JAVA 프로그래밍

클래스와 객체 2

객체 지향의 3대 특징

- **캡슐화**(Encapsulation)

  : 객체의 필드, 메서드를 하나로 묶고, 실제 구현 내용을 감추는 것

- **상속**(Inheritance)

  : 부모 클래스를 물려받아 자식 클래스를 생성하여 사용 가능하도록 하는 것

- **다형성**(Polymorphism)

  : 같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질

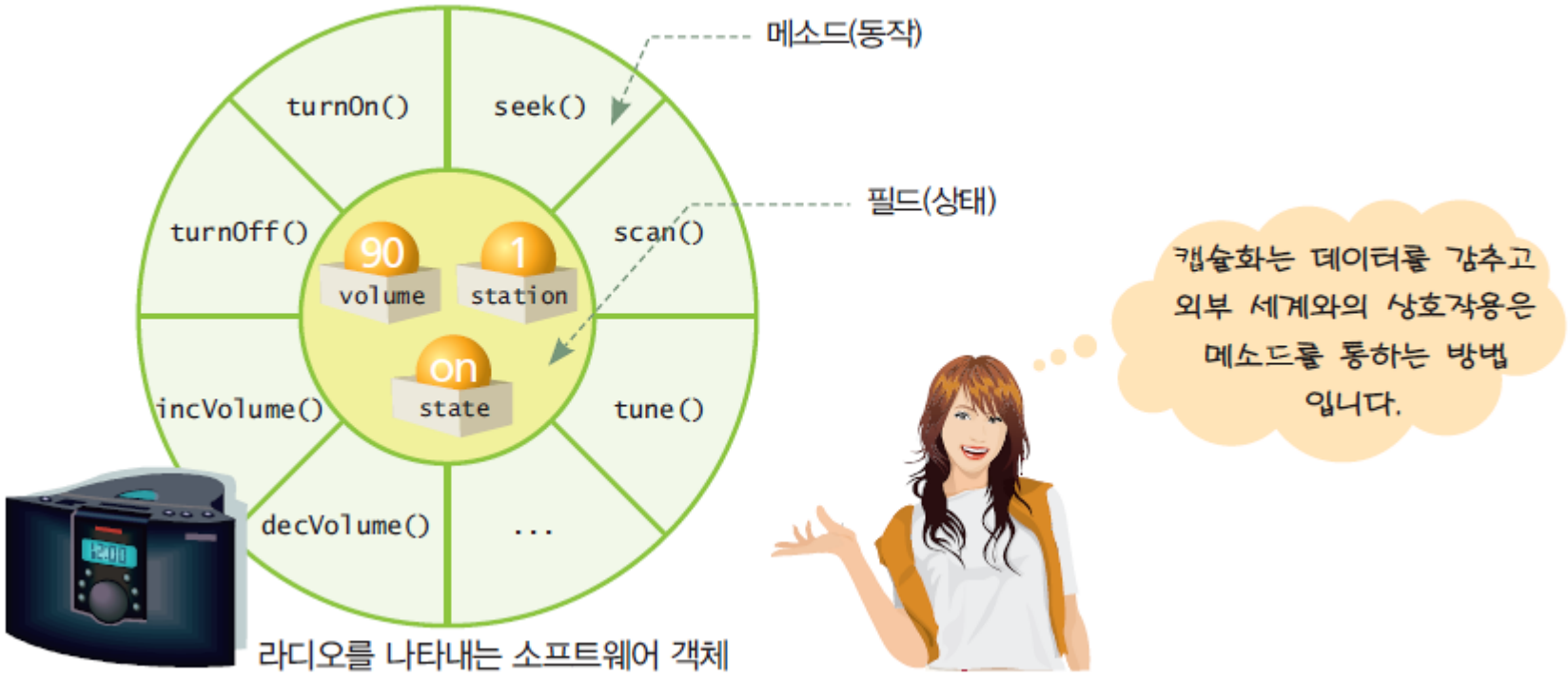
캡슐화

- **캡슐화(Encapsulation)**

  : 객체의 필드, 메서드를 하나로 묶고, 실제 구현 내용을 감추는 것

외부 객체는 객체 내부의 구조를 알지 못하며,
객체가 노출하여 제공하는 필드와 메서드만 이용 가능

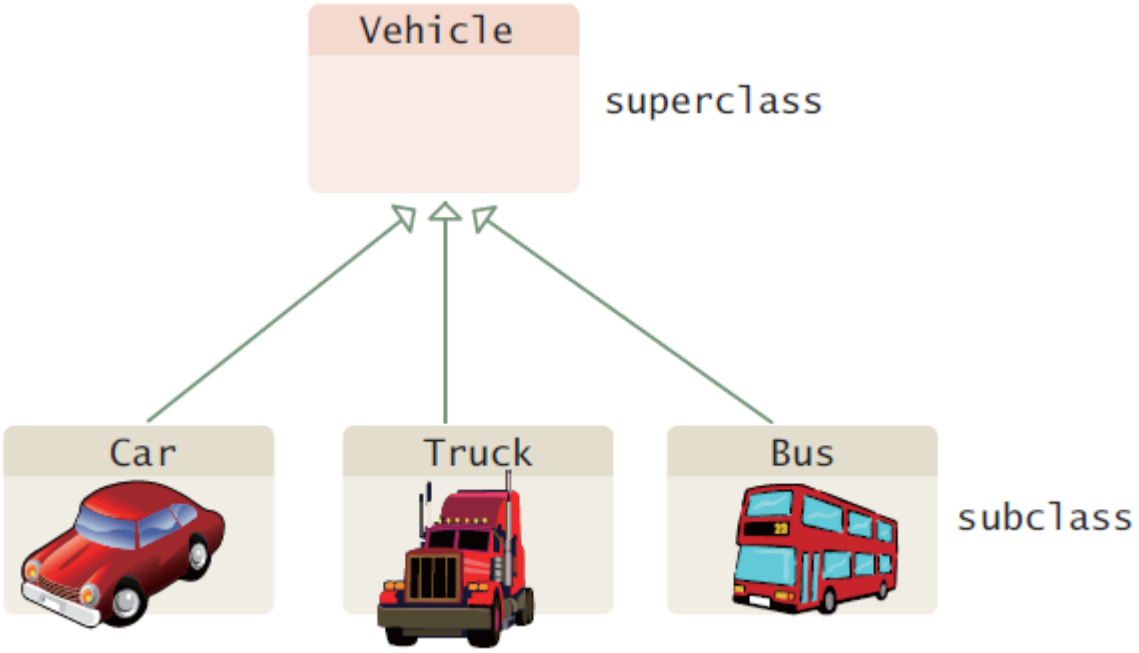
캡슐화와 정보은닉


메소드(동작)
turnOn()
seek()
turnOff()
scan()
필드(상태)
90
volume
1
station
on
state
incVolume()
tune()
decVolume()
...
캡슐화는 데이터를 감추고
외부 세계와의 상호작용은
메소드를 통하는 방법
입니다.
라디오를 나타내는 소프트웨어 객체

상속

- **상속(inheritance)**
  : 이미 작성된 클래스(부모 클래스, superclass)를 물려받아 새로운 클래스(자식 클래스, subclass)를 생성하는 기법

- 기존의 코드를 재활용하기 위한 기법(반복된 코드의 중복 제거)

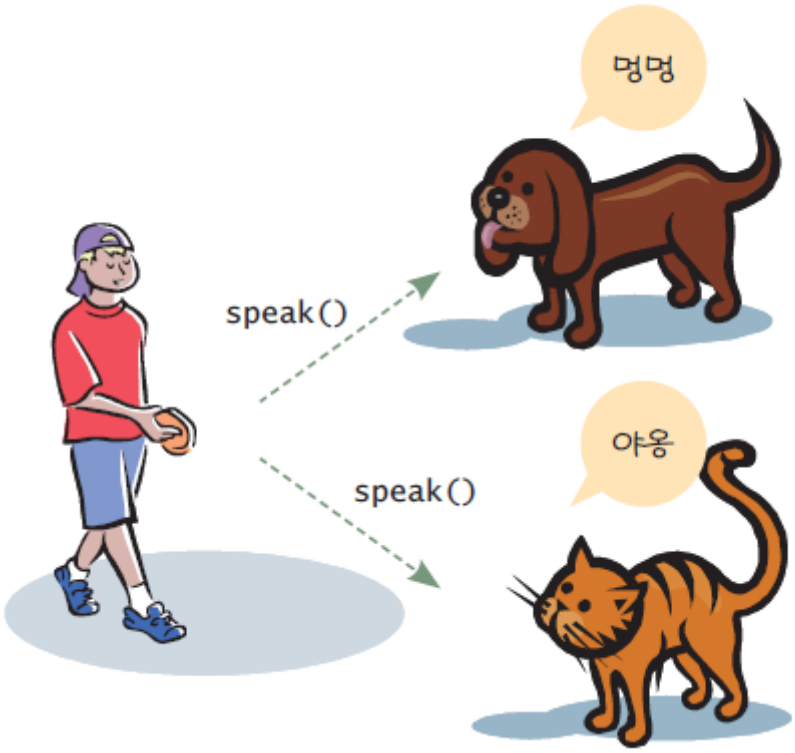


다형성(Polymorphism)

- **다형성(Polymorphism)**

  : 같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질

- 개념적으로 동일한 작업을 하는 멤버 함수들에 같은 이름을 부여할 수 있으므로 코드가 더 간단해진다.



접근 제어의 종류

- **접근 제어 :** 다른 클래스가 특정한 필드나 메서드에 접근하는 것을 제어하는 것

- **접근 제어자(access modifier)**
  - **public** : 외부 클래스 어디에서든 접근 할 수 있음(접근 제한 없음)
  - **package**(default) : 같은 패키지 안에 있는 클래스들만 사용할 수 있음
  - **private** : 같은 클래스 내부에서만 사용할 수 있음

분류	접근 지정자	클래스 내부	같은 패키지 내의 클래스	다른 모든 클래스
전용 멤버	private	O	X	X
패키지 멤버	없음	O	O	X
공용 멤버	public	O	O	O

정보 은닉
(information hiding)

- public 접근 제어자로 선언한 변수나 메서드는 외부 클래스에서 접근이 가능
- **private** 접근 제어자(access modifier)
- 클래스의 외부에서 클래스 내부의 멤버 변수나 메서드에 접근(access)하지 못하게 하는 경우 사용
- 멤버 변수나 메서드를 외부에서 사용하지 못하도록 하여 오류를 줄일 수 있음
- **private** 변수에 대해서는 외부코드에서 변수를 사용할 경우,
  get(), set() 메서드 제공해야 함
  → 이클립스에서는 자동으로 만들 수 있는 기능 제공

private 변수 사용 테스트

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

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

get(), set() 메서드 사용

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

    public String getStudentName() {
        return studentName;
    }

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

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

✓ public 메서드를 통해,
private 변수에 접근 가능

정보 은닉의 예
- public으로 선언한 경우

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

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

2월에 31일이 존재하지 않지만 오류를 막을 수 없음!
☞ 날짜가 변경되어 버린다.

정보 은닉의 예
- private으로 선언한 경우

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

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

❖ 날짜가 변수에 저장되기 전에 체크하여 오류를 방지 할 수 있음

this 예약어

- 자신의 메모리를 가리킴
- 생성자에서 다른 생성자를 호출
- 자신의 주소를 반환함

자신의 메모리를 가리키는 this

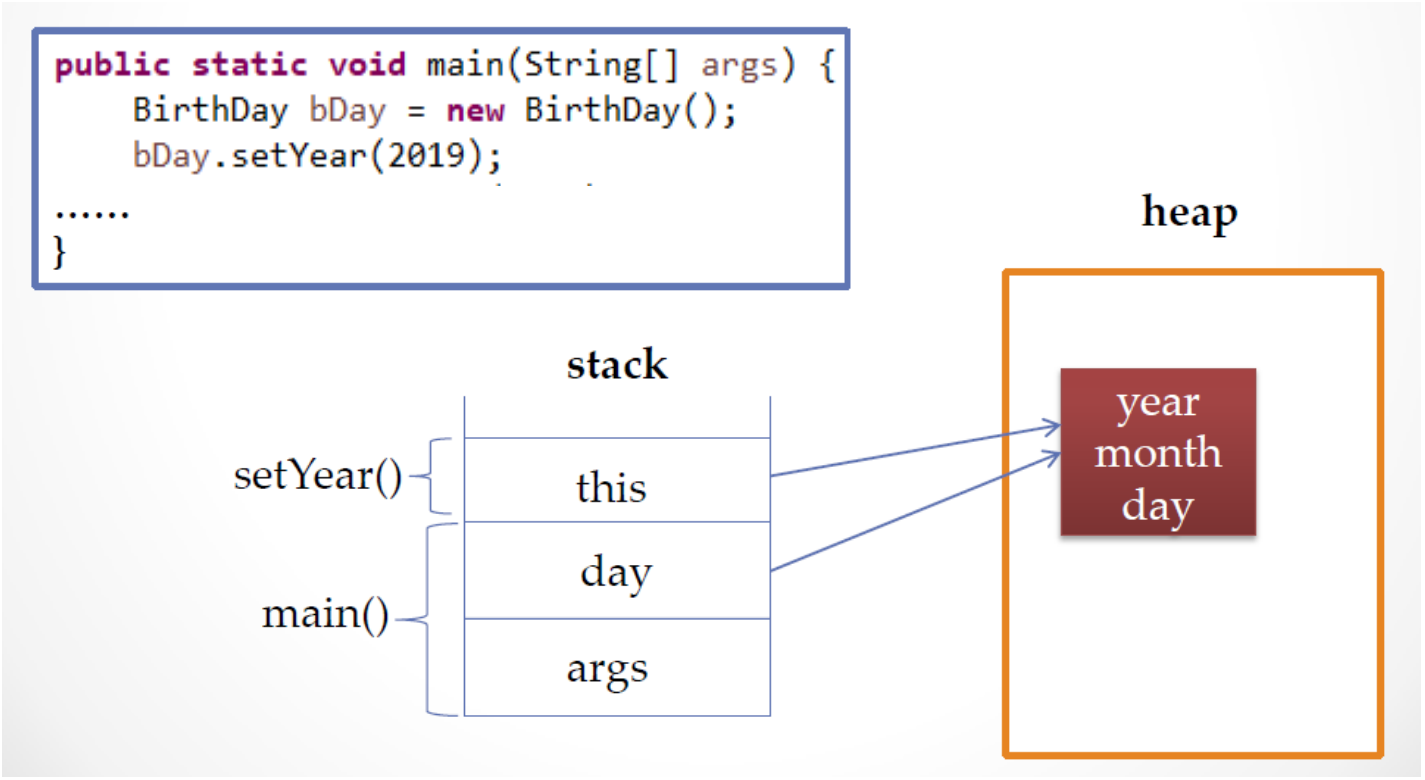
```java
class BirthDay {
    int day;
    int month;
    int year;

    public void setYear(int year) {
        // 자신의 객체의 year 멤버에 매개변수 year 값으로 설정
        this.year = year;
    }

    public void showThis() {
        System.out.println(this);
    }
}
```

```java
public class ThisExample {
    public static void main(String[] args) {
        BirthDay bDay = new BirthDay();
        bDay.setYear(2019);
        System.out.println(bDay);    // 참조 변수 출력
        bDay.showThis();             // this 출력 메서드 호출
    }
}
```

자신의 메모리를 가리키는 this

- 생성된 인스턴스 자신을 가리키는 예약어

```
public static void main(String[] args) {
    BirthDay bDay = new BirthDay();
    bDay.setYear(2019);
.....
}
```



자신의 메모리를 가리키는 this

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

위 코드에서 this를 생략하게 되면,

name이나 age는 매개변수(parameter)로 사용되는 name, age로 인식

생성자에서 다른 생성자를 호출하는 this

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

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

this를 이용하여 다른 생성자를 호출할 때,
그 이전에 어떠한 statement도 사용할 수 없다.

위와 같이 생성자가 여러 개이고 파라미터만 다른 경우,
생성자 오버로딩(constructor overloading)이라고 한다.

자신의 주소를 반환하는 this

```
Person getPerson(){
        return this;
}

public static void main(String[] args) {

        Person p = new Person();
        p.name = "James";
        p.married = true;
        p.numberOfChild = 3;
        p.age = 40;

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

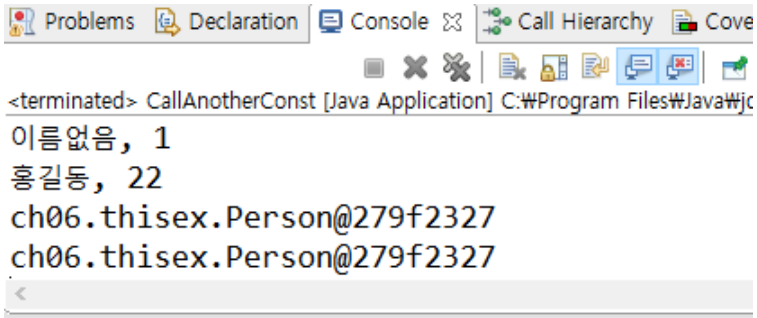
여기서의 this는
Reference value(참조 값)을 반환

자신의 주소를 반환하는 this

```
class Person {
    String name;
    int age;

    Person() {
        this("이름없음", 1);   // 생성자 내부에 다른 생성자를 호출할 경우 this 사용
        // Person(String name, int age) 생성자 호출
    }

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

    Person returnSelf() {
        return this;     // 자신을 반환(클래스형)
    }

    void printInfo() {
        System.out.println(name + ", " + age);
    }
}
```

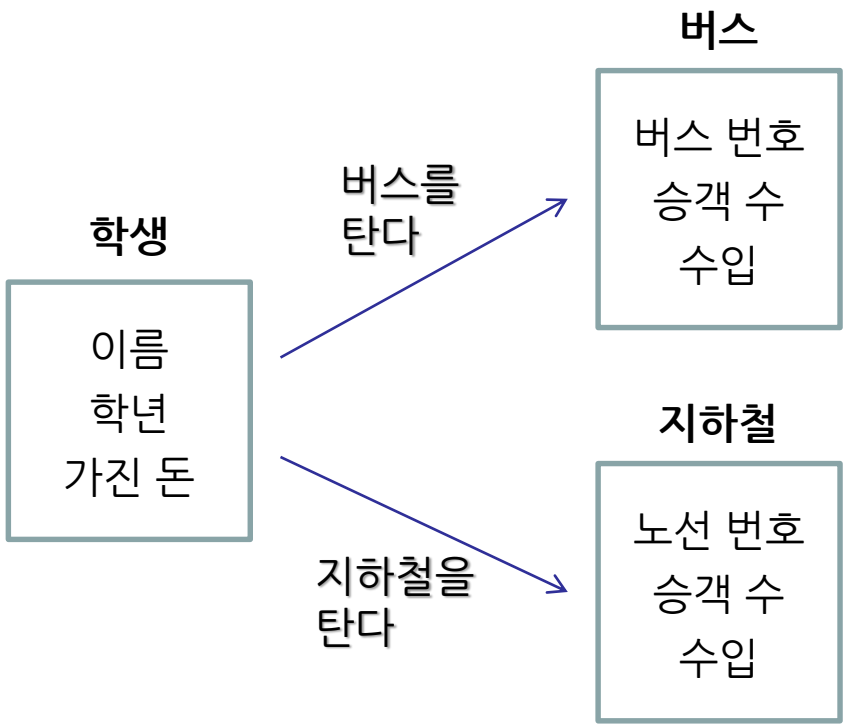
자신의 주소를 반환하는 this

```
public class CallAnotherConst {
    public static void main(String[] args) {
        Person p = new Person();
        Person p1 = new Person("홍길동", 22);
        p.printInfo();
        p1.printInfo();

        Person p2 = p.returnSelf();
        System.out.println(p2);      // 반환값
        System.out.println(p);       // 생성된 객체 값
    }
}
```



객체 간의 협력

✓ 학생이 버스나 지하철을 가는 상황을 객체 지향으로 프로그래밍



- 학생 클래스 구현

```java
public class Student {
    // 멤버 변수, 인스턴스 변수
    String studentName;
    int grade;
    int money;
    // 학생의 이름과 가진 돈을 매개변수로 만든 생성자
    Student(String studentName, int money) {
        this.studentName = studentName;
        this.money = money;
    }
    // 학생이 버스를 타는 메서드
    public void takeBus(Bus bus) {
        bus.take(1200);
        this.money -= 1200;
    }
    // 학생이 지하철을 타는 메서드
    public void takeSubway(Subway subway) {
        subway.take(1500);
        this.money -= 1500;
    }
    // 학생의 정보 출력 메서드
    void printInfo() {
        System.out.println(studentName + "학생의 현재 남은 돈은 " + money + "원입니다.");
    }
}
```

- 버스 클래스 구현

```
public class Bus {
    // 멤버 변수
    int busNumber;
    int passengerCount;
    int money;

    // 버스 번호로 생성자를 만든다. 매개변수 생성자
    Bus(int busNumber) {
        this.busNumber = busNumber;
    }
    // 승객이 버스를 타는 메서드
    public void take(int money) {
        this.money += money;     // 수입 증가
        this.passengerCount++;   // 승객의 수 증가
    }
    // 버스 정보 출력 메서드
    void printInfo() {
        System.out.println(busNumber + "버스의 승객은 " + passengerCount + "명이고, "
                           + "수입은 " + money + "원입니다.");
    }
}
```

- 지하철 클래스 구현

```
public class Subway {
    // 멤버 변수
    String lineNumber;
    int passengerCount;
    int money;
    // 지하철 호선 번호로 생성자를 만든다. 매개변수 생성자
    public Subway(String lineNumber) {
        this.lineNumber = lineNumber;
    }
    // 지하철을 승객이 타는 메서드
    public void take(int money) {
        this.money += money;     // 수입 증가
        this.passengerCount++;   // 승객의 수 증가
    }
    // 지하철 정보 출력 메서드
    void printInfo() {
        System.out.println(lineNumber + "의 승객은 " + passengerCount + "명이고, "
                           + "수입은 " + money + "원입니다.");
    }
}
```

- 객체 간의 협력
  (버스와 지하철 타기)

```
public class TakeTrans {
    public static void main(String[] args) {
        // 생성자를 호출하여 인스턴스 만들기
        Student studentTomas = new Student("Tomas", 10000);
        Student studentJane = new Student("Jane", 5000);
        Student stduentKim = new Student("Kim", 3000);

        Bus bus1004 = new Bus(1004);     // 1004번 버스 생성
        studentTomas.takeBus(bus1004);   // 1004번 버스를 토마스 학생이 탄다.

        studentJane.takeBus(bus1004);

        studentTomas.printInfo();
        studentJane.printInfo();
        bus1004.printInfo();

        Subway subwayOne = new Subway("1호선");// 1호선 지하철 생성
        stduentKim.takeSubway(subwayOne);// 김학생이 지하철 1호선을 탄다.
        stduentKim.printInfo();
        subwayOne.printInfo();
    }
}
```

```
<terminated> TakeTrans [Java Application] C:₩Program
Tomas학생의 현재 남은 돈은 8800원입니다.
Jane학생의 현재 남은 돈은 3800원입니다.
1004버스의 승객은 2명이고, 수입은 2400원입니다.
Kim학생의 현재 남은 돈은 1500원입니다.
1호선의 승객은 1명이고, 수입은 1500원입니다.
```

Quiz 객체 간 협력

- **택시를 타는 과정을 추가로 구현해 보세요.**

  길동 학생이 늦잠을 자버려 택시를 타고 학교에 가게 되었다고 합니다.
  택시 요금은 10,000원을 지불했습니다. 이 과정을 구현해 보면?

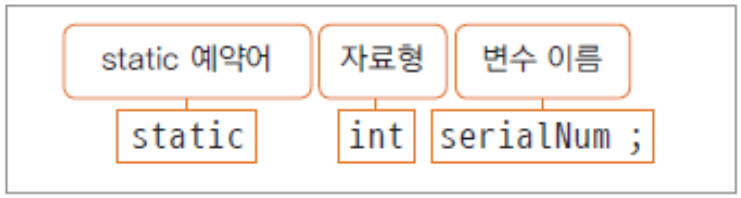
  (Hint)

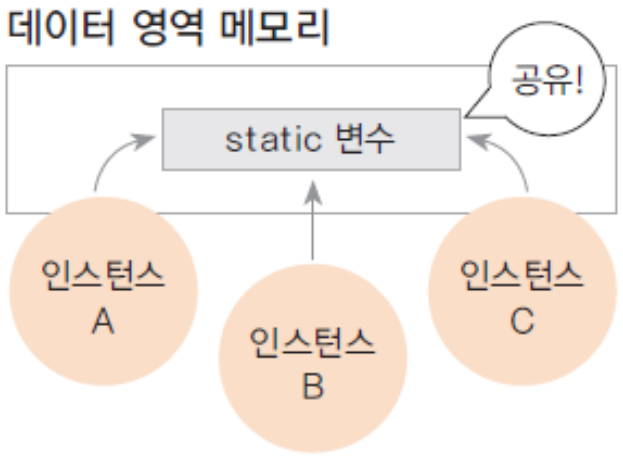
  택시 클래스와 학생이 택시를 타는 객체 간의 협력 관계를 구현

static 변수

- **static 변수**의 정의와 사용 방법

- 여러 개의 인스턴스가 **같은 메모리의 값을 공유하기 위해 사용**





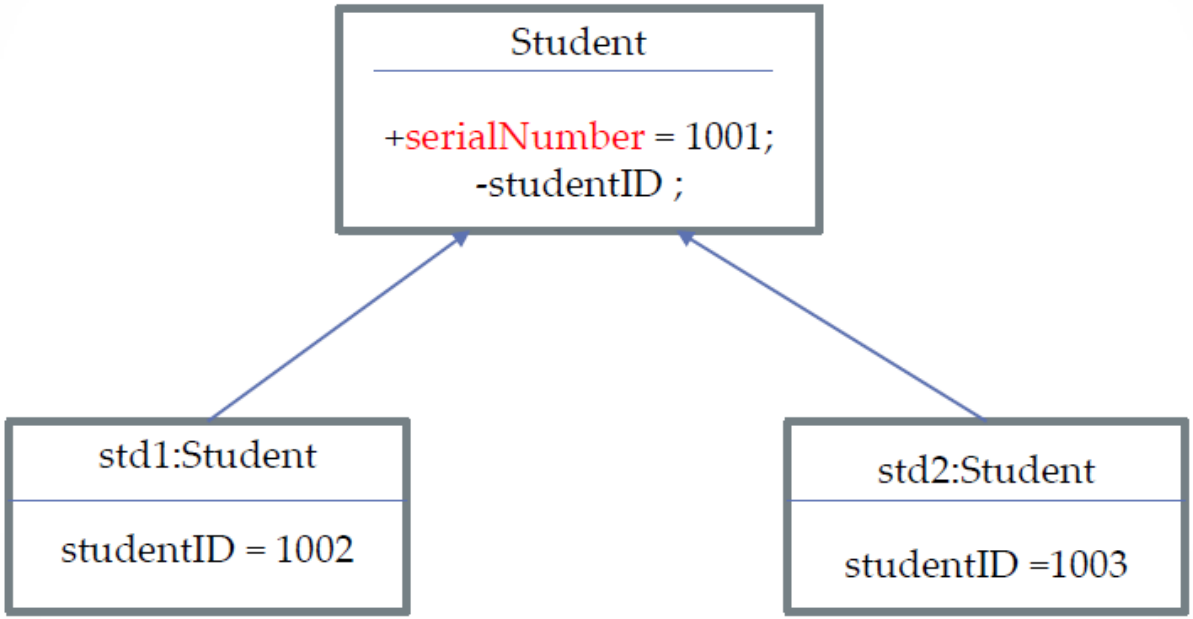
static 변수

✓ static 변수는 인스턴스가 생성될 때마다 다른 메모리를 가지는 것이 아니라 프로그램이 메모리에 적재(load)될 때, 데이터 영역의 메모리에 생성됨

✓ 인스턴스의 생성과 관계없이 클래스 이름으로 직접 참조함

- Student.serialNum = 100; // serailNum은 static 변수
- '클래스 변수' 라고도 함 (모든 인스턴스가 공유)
  - 멤버 변수는 다른 말로 '인스턴스 변수'라 함

static 변수 vs. 인스턴스 변수

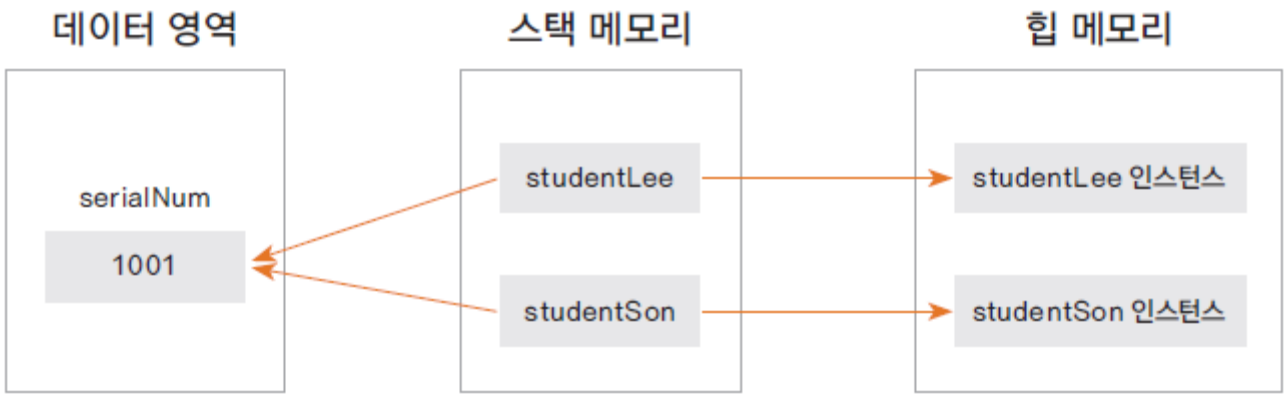


note *: serialNumber를 static 으로 선언하면 모든 student instance에 대해 하나의 변수로 유지 되고 이러한 변수를 class 변수라 한다.*

static 변수 예

- 여러 인스턴스가 하나의 메모리 값을 공유할 때 필요
- 학생이 생성될 때마다 학번이 증가해야 하는 경우
- 기준이 되는 값은 static 변수로 생성하여 유지함

  ✓ **각 학생이 생성될 때 마다 static 변수 값을 복사해서 하나 증가시킨 값을 생성된 인스턴스의 학번 변수에 저장해 줌**



static 변수 사용하기

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

    public String getStudentName() {
        return studentName;
    }

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

static 변수 테스트

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

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

학번 자동으로 부여하기

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

```
<terminated> StudentTest2 [Java Application]
1001
이루마 학번:1001
1002
손연재 학번:1002
```

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

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

static 메서드

- '클래스 메서드(class method)' 라고도 함
- 메서드에 static 키워드를 사용하여 구현
- 주로 static 변수를 위한 기능 제공
- 정적 메서드는 this 키워드를 사용할 수 없음 (this가 참조할 객체가 없기 때문)

- static 메서드에서 인스턴스 변수를 사용할 수 없음
- static 메서드도 인스턴스의 생성과 관계없이 클래스 이름으로 직접 메서드 호출
  - Student.getSerialNum(); // getSerialNum()은 static 메서드

✓ 인스턴스의 변수의 경우, 반드시 인스턴스가 먼저 생성되어야 하므로,
static 메서드에서는 생성이 불확실한 인스턴스 변수를 사용할 수 없음

클래스 변수

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

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

    }
}
```

클래스 메서드

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

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

    public String getStudentName() {
        return studentName;
    }

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

    public static int getSerialNum() {
        int i = 10;
        return serialNum;
    }

    public static void setSerialNum(int serialNum) {
        Student2.serialNum = serialNum;
    }
}
```

✓ *static 변수를 위한 메서드*

클래스 메서드로 학번 출력

```
public class StudentTest4 {
    public static void main(String[] args) {
        Student2 studentLee = new Student2();
        studentLee.setStudentName("이루마");
        //serialNum 값 가져오기 위해 get 메서드 호출, 클래스 이름으로 직접 호출
        System.out.println(Student2.getSerialNum());
        System.out.println(studentLee.studentName + " 학번:"
                                + studentLee.studentID);

        Student2 studentSon = new Student2();
        studentSon.setStudentName("손연재");
        System.out.println(Student2.getSerialNum());
        System.out.println(studentSon.studentName + " 학번:"
                                + studentSon.studentID);
    }
}
```

변수의 유효 범위

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 private이 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 private이 아니면 클래스 이름으로 다른 클래스에서 사용 가능	데이터 영역	프로그램이 처음 시작할 때 상수와 함께 데이터 영역에 생성되고 프로그램이 끝나고 메모리를 해제할 때 소멸됨

※ 질문에 알맞은 변수 유형은 무엇?

- **함수에서 기능 구현을 위해 잠시 사용한다면?**
- **클래스의 속성을 나타내고 인스턴스마다 다른 값을 가진다면?**
- **여러 인스턴스에 공유해서 사용하도록 한 번만 생성되어야 한다면?**

클래스 메서드 vs. 인스턴스 변수

- 클래스 메서드 내부에서는 인스턴스 변수를 사용할 수 없음.

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

   …

public static int getSerialNum() {
    int i = 10;      // local variable
    studentName = "이루마";   // error : non-static field
    return serialNum;
}

   …
```

Quiz static 변수, static 메서드

- 학생마다 각각 다른 학생 카드가 발급됩니다.

  학생 카드 번호에는 학번에 100을 더한 값입니다.

  Student3 클래스를 만들어 학생 카드번호 멤버 변수를 추가하고, 학생이 생성될 때마다 학생 카드 번호를 부여하도록 합니다.

  StudentTest6 클래스를 만들어 학생 두 명을 생성하고, 두 학생의 카드 번호를 출력해 보세요.

[실행 결과]

```
<terminated> StudentTest6 [Java Application]
학번 : 1001, 카드 번호 : 1101
학번 : 1002, 카드 번호 : 1102
```

Quiz static variable, static method

〈StaticMethodTest.java〉

[클래스 이름] Employee

[클래스 멤버]

- 변수 : name(이름), salary(월급), count(직원의 수 카운트, 정적 변수)
- 생성자
- getCount() : 직원의 수 반환

[main() Hint]

객체 생성, 생성자 호출, getCount() 호출, 출력문(직원의 수)

[실행 결과]

직원 이름 : 김철수, 월급 : 3,500,000원
직원 이름 : 최수철, 월급 : 5,000,000원
직원 이름 : 박미나, 월급 : 2,000,000원
현재 직원의 수 : 3

static 응용 : singleton 패턴

자동차 회사가 있고,
자동차 회사에는 여러 직원들이 있고,
자동차 회사에는 여러 개의 공장들이 있으며,
생산된 자동차를 운반하는 운반차들이 있다.

여기에서 객체는 무엇이고,
이 중에서 자동차 회사 시스템을 만들 때,
단 한 개만 존재하는 객체는 무엇일까?

싱글톤 패턴으로 회사 클래스 구현

- 1단계 : 생성자를 private로 만들기

```
package singleton;

public class Company {
    private Company() {
        System.out.println("유일한 회사");
    }
}
```

- 2단계 : 클래스 내부에 static으로 유일한 인스턴스 생성하기

```
package singleton;

public class Company {
    private static Company instance = new Company();  // 유일하게 생성한 인스턴스

    private Company() {
        System.out.println("유일한 회사");
    }
}
```

싱글톤 패턴으로 회사 클래스 구현

- 3단계 : 외부에서 참조할 수 있는 public 메서드 만들기

```
package singleton;

public class Company {
    private static Company instance = new Company();

    private Company() {
        System.out.println("유일한 회사");
    }

    public static Company getInstance() {
        if (instance == null) {
            instance = new Company();
        }
        return instance;     // 유일하게 생성된 인스턴스 반환
    }
}
```

싱글톤 패턴으로 회사 클래스 구현

- 4단계 : 실제 사용하는 유일한 인스턴스 코드 만들기

```
package singleton;

public class CompanyTest {
    public static void main(String[] args) {
        Company myCompany1 = Company.getInstance();
        Company myCompany2 = Company.getInstance();

        System.out.println(myCompany1 == myCompany2);
    }
}
```

Quiz 싱글톤 패턴으로 클래스 구현

- 자동차 공장은 유일한 객체이고, 이 공장에서 생산되는 자동차는 제작될 때마다 고유번호가 부여됩니다. 자동차 번호가 10001부터 시작되어 자동차가 생성될 때마다 10002, 10003, … 이렇게 번호가 붙도록 자동차 공장 클래스, 자동차 클래스를 만들어 보세요.(단, 두 클래스는 다음 테스트 코드가 수행될 수 있도록 구현하세요.)

```
public class CarFactoryTest {
    public static void main(String[] args) {
        CarFactory factory = CarFactory.getInstance();  // 싱글톤 패턴

        Car mySonata = factory.createCar();      // Car 생성
        Car yourSonata = factory.createCar();

        System.out.println(mySonata.getCarNum());
        System.out.println(yourSonata.getCarNum());
    }
}
```