.
- 이미 많은 메서드들이 최적의 알고리즘으로 구현되어 있어 각 메서드의 사용 방법만 익히면 유용하게 사용할 수 있음
  - 객체 배열을 사용하는데 필요한 여러 메서드들이 구현되어 있다.
    - 배열의 사이즈나 요소, 위치에 상관없이 메서드 사용 가능

### 주요 메서드

메서드	설명
<code>boolean add(E e)</code>	요소 하나를 배열에 추가합니다. E는 요소의 자료형을 의미합니다.
<code>int size()</code>	배열에 추가된 요소 전체 개수를 반환합니다.
<code>E get(int index)</code>	배열의 index 위치에 있는 요소 값을 반환합니다.
<code>E remove(int index)</code>	배열의 index 위치에 있는 요소 값을 제거하고 그 값을 반환합니다.
<code>boolean isEmpty()</code>	배열이 비어 있는지 확인합니다.

## ArrayList를 활용한 간단한 예제

```
import java.util.ArrayList;
import ch21.Book;

public class ArrayListTest {

    public static void main(String[] args) {

        ArrayList<Book> library = new ArrayList<Book>();

        library.add(new Book("태백산맥1", "조정래"));
        library.add(new Book("태백산맥2", "조정래"));
        library.add(new Book("태백산맥3", "조정래"));
        library.add(new Book("태백산맥4", "조정래"));
        library.add(new Book("태백산맥5", "조정래"));

        for(int i =0; i<library.size(); i++) {
            library.get(i).showBookInfo();
        }
    }
}
```

## 24. ArrayList를 활용한 간단한 성적 산출 프로그램

### 예제 시나리오

1001학번 Lee와 1002학번 Kim, 두 학생이 있습니다.  
 Lee 학생은 국어와 수학 2과목을 수강했고, Kim 학생은 국어, 수학, 영어 3 과목을 수강하였습니다.  
 Lee 학생은 국어 100점, 수학 50점입니다.  
 Kim 학생은 국어 70점, 수학 85점, 영어 100점입니다.  
 Student와 Subject 클래스를 만들고 ArrayList를 활용하여 두 학생의 과목 성적과 총점을 출력하세요

### Student 클래스

```
import java.util.ArrayList;

public class Student {

    int studentID;
    String studentName;
    ArrayList<Subject> subjectList;

    public Student(int studentID, String studentName){
        this.studentID = studentID;
        this.studentName = studentName;

        subjectList = new ArrayList<Subject>();
    }
}
```

```

public void addSubject(String name, int score){
    Subject subject = new Subject();

    subject.setName(name);
    subject.setScorePoint(score);
    subjectList.add(subject);
}

public void showStudentInfo()
{
    int total = 0;

    for(Subject s : subjectList){

        total += s.getScorePoint();
        System.out.println("학생 " + studentName + "의 " + s.getName() + " 과목 성적은 " +
            s.getScorePoint() + "입니다.");
    }

    System.out.println("학생 " + studentName + "의 총점은 " + total + " 입니다.");
}
}

```

## Subject 클래스

```

public class Subject {

    private String name;
    private int scorePoint;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getScorePoint() {
        return scorePoint;
    }
    public void setScorePoint(int scorePoint) {
        this.scorePoint = scorePoint;
    }
}

```

## 실행하기

```

public class StudentTest {

    public static void main(String[] args) {
        Student studentLee = new Student(1001, "Lee");

        studentLee.addSubject("국어", 100);
        studentLee.addSubject("수학", 50);

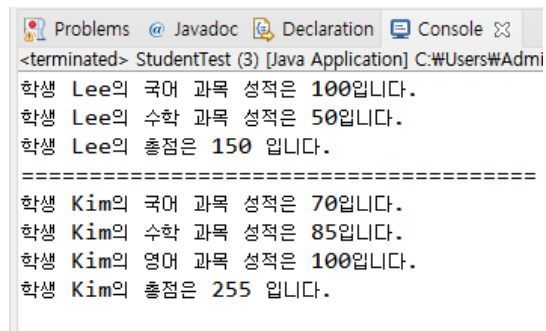
        Student studentKim = new Student(1002, "Kim");

        studentKim.addSubject("국어", 70);
        studentKim.addSubject("수학", 85);
        studentKim.addSubject("영어", 100);

        studentLee.showStudentInfo();
        System.out.println("=====");
        studentKim.showStudentInfo();
    }
}

```

## 결과



```
Problems @ Javadoc Declaration Console
<terminated> StudentTest (3) [Java Application] C:\Users\Admi
학생 Lee의 국어 과목 성적은 100입니다.
학생 Lee의 수학 과목 성적은 50입니다.
학생 Lee의 총점은 150 입니다.
=====
학생 Kim의 국어 과목 성적은 70입니다.
학생 Kim의 수학 과목 성적은 85입니다.
학생 Kim의 영어 과목 성적은 100입니다.
학생 Kim의 총점은 255 입니다.
```