

5/8 스터디노트

📌 지난 시간 복습 (Generic)

● 클래스 생성시 타입 제한하기

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    List list = new ArrayList();  
    list.add("홍길동");  
    list.add("임꺽정");  
    String name = list.get(0);  
    String name = (String)list.get(0);  
    System.out.println(((String)list.get(0)).length());  
    System.out.println(name);  
  
    list.add(1);  
    String name2 = (String)list.get(2); // 컴파일 에러가 발생하지 않는다. 하지만 런타임 에러가 발생한다.  
        I  
    System.out.println(((String)list.get(1)).length()); // 컴파일 에러가 발생하지 않는다. 하지만 런타임 에러가 발생한다.  
    System.out.println(name);  
}
```

→ list 에 3 번째 값 (인덱스 2) 에 String 타입이 아니라 Integer 타입인 1 을 add 했기 때문에 list.get(2) 를 String 으로 가져올 수 없음

<제너릭 적용하기>

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    List<String> list = new ArrayList<String>();  
    list.add("홍길동");  
    String name = list.get(0);  
    String name = (String)list.get(0);  
    System.out.println(((String)list.get(0)).length());  
    System.out.println(name);  
  
    list.add(1); // 런타임에러가 발생하기 전에 보다 컴파일 에러로 문제를 빨리 해결할 수 있다.  
    String name2 = (String)list.get(1); // 컴파일 에러가 발생하지 않는다. 하지만 런타임 에러가 발생한다.  
  
    System.out.println(((String)list.get(1)).length()); // 컴파일 에러가 발생하지 않는다. 하지만 런타임 에러가 발생한다.  
    System.out.println(name);  
}
```

→ list 를 만들 때 < > 로 타입을 제한함으로써, String 타입이 아닌 값을 넣으면 컴파일 에러 (빨간 줄)이 나타나면서 실행전에 실수를 발견가능함

※ 제너릭을 사용하는 이유

- 컴파일 시 강한 타입 체크 가능
- 타입 변환 코드 제거

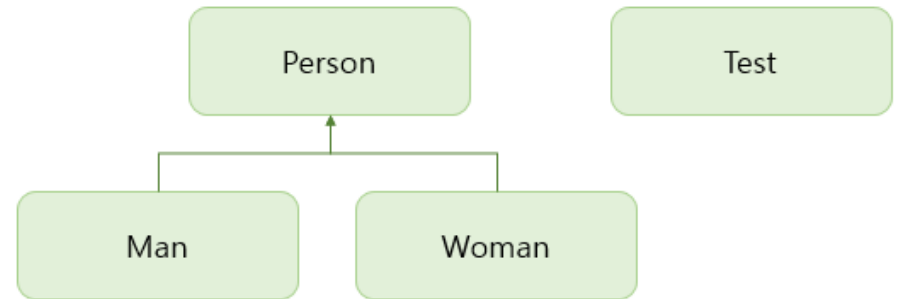
※ 제너릭 타입을 지정하지 않으면

Object 클래스 타입으로 정의됨.

- P410 클래스 생성시 타입을 특정 클래스의 부모 혹은 자식 클래스의 자료형으로 제한하기

- ✓ <?> : 모든 자료형 가능
- ✓ <? super Obj> : Obj 클래스와 그 상위 클래스의 자료형으로 제한
- ✓ <? extends Obj> : Obj 클래스와 그 하위 클래스의 자료형으로 제한
- ✓ 선생님 예제

```
public class Person {  
    String name;  
  
    public Person() {  
    }  
    Person(String name){  
        this.name = name;  
    }  
    public String toString(){  
        return name;  
    }  
}
```



```
public class Man extends Person {  
    Man(String name) {  
        this.name = name;  
    }  
    public String toString(){  
        return name.toString();  
    }  
}
```

```
public class Woman extends Person {  
    Woman(String name) {  
        this.name = name;  
    }  
    public String toString(){  
        return name.toString();  
    }  
}
```

```
public class Test {  
    String name;  
}
```

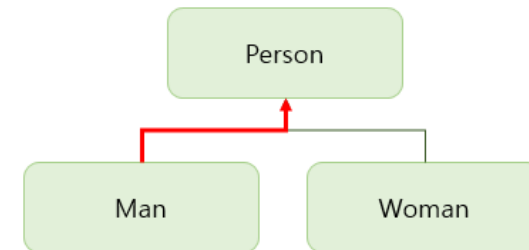
<? super [클래스]> : [클래스]와 그 상위클래스

```
import java.util.*;
public class WildSuper {
    public static void main(String[] args) {
        List<Person> listP = new ArrayList<Person>();
        listP.add(new Person( name: "사람"));
        listP.add(new Person( name: "인간"));
        printData(listP); //-> 출력결과 : 사람 인간

        List<Man> listM = new ArrayList<Man>();
        listM.add(new Man( name: "하현우"));
        listM.add(new Man( name: "박효신"));
        printData(listM); //-> 출력결과 : 하현우 박효신

        List<Woman> listW = new ArrayList<Woman>();
        listW.add(new Woman( name: "백예린"));
        listW.add(new Woman( name: "박정현"));
        printData(listW); //-> 출력결과 : 에러.
        // printData 메서드는 <? super Man> 으로 매개변수 자료형을 제한하였음
        // Woman은 Man 클래스의 상위 클래스가 아니므로 에러
    }

    public static void printData(List<? super Man> list){
        for(Object obj : list){
            System.out.println(obj);
        }
    }
}
```

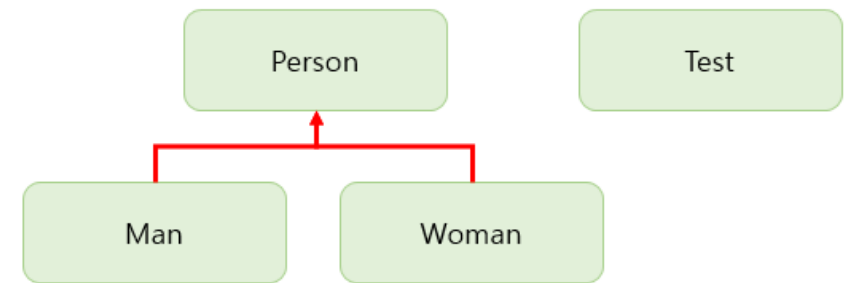


-> 상속관계가 위 그림과 같으므로

printData(List<? **super Man**> list) 메서드는
Man과 Man의 상위클래스인 Person 클래스의 자료형만
매개변수로 받을 수 있음

<? extends [클래스]> : [클래스]와 그 하위 클래스

```
public class WildExtends {  
    public static void main(String[] args) {  
        List<Person> listP = new ArrayList<Person>();  
        listP.add(new Person( name: "사람"));  
        listP.add(new Person( name: "인간"));  
        printData(listP); //-> 출력결과 : 사람 인간  
  
        List<Man> listM = new ArrayList<Man>();  
        listM.add(new Man( name: "하현우"));  
        listM.add(new Man( name: "박효신"));  
        printData(listM); //-> 출력결과 : 하현우 박효신  
  
        List<Woman> listW = new ArrayList<Woman>();  
        listW.add(new Woman( name: "백예린"));  
        listW.add(new Woman( name: "박정현"));  
        printData(listW); //-> 출력결과 : 백예린 박정현  
  
        List<Test> listT = new ArrayList<Test>();  
        listT.add(new Test());  
        printData(listT); //-> 출력결과 : 에러.  
        //printData 메서드의 매개변수를 Person과 그 하위클래스로 제한하였음.  
        //Test 클래스는 Person의 하위클래스가 아니므로 에러발생  
    }  
  
    public static void printData(List<? extends Person> list){  
        for(Object obj : list){  
            System.out.println(obj);  
        }  
    }  
}
```



-> 상속관계가 위 그림과 같으므로

printData(List<? extends Person> list) 메서드는 Person과 그 하위클래스인 Man과 Woman 클래스의 자료형만 매개변수로 받을 수 있음

- P407 제너릭 타입을 두개 사용하는 방법

```
class Generic2<K,V>{
```

```
    K name;
```

```
    V id;
```

```
    void set(K name, V id){
```

```
        this.name = name;
```

```
        this.id = id;
```

```
    }
```

```
    public K getName() { return name; }
```

```
    public V getId() { return id; }
```

```
}
```

```
public class GenericEx3 {
```

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

```
        Generic2<String, Integer> gen1 = new Generic2<String, Integer>();
```

```
        gen1.set("홍길동", 1111);
```

```
        gen1.set("이순신", "A111");
```

```
        System.out.println("<String, Integer>");
```

```
        System.out.println("name: "+gen1.getName());
```

```
        System.out.println("id : "+gen1.getId());
```

```
        Generic2<String, String> gen2 = new Generic2<String, String>();
```

```
        gen2.set("이순신", "A111");
```

```
        System.out.println("<String, String>");
```

```
        System.out.println("name: "+gen2.getName());
```

```
        System.out.println("id : "+gen2.getId());
```

```
    }
```

```
}
```

-> Generic2 클래스의 첫번째 자료형을 K, 두번째 자료형을 V로 임의지정

-> Generic2 클래스의 객체 gen1 의 자료형을

K (첫번째 자료형) 은 String으로,

V (두번째 자료형) 은 Integer로 재지정

=> 두번째 자료형은 Integer인데 "A111" 은 String이라서 에러

-> Generic2 클래스의 객체 gen2 의 자료형을

K (첫번째 자료형) 은 String으로,

V (두번째 자료형) 도 String으로 재지정

=> "이순신", "A111" 모두 String이므로 에러 발생하지 않음

- P409 제네릭 타입을 HashMap으로 지정해보기

```
public static void main(String[] args) {  
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();  
  
    Map<String, Object> hm = new HashMap<String, Object>();  
    hm.put("name", "홍길동");  
    hm.put("id", "hong");  
    hm.put("age", 30);  
    list.add(hm);  
  
    hm = new HashMap<String, Object>();  
    hm.put("name", "이순신");  
    hm.put("id", "lee");  
    hm.put("age", 40);  
    list.add(hm);  
  
    hm = new HashMap<String, Object>();  
    hm.put("name", "김유신");  
    hm.put("id", "kim");  
    hm.put("age", 50);  
    list.add(hm);  
  
    for (int i = 0; i < list.size(); i++) {  
        System.out.println("인덱스: "+i+", 이름: "+list.get(i).get("name")  
            +", 아이디: "+list.get(i).get("id")  
            +", 나이: "+list.get(i).get("age"));  
    }  
}
```

-> Map 자료형으로 이루어진 list라는 ArrayList 객체 생성

-> HashMap 클래스의 객체 hm 생성 후
값 넣고 (홍길동 hong 30)

-> list에 hm 을 add

-> 위 과정을
(이순신 lee 40) (김유신 kim 50) 값으로 반복

-> list 출력해보기

| |
|------------------------------------|
| 인덱스: 0, 이름: 홍길동, 아이디: hong, 나이: 30 |
| 인덱스: 1, 이름: 이순신, 아이디: lee, 나이: 40 |
| 인덱스: 2, 이름: 김유신, 아이디: kim, 나이: 50 |

● P413 chapter14 제너릭 연습문제

- ✓ 1. 제네릭에 대한 설명으로 옳바르지 않은 것은? ① 자동으로 형변환을 해준다 X -> 자동으로 형변환 하는게 아니라 내부적으로 재정의하는것
- ✓ 2. List 인터페이스와 ArrayList 클래스를 Member 타입으로 제네릭을 선언하고 전체 회원을 출력하는 코드를 작성하시오.

```
class MemberEx{
    private String id;
    private String name;
    private int age;

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

    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

필드

생성자

getter and setter

```
public class Excercise2 {
    public static void main(String[] args) {
        MemberEx me1 = new MemberEx( id: "hong", name: "홍길동", age: 30);
        MemberEx me2 = new MemberEx( id: "lee", name: "이순신", age: 40);
        MemberEx me3 = new MemberEx( id: "kim", name: "김유신", age: 50);

        List<MemberEx> memberList = new ArrayList<MemberEx>();
        memberList.add(me1);
        memberList.add(me2);
        memberList.add(me3);

        for (int i = 0; i < memberList.size(); i++) {
            System.out.println(memberList.get(i).getId()+" "+
                                memberList.get(i).getName()+" "+
                                memberList.get(i).getAge());
        }
    }
}
```

- 선생님 Generic 기초문제

- ✓ 1번 ArrayList에 제네릭을 사용하여 문자열을 저장하는 예시 코드를 작성하세요.

ArrayList 변수 list를 생성하고 여기에 apple, banana, orange 문자열을 추가하고 ArrayList의 사이즈와 ArrayList를 출력하세요.

```
import java.util.ArrayList;
import java.util.List;
public class Excercise_hk_1 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        list.add("apple");
        list.add("banana");
        list.add("orange");

        System.out.println("ArrayList size: "+list.size());
        for (int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i));
        }
    }
}
```

* 화면출력

```
ArrayList size: 3
apple
banana
orange
```

```
ArrayList size: 3
apple
banana
orange
```


- ✓ 2. ArrayList에 제너릭을 사용하여 Employee 클래스의 내용을 출력하는 코드를 작성하세요.

- Employee1 클래스

- + name, age, salary 필드를 정의하세요
- + 필드값으로 초기화할 수 있는 생성자를 만드세요.
- + 필드내용을 출력할 수 있도록 toString메소드를 재정의하세요.

```
class Employee1{
    private String name;    private int age;    private int salary;

    public Employee1(String name, int age, int salary) {
        this.name = name;    this.age = age;    this.salary = salary;
    }

    public String toString(){
        return "이름: "+name+", 나이: "+age+", 급여: "+salary; }
}

public class Excercise_hk_2 {
    public static void main(String[] args) {
        ArrayList<Employee1> employees = new ArrayList<>();

        employees.add(new Employee1( name: "홍길동", age: 25, salary: 50000));
        employees.add(new Employee1( name: "이순신", age: 30, salary: 60000));
        employees.add(new Employee1( name: "강감찬", age: 35, salary: 70000));

        for (int i = 0; i < employees.size(); i++) {
            System.out.println(employees.get(i));
        }
    }
}
```

* 화면출력

이름: 홍길동, 나이: 25, 급여: 50000

이름: 이순신, 나이: 30, 급여: 60000

이름: 강감찬, 나이: 35, 급여: 70000

이름: 홍길동, 나이: 25, 급여: 50000

이름: 이순신, 나이: 30, 급여: 60000

이름: 강감찬, 나이: 35, 급여: 70000

- ✓ 3번 학생들의 이름과 점수를 저장하는 HashMap에 등록하고, 이름을 입력하면 해당 학생의 점수를 출력하는 프로그램을 작성
 - HashMap 클래스에서 홍길동, 이순신, 강감찬 이름과 임의의 점수를 만드세요.
 - 학생이름정보(Key)를 넣어서 학생의 점수를 가져오는 코드를 작성하세요.
- * 화면출력
 - 이순신의 점수는 80점 입니다.

```
class student {
    String name;    int score;

    public student(String name, int score) {
        this.name = name;        this.score = score;    }

    public int getScore() { return score; }
}

public class Excercise_hk_3 {
    public static void main(String[] args) {
        HashMap<String, Integer> students = new HashMap<>();

        students.put("이순신", 80);
        students.put("홍길동", 90);
        students.put("강감찬", 70);

        String name = "이순신";
        int score = students.get(name);
        System.out.println(name+"의 점수는 "+score+"점 입니다.");
    }
}
```

-> list 가 아니라 map 이므로
add메서드가 아니라 put 메서드 이용

이순신의 점수는 80점 입니다.

- ✓ 4번 Employee3 클래스를 생성하고, Employee 객체의 id와 이름을 저장하는 HashMap을 만들고, id를 입력하면 해당하는 Employee 객체의 이름을 출력하는 프로그램을 작성

- Employee3 클래스 : + id, name 필드, 두 필드를 초기화 할수 있는 생성자, 두 필드의 getter함수를 만드세요.

```
class Employee3 {
    int id; String name;

    public Employee3(int id, String name) {
        this.id = id;        this.name = name;    }

    public int getId() { return id; }
    public String getName() { return name; }
}

public class Excercise_hk_4 {
    public static void main(String[] args) {
        HashMap<Integer, Employee3> employees = new HashMap<>();

        Employee3 e1 = new Employee3( id: 1, name: "홍길동");
        Employee3 e2 = new Employee3( id: 2, name: "이순신");
        Employee3 e3 = new Employee3( id: 3, name: "강감찬");

        employees.put(e1.getId(), e1);
        employees.put(e2.getId(), e2);
        employees.put(e3.getId(), e3);

        int id = 2;
        Employee3 employee = employees.get(id);
        System.out.println("id가 "+id+"인 Employee 의 이름은 "
            +employee.getName()+"입니다.");
    }
}
```

id 와 이름 값을 가지는 Employee3 클래스 생성

integer 자료형을 key로, Employee3 클래스 객체를 value로 가지는 HashMap인 employees 생성

Employee3 객체인 e1, e2, e3 생성

-> key 가 되는 int 자료형의 값을
e1, e2, e3객체의 첫번째 값인 id 값으로 가져오고,
value 는 e1, e2, e3 객체로 가져옴

=> 따라서 HashMap 인 employees 의 데이터는 아래와 같음

1 1 홍길동 / 2 2 이순신 / 3 3 강감찬

employee 의 이름

id가 2인 Employee 의 이름은 이순신입니다.

- ✓ 5번 ArrayList 로 작성했던 학생 점수 관리 프로그램을 HashMap 버전으로 업그레이드하세요
 - Student 클래스 : int 타입이었던 id 필드를 String 필드로 변경하세요.
 - StudentManagement3 클래스 : 아래와 같이 학생 ID를 Key로 하여 Student 클래스의 객체를 Value로 하여 HashMap을 활용한다.
 - > private static HashMap<String, Student> studentMap = new HashMap<>();

기존 Studnet 클래스에서 id에 포맷적용하는 기능을 StudnetManagement3 클래스에서 담당한다.

[Student 클래스 비교]

```
public class Student {
//필드
    private static int nextId = 1;
    private int id;
    private String name;
    private int koreanScore;
    private int englishScore;
    private int mathScore;

//생성자
    public Student(String name, int koreanScore, int englishScore, int mathScore) {
        this.id = nextId++;
        this.name = name;
        this.koreanScore = koreanScore;
        this.englishScore = englishScore;
        this.mathScore = mathScore;
    }
}
```

ArrayList 버전에서는 자동으로 id 가 ++ 될 수 있도록
nextId 변수 생성

```
public class Student {
//필드
    private String id;
    private String name;
    private int koreanScore;
    private int englishScore;
    private int mathScore;

//생성자
    public Student(String id, String name, int koreanScore, int englishScore, int mathScore) {
        this.id = id;
        this.name = name;
        this.koreanScore = koreanScore;
        this.englishScore = englishScore;
        this.mathScore = mathScore;
    }
}
```

HashMap 버전에서는 메인 클래스에서 id 를 ++하도록 코드를 작성했으므로
nextId 변수 필요없음

[메인 메서드에서 학생 리스트 생성 비교]

```
public class StudentManagement2 {  
    private static ArrayList studentList = new ArrayList();
```

ArrayList 버전에서는

학생 리스트를 ArrayList 객체로 생성

```
public class StudentManagement3 {  
    private static HashMap<String, Student> studentMap = new HashMap();  
    ➡ private static int idNum = 1;  
    ➡ private static DecimalFormat df = new DecimalFormat( pattern: "ID000");
```

HashMap 버전에서는

학생 리스트를 HashMap 객체로 생성하고, Key는 id (string타입) 로 설정

DecimalFormat 클래스를 활용하여 id의 형태가 " ID000 " 로 표시될 수 있도록 함

[초기 실행 메서드에서 임의의 학생 5명의 데이터 생성시 코드 비교]

//5명의 임의의 학생 데이터 생성

```
private static void initializeStudents() {
    String[] names = {"홍길동", "김철수", "이영희", "박종수", "최지수"};
    int[] koreanScores = {80, 90, 70, 85, 95};
    int[] englishScores = {85, 85, 75, 95, 90};
    int[] mathScores = {90, 70, 80, 95, 85};

    for (int i = 0; i < names.length; i++) {
        studentList.add(new Student(names[i], koreanScores[i], englishScores[i], mathScores[i]));
    }
}
```

ArrayList 버전에서는 학생 리스트에 데이터 추가시 add 메서드 이용

//5명의 임의의 학생 데이터 생성

```
private static void initializeStudents() {
    String[] names = {"홍길동", "김철수", "이영희", "박종수", "최지수"};
    int[] koreanScores = {80, 90, 70, 85, 95};
    int[] englishScores = {85, 85, 75, 95, 90};
    int[] mathScores = {90, 70, 80, 95, 85};

    for (int i = 0; i < names.length; i++) {
        ➡ String id = df.format(idNum++);
        studentMap.put(id, new Student(id, names[i], koreanScores[i], englishScores[i], mathScores[i]));
    }
}
```

HashMap 버전에서는

학생을 추가할 때마다 id 가 1씩 올라가도록 df.format(idNum++) ; 실행 후
put 메서드를 이용하여 key값은 id로, value를 Student 클래스의 객체로 저장

[전체 학생 정보 출력 코드 비교]

//1. 전체 학생 정보 출력

```
private static void printAllStudents(){
    if(studentList.isEmpty()){ System.out.println("등록된 학생이 없습니다."); }
    System.out.println("\n=====");
    System.out.println("이름\t\tID\t국어\t영어\t수학\t평균");
    System.out.println("\n=====");

    for(Object student: studentList){
        System.out.println((Student)student);
    }
    System.out.println("\n=====");
}
```

ArrayList 버전에서는 for문에서 List 객체인 studentList 에서 하나씩 출력하도록 코드 작성

//1. 전체 학생 정보 출력

```
private static void printAllStudents(){
    if(studentMap.isEmpty()){ System.out.println("등록된 학생이 없습니다."); }

    System.out.println("\n=====");
    System.out.println("이름\t\tID\t국어\t영어\t수학\t평균");
    System.out.println("\n=====");

    for(Object student: studentMap.values()){
        System.out.println(student);
    }
    System.out.println("\n=====");
}
```

HashMap 버전에서는 for문에서 Map 객체인 studentMap 에서 key를 제외한 value만 출력될 수 있도록 studentMap.values() 로 코드 작성

[최고/최저 평균 점수 학생 검색 코드 비교]

//3. 최저 평균점수 학생 검색

```
private static void searchLowestAverageStudent(){
    if(studentList.isEmpty()){ System.out.println("등록된 학생이 없습니다."); }

    ➡ double minAvg=((Student)studentList.get(0)).getAverage();
    int minIndex = 0;

    for (int i = 0; i < studentList.size(); i++) {
        if (((Student)studentList.get(i)).getAverage() < minAvg) {
            minAvg = ((Student)studentList.get(i)).getAverage();
            minIndex = i;
        }
    }

    System.out.println("최고 평균 점수 학생: "+((Student)studentList.get(minIndex)).getName());
}
```

ArrayList 버전에서는

for문에서 List 의 size 만큼 **인덱스** i가 1씩 커지면서

첫번째 학생의 평균점수를 minAvg로 대입하고

학생 각각의 평균점수와 minAvg 를 비교하여

가장 작은 평균점수를 minAvg 에 대입하여 해당 인덱스의 학생 이름을 출력함



```
➡ double minAvg=Double.MAX_VALUE;
String minKey = "";

for (String id : studentMap.keySet()) {
    Student student = studentMap.get(id);
    Double avg = student.getAverage();

    if (avg < minAvg) {
        minAvg = avg;
        minKey = id;
    }
}

System.out.println("최고 평균 점수 학생: "
    +(studentMap.get(minKey)).getName());
```

HashMap 버전에서는

for문에서 Map 의 **key** 에 해당하는 id 마다

가장 큰 상수를 minAvg에 대입하고

학생 각각의 평균점수와 minAvg를 비교하여

가장 작은 평균점수를 minAvg 에 대입하고, 해당 key의 학생 이름을 출력함

[학생 검색 코드 비교]

//4. 학생 검색

```
private static void searchStudent(){
    Scanner scanner = new Scanner(System.in);
    System.out.print("검색할 학생 이름을 입력하세요: ");
    String name = scanner.nextLine();
    boolean found = false;

    for (Object student:studentList) {
        if (((Student)student).getName().equals(name)) {
            System.out.println((Student)student);
            found = true;
        }
    }
    if (!found){
        System.out.println("해당 학생을 찾을 수 없습니다.");
    }
}
```

ArrayList 버전에서는 boolean 타입의 변수 found 를 활용한다.
for문 안에 if 문을 사용해서
studentList 의 학생 이름을 입력받은 이름과 하나하나 비교하여
같은 경우 해당 학생 데이터를 출력하고 found를 true로 바꿈.
모든 학생 이름 비교 후 입력받은 이름과 같은 이름이 없다면
found 가 false로 유지되어 해당학생이 없다는 메시지 출력함.

//4. 학생 검색

```
private static void searchStudent(){
    Scanner scanner = new Scanner(System.in);
    System.out.print("검색할 학생 ID를 입력하세요: ");
    String name = scanner.nextLine();

    if (!studentMap.containsKey(name)) {
        System.out.println("해당 학생이 존재하지 않습니다.");
        return;
    }
    System.out.println(studentMap.get(name));
}
```

HashMap 버전에서는

if 문을 사용해서

입력받은 id 와 같은 key 값이 student HashMap 에 없으면

해당 학생이 없다는 메시지를 출력하고,

입력받은 id 와 같은 key 값이 student HashMap 에 있으면

if문을 빠져나와

입력받은 id (key값)의 학생 정보를 출력함

[학생 추가 코드 비교]

//5. 학생 추가

```
private static void addStudent(){
    Scanner scanner = new Scanner(System.in);
    System.out.print("추가할 학생 이름을 입력하세요: ");
    String name = scanner.nextLine();
    System.out.print("국어 점수를 입력하세요: ");
    int koreanScore = scanner.nextInt();
    System.out.print("영어 점수를 입력하세요: ");
    int englishScore = scanner.nextInt();
    System.out.print("수학 점수를 입력하세요: ");
    int mathScore = scanner.nextInt();
    scanner.nextLine();

    Student newStudent = new Student(name, koreanScore, englishScore, mathScore);
    studentList.add(newStudent);

    System.out.println("학생이 추가되었습니다.");
}
```

ArrayList 버전에서는

새 학생이라는 객체에 입력받은 정보를 넣고

List에 add 메서드로 새 학생을 추가함

```
String id = df.format(idNum++);
Student newStudent = new Student(id, name, koreanScore, englishScore, mathScore);
studentMap.put(id, newStudent);

System.out.println("학생이 추가되었습니다.");
}
```

HashMap 버전에서는

id를 +1 해주고

새 학생이라는 객체에 입력받은 정보를 넣고

Map 에 put 메서드로

새 학생과 id (key값) 를 추가함

Chapter 15 랴다식

- ✓ 지금까지 클래스를 활용해 객체 지향적으로 코드를 작성 했으나
- ✓ 코드를 간결하게 하기 위해서 클래스 없이 기능 위주로 코드를 작성하는 것 (함수 지향적)

● P416 랴다식

[메서드 선언]

```
class LambdaFunctionEx1 implements InterfaceEx{  
    public int sum(int x, int y){ return x+y; }  
}
```

-> 해당 인터페이스를 구현 (implements) 하고나서
해당 인터페이스의 추상메서드를 선언

[람다식으로 메서드 선언]

```
public class LambdaFunctionEx {  
  
    public static void main(String[] args) {  
  
        InterfaceEx ie = (int x, int y) -> x+y;  
        // 랴다식, 랴다함수  
        // 랴다함수는 메서드를 구현할 인터페이스가 필요함  
        // 랴다함수는 인터페이스의 추상메서드를 랴다식으로 정의한 것  
  
        System.out.println(ie.sum(1, 2));  
  
    }  
}  
  
interface InterfaceEx {  
    public int sum(int x, int y);  
}
```

-> 해당 인터페이스를 구현 (implements) 하지 않으며,
메서드 선언시 리턴타입, 함수명, {}, "return" 을 생략하고
매개변수 타입 (int) 도 생략가능함
해당 인터페이스의 객체에서 " -> " 로 메서드의 기능을 표시

자바는 메서드만 실행할 수 없고, 객체를 통해서만 실행 할 수 있기 때문에
람다식은 메서드를 단순히 실행하는 것이 아니라,
해당 메서드를 포함하는 객체를 생성하는 것. (익명구현객체와 같은 개념)

- 람다식 만들기 연습 (선생님 예제)

-> 리턴타입, 함수명, {}, "return" 을 생략하고 ({})는 람다식 안에 실행문이 1줄일 때 생략)

매개변수 타입 (int) 도 생략가능함

해당 인터페이스의 객체에서 " -> " 로 메서드의 기능을 표시

| 메서드 | 람다식 |
|---|--|
| <pre>int max(int a, int b) { return a > b ? a : b; }</pre> | <p>① (int a, int b) -> a>b ? a:b ;</p> <p>매개변수 타입도 생략하면 : (a,b) -> a>b ? a:b</p> |
| <pre>int printVar(String name, int i) { System.out.println(name+"="+i); }</pre> | <p>② (name, i) -> System.out.println(name+"="+i);</p> |
| <pre>int square(int x) { return x * x; }</pre> | <p>③ (x) -> x * x ;</p> <p>매개변수가 한 개면 ()도 생략가능 : x -> x * x ;</p> |
| <pre>int roll() { return (int) (Math.random()*6); }</pre> | <p>④ ()->(int)(Math.random()*6);</p> <p>매개변수가 없을 때는 ()를 생략할 수 없음</p> |

- P417 람다식은 메서드 명이 없기 때문에 추상메서드가 1개 있는 인터페이스만 람다식으로 객체 생성 가능.
이렇게 메서드가 1개 있는 인터페이스를 함수적 인터페이스라고 부르고, (자바가 제공하는 대표적인 것은 Runnable)
추상메서드가 2개 이상인 인터페이스는 람다식 표현을 사용할 수 없음.
- P418 스레드 : 동시에 메서드가 수행되도록 병렬처리함

```
public class LambdaEx2 {
    public static void main(String[] args) {
        System.out.println("시작");
        Runnable run = () -> {
            for (int i = 0; i < 10; i++) {
                System.out.println("첫번째: " + i);
            }
        };
        Runnable run2 = () -> {
            for (int i = 0; i < 10; i++) {
                System.out.println("두번째: " + i);
            }
        };

        Thread t = new Thread(run);
        Thread t2 = new Thread(run2);
        t.start();
        t2.start();
        System.out.println("종료");
    }
}
```

Runnable 클래스 객체인 run과 run2를

각각 Thread 의 객체에 대입하여

각각 t, t2로 스레드 실행되도록 함

-> t과 t2가 동시에 실행되어

출력결과가 run 과 run2가 섞여서 나옴

(스레드는 메인 프로그램과 별개로 실행
되므로 "종료"라는 문자열이 먼저 출력되고
run과 run2가 실행됨)

| 시작 |
|--------|
| 종료 |
| 두번째: 0 |
| 두번째: 1 |
| 두번째: 2 |
| 두번째: 3 |
| 두번째: 4 |
| 두번째: 5 |
| 첫번째: 0 |
| 첫번째: 1 |
| 첫번째: 2 |
| 첫번째: 3 |
| 첫번째: 4 |
| 첫번째: 5 |
| 첫번째: 6 |
| 첫번째: 7 |
| 첫번째: 8 |
| 첫번째: 9 |
| 두번째: 6 |
| 두번째: 7 |
| 두번째: 8 |
| 두번째: 9 |

- P421 추상메서드를 다양하게 람다식으로 재정의하기

```
public class LambdaEx4 {  
    public static void main(String[] args) {  
        LambdaInterface4 f4 = (x,y) -> {return x * y;};  
        System.out.println(f4.cal( x: 3, y: 2));  
  
        f4 = (x,y) -> x+y;  
        System.out.println(f4.cal( x: 3, y: 2));  
  
        f4 = (x,y) -> {return x/y;};  
        System.out.println(f4.cal( x: 3, y: 2));  
  
        f4 = (x,y) -> x % y;  
        System.out.println(f4.cal( x: 3, y: 2));  
  
        f4 = (x,y) -> sum(x,y);;  
        System.out.println(f4.cal( x: 3, y: 2));  
    }  
  
    static int sum(int x, int y){ return x+y; }  
}  
  
@FunctionalInterface  
interface LambdaInterface4{  
    int cal (int x, int y);  
}
```



6
5
1
1
5

-> LambdaInterface4의 추상메서드 cal을

메인메서드에서 람다 표현식으로

합, 곱, 나머지, 나누기, 등으로 재정의함

마지막 실행문 sum(x,y) 의 경우

해당 매개변수 x,y 그대로 sum 메서드에 넣고 sum의 리턴값을 리턴함

(= sum 메서드와 같은 기능 수행)

- P422 람다식에서 사용하는 this

```
public class LambdaEx5 {  
    public static void main(String[] args) {  
        Outer o = new Outer();  
        Outer.Inner i = o.new Inner();  
        i.method();  
    }  
}  
  
@FunctionalInterface  
interface LambdaInterface5{  
    void method();  
}  
  
class Outer{  
    public int iv = 10;  
  
    class Inner {  
        public int iv = 20;  
  
        void method() {  
            final int iv = 40;  
            LambdaInterface5 f5 = () -> {  
                System.out.println("Outer.this.iv : " + Outer.this.iv);  
                System.out.println("this.iv : " + this.iv);  
                System.out.println("iv : " + iv);  
            };  
            f5.method();  
        }  
    }  
}
```

람다식 내부의 this 는 람다 객체를 감싸는 class를 의미함.

람다 표현식 내의 지역변수 접근시에는 지역변수 상수화 필요.

```
Outer.this.iv : 10  
this.iv : 20  
iv : 40
```