

클래스, 객체

객체 메소드, private, 오버로딩, this(), static

객체 메소드, instance method, method

```
public class Student {  
    String    name;  
    int       score;  
    public Student(String name, int score) {  
        this.name=name;  
        this.score=score;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student s=new Student("김철수", 99);  
        char grade=getGrade(s);  
        System.out.println(grade);  
    }  
    private static char getGrade(Student s) {  
        if(s.score>=70) return 'P';  
        return 'F';  
    }  
}
```

- Student 클래스의 객체 메소드 아님
- Test 클래스의 static 메소드임

```
public class Student {  
    String    name;  
    int       score;  
    public Student(String name, int score) {  
        this.name=name;  
        this.score=score;  
    }  
    public char getGrade() {  
        if(this.score>=70) return 'P';  
        return 'F';  
    }  
}
```

Student 클래스의 객체 메소드

- 객체에 적용될
절차를 함수
형식으로 해당
객체의 클래스
내부에 기술한 코드

```
public class Test {  
    public static void main(String[] args) {  
        Student s=new Student("김철수", 99);  
        char grade=s.getGrade();  
        System.out.println(grade);  
  
        Student t=new Student("홍길동", 60);  
        System.out.println(t.getGrade());  
    }  
}
```

객체 s에 적용

객체 t에 적용

객체 메소드: getter, setter

```
public class Student {
    String name;
    int    score;
    public Student(String name, int score) {
        this.name=name;
        this.score=score;
    }
    @Override
    public String toString() {
        return "이름:"+name+", 점수:"+score;
    }
    public String getName() {
        return name;
    }
    public int getScore() {
        return score;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setScore(int score) {
        this.score = score;
    }
}
```

객체 메소드

- 객체에 적용될 절차를 함수 형식으로 해당 객체의 클래스 내부에 기술한 코드

객체 메소드 예

- **getter** 메소드: 객체의 필드 값을 반환하는 메소드
 - ✓ getName(), getScore()
- **setter** 메소드: 객체의 필드 값을 변경하는 메소드
 - ✓ setName(), setScore()

```
public class Test {
    public static void main(String[] args) {
        Student    s=new Student("이영희", 95);
        System.out.println(s);

        String name=s.getName();
        int    score=s.getScore();
        s.setName(name+"(오)");
        s.setScore(score+2);

        System.out.println(s);
    }
}
```

객체 메소드 실습 A

```
public class Player {  
    String    id;  
    int    record1, record2, record3;  
    public Player(String id, int record1, int record2, int record3) {  
        this.id=id;  
        this.record1=record1;  
        this.record2=record2;  
        this.record3=record3;  
    }  
}
```

실습: Player 클래스에 객체 메소드 `getAverage()`를 정의하여 아래 코드가 p에 저장된 선수의 평균을 출력하도록 하시오.

```
public class Test {  
    public static void main(String[] args) {  
        Player p=new Player("P001", 210, 205, 220);  
        System.out.println(p.getAverage());  
    }  
}
```

객체 메소드 실습 B

```
public class Movie {  
    String    title;  
    String    releaseDate;  
    public Movie(String title, String releaseDate) {  
        this.title=title;  
        this.releaseDate=releaseDate;  
    }  
}
```

실습: 아래 코드는 m에 저장된 영화의 개봉일을 출력한다. 이 코드가 정상 동작하도록 Movie 클래스를 수정하시오. (Movie 클래스의 releaseDate에 저장되는 값은 YYYY-MM-DD의 형식이라고 가정)

```
public class Test {  
    public static void main(String[] args) {  
        Movie  m=new Movie("Mission: Impossible – Rogue Nation", "2015-07-23");  
        int year=m.getReleaseYear();  
        int month=m.getReleaseMonth();  
        int day=m.getReleaseDay();  
        System.out.println("개봉일:"+year+"년"+month+"월"+day+"일");  
    }  
}
```

객체 메소드 실습 C

실습

- 다음은 코드와 그 실행결과를 보인 것이다. 아래 코드가 정상 동작하도록 User 클래스를 완성하시오.

```
public class User {  
    String id;  
    String email;  
    public User(String id, String email) {  
        this.id=id;  
        this.email=email;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        User u=new User("gdhong", "gdhong@ks.ac.kr");  
        System.out.println(u);  
        String id=u.getId();  
        u.setId(id.toUpperCase());  
        System.out.println(u);  
        String email=u.getEmail();  
        u.setEmail(email.toUpperCase());  
        System.out.println(u);  
    }  
}
```

실행결과

```
User id=gdhong, email=gdhong@ks.ac.kr  
User id=GDHONG, email=gdhong@ks.ac.kr  
User id=GDHONG, email=GDHONG@KS.AC.KR
```

접근 제어자 (access modifier)

참조: <https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>

접근 제어자

- 한 클래스의 필드 및 메소드를 다른 클래스에서 접근(사용하거나 호출) 가능하게 할지 여부를 결정하는 키워드

접근 제어자	클래스	패키지	하위클래스	외부
public	O	O	O	O
protected	O	O	O	X
no modifier	O	O	X	X
private	O	X	X	X

- private → 클래스 내부에서만 접근 가능, 클래스 외부에서는 접근 불가
- no modifier (default) → 클래스 외부더라도 같은 패키지 내 클래스에서까지 접근 가능
- protected → 패키지 외부더라도 하위 클래스에서까지 접근 가능
- public → 클래스 내부 및 외부 어디에서든 접근 가능

정보은닉(Information hiding)

정보은닉

- 객체의 정보에 대한 외부로부터의 직접 접근을 허용하지 않는 것
 - ◆ 이를 통해 해당 클래스의 설계 변경 독립성과 데이터 일관성을 보장할 수 있게 됨
- 아래 코드는 정보은닉 원칙을 구현하지 못하고 있음
 - ◆ Member 클래스 내부의 필드에 대해 외부 프로그램이 직접 접근하여 그 값을 변경하고 있음
 - ◆ 직접 접근 대신 getter, setter 등의 메소드를 통한 간접 접근을 제공할 필요가 있음

```
public class Member {  
    String    id;  
    String    name;  
    public Member(String id, String name) {  
        this.id=id;  
        this.name=name;  
    }  
    @Override  
    public String toString() {  
        return id+","+name;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Member member;  
        member=new Member("M-0123", "브라운");  
        member.name="제임스 브라운";  
        System.out.println(member);  
    }  
}
```

- 필드 name에 대한 직접 접근을 허용하고 있음
- 이 경우 Member의 이름을 firstname, lastname으로 분리 저장하도록 구현 방법이 변경될 경우 Member 클래스를 사용하는 다른 코드들의 확인 및 변경 필요

정보은닉: private, getter, setter

- ✚ 정보은닉을 보장하는 수단으로 private, getter, setter 사용
 - private → 다른 모든 클래스로부터의 접근 불허 (자신 클래스 내부로부터의 접근만 허용)
 - getter → 객체의 속성 값을 반환하는 메소드
 - setter → 객체의 속성 값을 변경하는 메소드

```
public class Member {  
    private String id;  
    private String name;  
    public Member(String id, String name) {  
        this.id=id;  
        this.name=name;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    @Override  
    public String toString() {  
        return id+","+name;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Member member;  
        member=new Member("M-0123", "이영희");  
        member.setName("김철수");  
        System.out.println(member.getName());  
    }  
}
```

private 실습 A

✚ 회원 클래스 Member를 다음 조건에 따라 작성하시오

- 필드 변수명(설명, 자료형): email(메일주소, String), joinDate(가입일-8자리, String)
 - ◆ 클래스 외부로부터의 직접 접근 불허 설정할 것
- 메일주소와 가입일을 파라미터로 전달받아 대응하는 필드에 대입하는 생성자
- 회원의 email을 반환하는 public 메소드 getEmail()
- 새로운 메일주소를 파라미터로 전달받아 객체의 기존 메일주소를 변경하는 메소드 setEmail()
- 회원의 joinDate를 8자리 숫자 문자열로 반환하는 public 메소드 getJoinDate()
- 회원의 가입년도를 4자리 숫자 문자열로 반환하는 public 메소드 getJoinYear()

✚ 클래스 Test의 main() 내 다음 절차를 코딩하시오

- Member 객체(메일주소 yhkim@ks.ac.kr, 가입일 20190214) 생성
- Member 객체의 메일주소와 가입일을 출력
- Member 객체의 메일주소를 yhkim@cs.ks.ac.kr로 변경
- Member 객체의 메일주소와 가입일을 출력
- Member 객체의 가입년도를 출력

private 실습 B

날짜 클래스 DateInfo를 다음 조건에 따라 작성하시오

- 필드변수명(설명,자료형): year(연-4자리String), month(월-2자리, String), day(일-2자리,String)
- 연,월,일을 파라미터로 전달받아 대응하는 필드에 대입하는 생성자
- 연,월,일을 8자리 숫자 문자열로 반환하는 toString()

회원 클래스 Member를 다음 조건에 따라 작성하시오

- 필드변수명(설명,자료형): email(메일주소,String), joinDate(가입일-8자리,DateInfo)
 - ◆ 클래스 외부로부터의 직접 접근 불허 설정
- 메일주소와 가입일을 파라미터로 전달받아 대응하는 필드에 대입하는 생성자
- 회원의 email을 반환하는 public 메소드 getEmail()
- 새로운 메일주소를 파라미터로 전달받아 객체의 기존 메일주소를 변경하는 메소드 setEmail()
- 회원의 joinDate를 8자리 숫자 문자열로 반환하는 public 메소드 getJoinDate()
- 회원의 가입년도를 4자리 숫자 문자열로 반환하는 public 메소드 getJoinYear()

클래스 Test의 main() 내 다음 절차를 코딩하시오

- Member 객체(메일주소 yhkim@ks.ac.kr, 가입일 20190214) 생성
- Member 객체의 메일주소와 가입일을 출력
- Member 객체의 메일주소를 yhkim@cs.ks.ac.kr로 변경
- Member 객체의 메일주소와 가입일을 출력
- Member 객체의 가입년도를 출력

확인

- Private 실습 B에서의 Test 클래스 코드는 Private 실습 A에서의 Test 클래스 코드와 동일함
- 즉, Member 클래스의 구현 방식이 변경되었음에도 Member 클래스를 사용하는 Test 클래스의 코드는 변경 필요 없음

오버로딩(overloading)

오버로딩(overloading)

- 파라미터의 개수나 타입이 다르면서 이름이 같은 생성자 혹은 메소드를 정의하는 것 (리턴 타입만 다른 경우는 오버로딩 아님, 오류 발생)

```
public class Member {  
    String    id;  
    char      gender;  
    public Member(String id) {  
        this.id=id;  
    }  
    public Member(String id, char gender) {  
        this.id=id;  
        this.gender=gender;  
    }  
    @Override  
    public String toString() {  
        return id+","+gender;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Member    member1, member2;  
        member1=new Member("M-123");  
        member2=new Member("M-124",'여');  
  
        System.out.println(member1);  
        System.out.println(member2);  
    }  
}
```

오버로딩(overloading)

```
public class Customer {
    String email;
    double point;
    public Customer(String email) {
        this.email=email;
    }
    public void addToPoint(int point) {
        this.point+=point;
    }
    public void addToPoint(double point) {
        this.point+=point;
    }
    public void addToPoint(int p1, int p2) {
        this.point+=p1+p2;
    }
    public void addToPoint(int[] v) {
        for (int i = 0; i < v.length; i++) {
            this.point+=v[i];
        }
    }
    @Override
    public String toString() {
        return email+","+point;
    }
}
```



오버로딩(overloading)

- 파라미터의 개수나 타입이 다르면서 이름이 같은 생성자 혹은 메소드를 정의하는 것 (리턴 타입만 다른 경우는 오버로딩 아님, 오류 발생)

```
public class Test {
    public static void main(String[] args) {
        Customer c=new Customer("goodmorning@ks.ac.kr");
        c.addToPoint(10);
        c.addToPoint(45.3);
        c.addToPoint(10, 20);
        int v[]={2,7,1};
        c.addToPoint(v);
        System.out.println(c);
    }
}
```

오버로딩(overloading)

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(100);  
        System.out.println(3.14);  
        System.out.println('봄');  
        System.out.println(true);  
        System.out.println("대한민국");  
    }  
}
```

클래스 작성: this()

this()

- 한 클래스 내의 다른 생성자를 호출할 때 사용
 - ◆ 아래 코드에서 this(id, name);을 Member(id, name);로 작성하면 오류 발생

```
public class Member {  
    String    id;  
    String    name;  
    char      gender;  
    public Member(String id, String name) {  
        this.id=id;  
        this.name=name;  
    }  
    public Member(String id, String name, char gender) {  
        this.id=id;  
        this.name=name;  
        this.gender=gender;  
    }  
    @Override  
    public String toString() {  
        return id+","+name+","+gender;  
    }  
}
```



```
public class Member {  
    String    id;  
    String    name;  
    char      gender;  
    public Member(String id, String name) {  
        this.id=id;  
        this.name=name;  
    }  
    public Member(String id, String name, char gender) {  
        this(id, name);  
        this.gender=gender;  
    }  
    @Override  
    public String toString() {  
        return id+","+name+","+gender;  
    }  
}
```

다른 생성자 호출문은 생성자 내 첫 행에 위치
참조: <https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html>

오버로딩 실습 A

```
public class Employee {  
    String id;  
    double monthlyPayBase=250; // 기본 월 급여액 250만원  
    public Employee(String id) {  
        this.id=id;  
    }  
    public double getMonthlyPay() {  
        return monthlyPayBase;  
    }  
}
```

실습

- 아래 Test 클래스가 정상 동작하도록 Employee 클래스를 수정하시오
- 생성자 오버로딩 시 this()를 사용하시오

```
public class Test {  
    public static void main(String[] args) {  
        Employee e1=new Employee("E-001"); // 일반 직원  
        Employee e2=new Employee("E-002", 100); // 인턴 직원, 기본 월 급여액 100만원  
        System.out.println(e1.getMonthlyPay()); // 월 급여액 출력  
        System.out.println(e2.getMonthlyPay()); // 월 급여액 출력  
    }  
}
```


오버로딩 실습 B

```
public class Employee {  
    String id;  
    double monthlyPayBase=250; // 기본 월 급여액 250만원  
    public Employee(String id) {  
        this.id=id;  
    }  
    public double getMonthlyPay() { // 10% 기본 수당 지급  
        double pay=monthlyPayBase+monthlyPayBase*0.1;  
        return pay;  
    }  
}
```

실습

- 아래 Test 클래스가 정상 동작하도록 Employee 클래스를 수정하시오
- 생성자 오버로딩 시 this()를 사용하시오
- Test 클래스 실행 결과
275.0
110.0
247.5

```
public class Test {  
    public static void main(String[] args) {  
        Employee e1=new Employee("E-001"); // 일반 직원  
        Employee e2=new Employee("E-002", 100); // 인턴 직원, 기본 월 급여액 100만원  
        Employee e3=new Employee("E-003"); // 일반 직원  
        System.out.println(e1.getMonthlyPay()); // 월 급여액 출력  
        System.out.println(e2.getMonthlyPay()); // 월 급여액 출력  
        int absentDays=3; // 결근 일수 3일  
        System.out.println(e3.getMonthlyPay(absentDays)); // 결근일 반영 월 급여액 출력 (월 30일 근무 기준)  
    }  
}
```

인스턴스 변수, 인스턴스 메소드

- 인스턴스 변수(instance variable), (non-static) 필드
 - 특정 객체 고유의 값을 보관, 객체 생성 이후 사용 가능
- 인스턴스 메소드 (instance method), 객체 메소드
 - 특정 객체에 적용되는 메소드, 객체 생성 이후 사용 가능
- 인스턴스 변수 및 인스턴스 메소드는 객체 생성 후 객체 참조 값을 통해 접근

```
public class Robot {  
    String id; // 필드, 인스턴스 변수  
    public Robot(String id) {  
        this.id=id;  
    }  
    // 인스턴스 메소드, 객체 메소드  
    public void changeld(String id) {  
        this.id=id;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Robot robot1=new Robot("R001");  
        robot1.id="R100";           // 객체 필드 값 변경  
        robot1.changeld("R111");    // 인스턴스 메소드 호출  
        // Robot.id="R200";         // 오류  
        // Robot.changeld("R222");  // 오류  
    }  
}
```

클래스 변수, 클래스 메소드

- 클래스 변수(class variable, static field)
 - 클래스 관련 데이터를 보관, 객체 생성 이전 사용 가능, 변수 선언 시 **static 키워드** 부착
- 클래스 메소드 (class method, static method)
 - 클래스 수준의 절차를 명시, 객체 생성 이전 사용 가능, 메소드 정의 시 **static 키워드** 부착
- 클래스 변수 및 클래스 메소드는 객체 참조 값이 아닌 클래스명을 통해 접근
 - 클래스 변수 및 클래스 메소드는 객체 참조 값을 통해서도 접근 가능하나 이는 가독성 측면에서 지양

```
public class Robot {  
    String id;  
    static int numRobot; // 클래스 변수, static 필드  
    public Robot(String id) {  
        this.id=id;  
        numRobot++;  
    }  
    public void changeId(String id) {  
        this.id=id;  
    }  
    public static int getNumRobot() { // 클래스 메소드  
        return numRobot;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Robot robot1=new Robot("R001");  
        Robot robot2=new Robot("R002");  
  
        System.out.println(Robot.numRobot);  
        // System.out.println(robot1.numRobot);  
        // System.out.println(robot2.numRobot);  
  
        System.out.println(Robot.getNumRobot());  
        // System.out.println(robot1.getNumRobot());  
        // System.out.println(robot2.getNumRobot());  
    }  
}
```

static 필드 (static field)

Non-static 필드
인스턴스(instance) 변수

```
public class Employee {  
    String id;  
    double monthlyPayBase=250; // 월기본급  
    public Employee(String id) {  
        this.id=id;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Employee e1=new Employee("E-001");  
        Employee e2=new Employee("E-002");  
        System.out.println(e1.monthlyPayBase);  
        System.out.println(e2.monthlyPayBase);  
    }  
}
```

- 클래스 Employee의 monthlyPayBase는 객체 공통 데이터임 (이 회사의 월기본급은 직원마다 다르지 않다고 가정)
- 이 경우 개별 직원 객체를 통해 월기본급을 확인하는 것은 비효율적

static 필드
클래스 변수

```
public class Employee {  
    String id;  
    static double monthlyPayBase=250;  
    public Employee(String id) {  
        this.id=id;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(Employee.monthlyPayBase);  
    }  
}
```

- 클래스 Employee의 monthlyPayBase를 static 필드로 정의
- static 필드는 객체를 생성하지 않고 클래스를 통해 접근 가능

static 메소드 (static method)

Non-static 메소드
인스턴스(instance) 메소드

```
public class Calculator {  
    public int sum(int x, int y) {  
        return x+y;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Calculator c1=new Calculator();  
        int v1=c1.sum(11,22);  
        System.out.println(v1);  
        Calculator c2=new Calculator();  
        int v2=c2.sum(33,44);  
        System.out.println(v2);  
    }  
}
```

- 클래스 Calculator에는 객체 고유의 자료 (서로 다른 객체마다 서로 다른 자료)를 관리하지 않음
- 이 경우 메소드 sum()을 인스턴스 메소드로 정의하는 것은 비효율적 (sum() 호출을 위해 Calculator 객체를 생성해야 함)

static 메소드
클래스 메소드

```
public class Calculator {  
    public static int sum(int x, int y) {  
        return x+y;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        int v1=Calculator.sum(11,22);  
        System.out.println(v1);  
  
        int v2=Calculator.sum(33,44);  
        System.out.println(v2);  
    }  
}
```

- 클래스 Calculator의 sum() 메소드를 static 메소드로 정의 (static 키워드 부착)
- static 메소드는 객체를 생성하지 않고 클래스를 통해 호출 가능

static 필드, static 메소드 사용 예

```
public class Test {  
    public static void main(String[] args) {  
        int  max=Math.max(43, 76);  
        System.out.println(max); // 76  
        int  v=Integer.parseInt("123")+5;  
        System.out.println(v); // 128  
        double  w=Double.parseDouble("3.14")+0.05;  
        System.out.println(w); // 3.19  
        long  curTime=System.currentTimeMillis();  
        System.out.println(curTime);  
        System.out.println(Math.PI); // 원주율  
        System.out.println(Color.RED);  
        double  x=Math.pow(2, 10); // 2의 10승  
        System.out.println(x); // 1024.0  
    }  
}
```

static 실습 A

- 다음 Test 클래스가 정상 동작하도록 클래스 Calc를 작성하시오
- Calc.max()는 파라미터로 전달받은 두 정수 중 큰 값을 반환한다
- Calc.areaOfCircle()은 원의 반지름을 파라미터로 전달받아 원의 면적을 반환한다
- 원의 면적 계산을 위해 static field 변수 PI에 저장된 원주율값(3.14159로 가정)을 사용하시오

```
public class Test {  
    public static void main(String[] args) {  
        int max=Calc.max(12,34);  
        System.out.println(max);  
        System.out.println("원주율="+Calc.PI);  
        double radius=2.0;  
        double area=Calc.areaOfCircle(radius);  
        System.out.println("원의 면적="+area);  
    }  
}
```

static 실습 B

- 다음 Test 클래스가 정상 동작하도록 클래스 Soldier를 작성하시오
- Soldier.getCount()는 현재까지 생성된 Soldier 객체의 총 개수를 정수로 반환한다
- decreaseCount()는 Soldier 객체의 수를 1 감소시킨다

```
public class Test {  
    public static void main(String[] args) {  
        Soldier s[]={new Soldier(), new Soldier(), new Soldier(), new Soldier()};  
        System.out.println(Soldier.getCount()); // 4  
        s[3]=null;  
        Soldier.decreaseCount();  
        System.out.println(Soldier.getCount()); // 3  
    }  
}
```


static 실습 C

- 아래 코드는 Soldier 배열에 저장된 Soldier 객체들을 출력하는 것으로 실행 결과는 다음과 같다. 아래 코드를 완성하시오.

- **실행결과**

병사 고유번호: Soldier-1

병사 고유번호: Soldier-2

병사 고유번호: Soldier-3

병사 고유번호: Soldier-4

```
public class Test {  
    public static void main(String[] args) {  
        Soldier s[]={new Soldier(), new Soldier(), new Soldier(), new Soldier()};  
        for (int i = 0; i < s.length; i++) {  
            System.out.println(s[i]);  
        }  
    }  
}
```

static 실습 D

- 다음 Test 클래스가 정상 동작하도록 클래스 Calc를 작성하시오
- Calc.sum(int, int)은 파라미터로 전달받은 두 정수의 합을 정수로 반환한다
- Calc.sum(double, double)은 파라미터로 전달받은 두 실수의 합을 실수로 반환한다
- Calc.sum(int n[])은 파라미터로 전달받은 배열 내 정수들의 합을 정수로 반환한다
- Calc.sum(String s[])은 파라미터로 전달받은 배열 내 정수문자열들의 합을 정수로 반환한다


↑
메소드 오버로딩

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(Calc.sum(11,22)); // 33  
        System.out.println(Calc.sum(1.5,2.6)); // 4.1  
        int n[]={1,2,3,4,5};  
        System.out.println(Calc.sum(n)); // 15  
        String s[]{"1","2","3","4","5"};  
        System.out.println(Calc.sum(s)); // 15  
    }  
}
```

References

 <http://docs.oracle.com/javase/7/docs/api/>

 <https://docs.oracle.com/javase/tutorial/java/>

 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.

 남궁성. 자바의 정석. 도우출판.

 황기태, 김효수 (2015). 명품 Java Programming. 생능출판사.