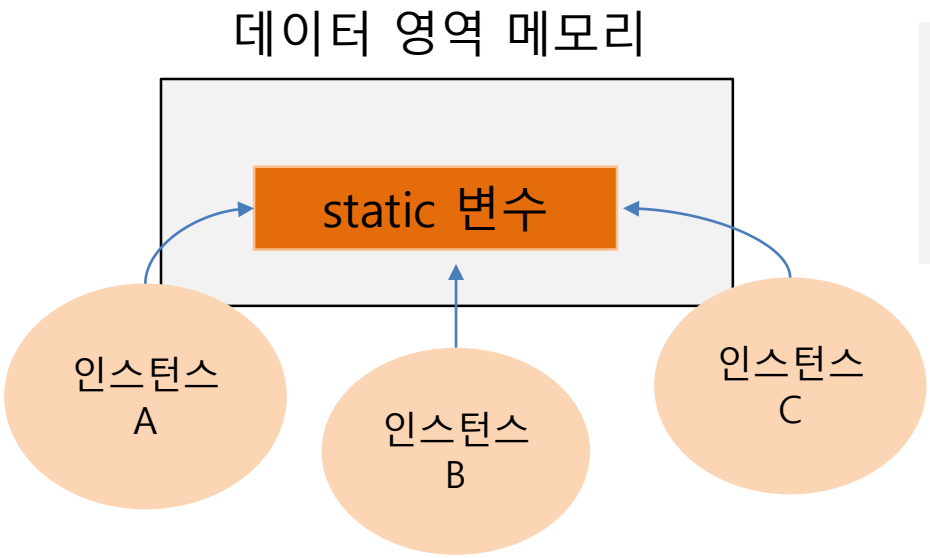
6장. 클래스와 객체2

static

static 변수

▪ static 변수의 정의와 사용 방법

- 다른 멤버변수처럼 인스턴스가 생성될 때마다 새로 생성되는 변수가 아니다.
- 프로그램이 실행되어 **메모리에 적재(load)**될때 메모리 공간이 할당된다.
- 여러 개의 인스턴스가 같은 **메모리의 값을 공유**하기 위해 사용



static 예약어

```
static int serialNum=1000;
```

static 변수

- 차량번호 자동 부여

```
public class Car {

    private static int serialNum = 1000; //정적 변수
    private int carNumber;

    public Car() {
        serialNum++;
        carNumber = serialNum;
    }

    public int getCarNumber() {
        return carNumber;
    }
}
```

static 변수

- 차량번호 자동 부여

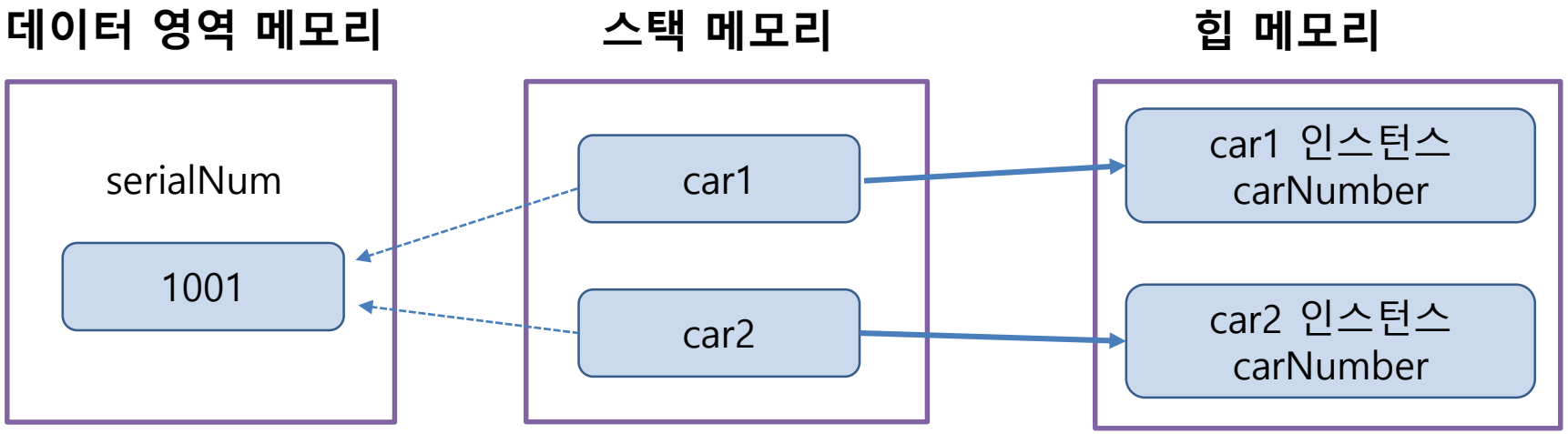
```
public class CarTest {

    public static void main(String[] args) {
        Car car1 = new Car();
        Car car2 = new Car();
        Car car3 = new Car();

        System.out.println("차량번호: " + car1.getCarNumber());
        System.out.println("차량번호: " + car2.getCarNumber());
        System.out.println("차량번호: " + car3.getCarNumber());
    }
}
```

```
차량번호: 1001
차량번호: 1002
차량번호: 1003
```

인스턴스와 참조변수

◆ 차량번호 자동 부여

- 차가 생성될 때마다 차량번호가 증가해야 하는 경우
- 기준이 되는 값은 static 변수로 생성하여 유지 함.



static으로 선언한 serialNum 변수는 모든 인스턴스가 공유한다 . 즉 두 개의 참조변수가 동일한 변수의 메모리를 가리키고 있다.

static 변수 사용하기

- **static 변수 사용하기**

```
public class ScopeStaticVar {

    static int x = 0;  //정적 변수

    public static int oneUp() {
        x++;
        return x;
    }

    public static void main(String[] args) {

        System.out.println(oneUp());
        System.out.println(oneUp());
        System.out.println(oneUp());

        //x의 값
        System.out.println("x = " + x);
    }
}
```

static 메서드 만들기

- **static 메서드 정의하기**

```
class Greeting{

    public static void sayHello() {
        System.out.println("안녕~");
    }

    public static void sayGoodBye() {
        System.out.println("잘가~ ");
    }
}

public class UseHello {

    public static void main(String[] args) {
        //static이 있는 메서드는 클래스 이름으로 직접 접근
        //new로 생성하지 않음
        Greeting.sayHello();
        Greeting.sayGoodBye();
    }
}
```

static 클래스 만들기

- **Math 클래스 만들기**

```
class MyMath{
    //절대값 계산
    public static int abs(int x) {
        if (x < 0)
            return -x;
        else
            return x;
    }

    //거듭제곱 계산
    public static int pow(int x, int y) {
        int num = 1;  //곱하기 초기값

        for(int i=0; i < y; i++) {
            num *= x; //num = num * x;
        }

        return num;
    }
}
```

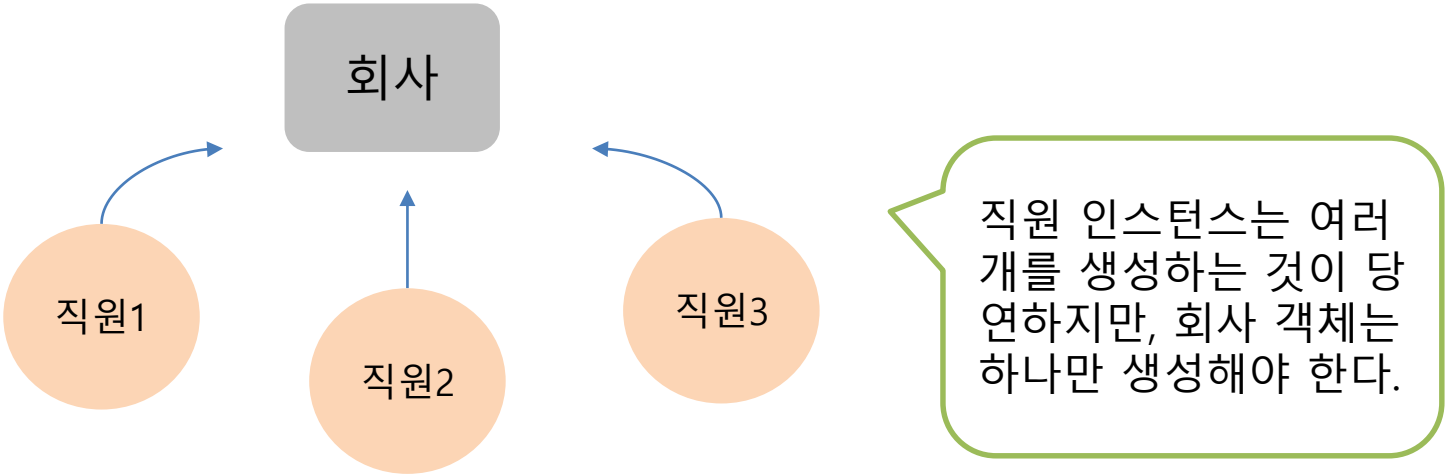
static 클래스 만들기

- **Math 클래스 만들기**

```
public class MathTest {

    public static void main(String[] args) {

        //절대값 호출
        int value1 = MyMath.abs(-5);
        System.out.println(value1);

        //거듭제곱 호출
        int value2 = MyMath.pow(2, 3);
        System.out.println(value2);

        //Math 클래스와 비교
        System.out.println(Math.abs(-5));
        System.out.println(Math.pow(2, 3));
    }
}
```

static 응용 : 싱글톤 패턴

- **Single 패턴이란?**
  - 객체지향 프로그램에서 인스턴스를 단 하나만 생성하는 디자인 패턴
  - static을 응용하여 프로그램 전반에서 사용하는 인스턴스를 하나만 구현하는 방식



static 응용 : 싱글톤 패턴

- **Singleton 패턴으로 회사 클래스 구현하기**

  1. 생성자를 private으로 만들기

  2. static으로 유일한 인스턴스 생성하기 – getInstance() 메서드

```
public class Company {
    private static Company instance; //instance 객체 선언

    private Company() {}; //외부에서 생성자를 호출 불가

    public static Company getInstance() { //Company로 직접 접근 가능
        if(instance == null) {
            instance = new Company();
        }
        return instance;
    }
}
```

static 응용 : 싱글톤 패턴

- **Singleton 패턴으로 회사 클래스 구현하기**

```
public class CompanyTest {

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

        Company myCompany2 = Company.getInstance();

        //두 변수가 같은 주소인지 확인
        System.out.println(myCompany1==myCompany2);

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

```
true
singleton.Company@7d6f77cc
singleton.Company@7d6f77cc
```

static 응용 : 싱글톤 패턴

자동차 공장이 1개 있고, 이 공장에서 생산되는 자동차는 제작될 때마다 고유 번호가 부여된다. 자동차번호가 1001부터 시작되어 1002, 1003으로 붙도록 자동차 공장 클래스, 자동차 클래스를 만들어 본다.

```
public class CarTest {

    public static void main(String[] args) {
        //자동차 회사 객체 생성
        CarFactory factory = CarFactory.getInstance();

        //자동차 객체 생성
        Car car1 = factory.createCar();
        Car car2 = factory.createCar();
        Car car3 = factory.createCar();

        System.out.println(car1.getCarNumber());
        System.out.println(car2.getCarNumber());
        System.out.println(car3.getCarNumber());
    }
}
```

```
신차 번호: 1001
신차 번호: 1002
신차 번호: 1003
```

static 응용 : 싱글톤 패턴

- Car 클래스

```
public class Car {

    private static int serialNum = 1000; //정적 변수
    private int carNumber;

    public Car() {
        serialNum++;
        carNumber = serialNum;
    }

    public int getCarNumber() {
        return carNumber;
    }
}
```

static 응용 : 싱글톤 패턴

- **CarFactory 클래스**

```
public class CarFactory {
    private static CarFactory instance = new CarFactory();

    private CarFactory() {}

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

    public Car createCar() {  //자동차 생성 메서드
        Car car = new Car();
        return car;
    }
}
```

실습 문제 1 – 싱글톤 패턴

카드 회사에서 카드를 발급할 때마다 카드 고유 번호를 부여해줍니다.
카드 클래스를 만들고, 카드 회사 클래스 CardCompany를 싱글톤 패턴을 사용하여 구현해 보세요.

☞ 실행 결과

```
카드번호: 1001
카드번호: 1002
카드번호: 1003
```

열거 타입(enum)

- **열거 타입**

  한정된 값인 열거 상수 중에서 하나의 상수를 저장하는 타입이다.

```
public enum Season {
    봄,
    여름,
    가을,
    겨울
}
```

Season season = null;

season = Season.여름

열거 타입(enum)

- 열거 타입

```
public class SeasonTest {

    public static void main(String[] args) {
        Season season = null;
        season = Season.여름;

        switch(season) {
        case 봄:
            season = Season.봄;
            break;
        case 여름:
            season = Season.여름;
            break;
        case 가을:
            season = Season.가을;
            break;
        case 겨울:
            season = Season.겨울;
            break;
        }
        System.out.println("현재 계절은 " + season + "입니다.");

        if(season == Season.여름) {
            System.out.println("무더위와 장마가 옵니다.");
        }else {
            System.out.println("무더위와 장마가 별로 없습니다.");
        }
    }
```

열거 타입(enum)

- 열거 타입

```
//1, 2, 3... 순서로 나열됨
enum Week{
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY
}
```

```
Week today = null;  //enum 객체 생성

Calendar cal = Calendar.getInstance(); //Calendar 객체 생성
//요일 가져옴(1-일, 2-월, 3-화, 4-수, 5-목, 6-금, 7-토)
int week = cal.get(Calendar.DAY_OF_WEEK);
//System.out.println(week);

switch(week) {
case 1:
    today = Week.SUNDAY; break;
case 2:
    today = Week.MONDAY; break;
case 3:
    today = Week.TUESDAY; break;
case 4:
    today = Week.WEDNESDAY; break;
case 5:
    today = Week.THURSDAY; break;
case 6:
    today = Week.FRIDAY; break;
case 7:
    today = Week.SATURDAY; break;
```

열거 타입(enum)

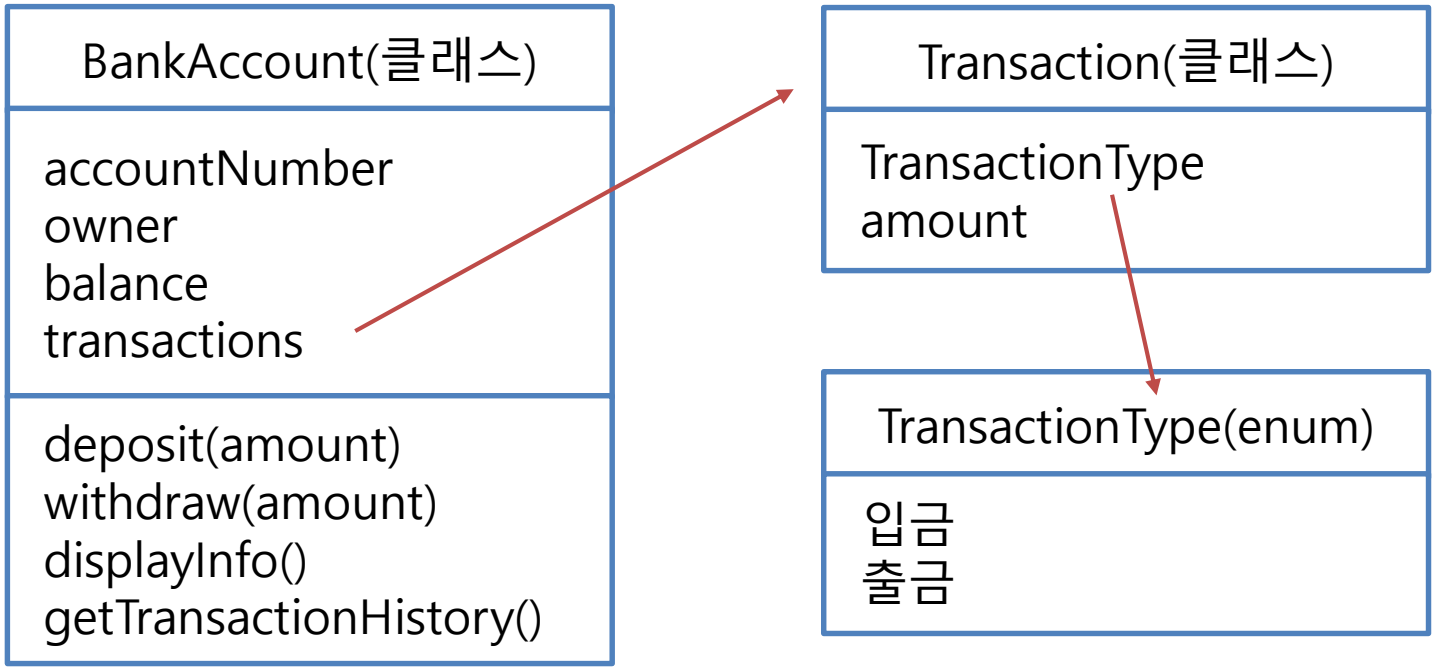
- 열거 타입

```
default:
    System.out.println("요일이 없습니다."); break;
}
System.out.println("Today is " + today);

if(today == Week.SUNDAY) {
    System.out.println("일요일에는 놀러 나갑니다.");
}else {
    System.out.println("평일에는 열심히 코딩합니다.");
}
```

```
5
Today is THURSDAY
평일에는 열심히 코딩합니다.
```

은행 거래 프로젝트

- **BankProject > 클래스 다이어그램**



은행 거래 프로젝트

- **은행 거래 내역 테스트 출력**

```
10000원이 입금되었습니다. 현재 잔액: 10000원
20000원이 입금되었습니다. 현재 잔액: 20000원
5000원이 출금되었습니다. 현재 잔액: 15000원
잔액이 부족합니다.
계좌 정보
    계좌 번호: 101-1234
    계좌주: 이우주
    잔고: 10000
 |입금| 10000원
계좌 정보
    계좌 번호: 102-1234
    계좌주: 정은하
    잔고: 15000
 |입금| 20000원
 |출금| 5000원
계좌 정보
    계좌 번호: 103-1234
    계좌주: 한강
    잔고: 0
거래 내역이 없습니다.
```

은행 거래 프로젝트

- 거래(트랜잭션) 유형 - enum

```
package bankapp1_1;

public enum TransactionType {
    입금,
    출금
}
```

은행 거래 프로젝트

- 거래(트랜잭션) - 클래스

```
public class Transaction {
    TransactionType type;   //거래 유형(enum 참조)
    int amount;             //거래 금액

    public Transaction(TransactionType type, int amount) {
        this.type = type;
        this.amount = amount;
    }
}
```

은행 거래 프로젝트

- **은행계좌(BankAccount) - 클래스**

```
public class BankAccount {
    private String accountNumber;    //계좌 번호
    private String owner; //계좌주
    private int balance;  //잔고
    Transaction[] transactions;

    //생성자
    public BankAccount(String accountNumber, String owner) {
        this.accountNumber = accountNumber;
        this.owner = owner;
        this.balance = 0;
        transactions = new Transaction[100];
    }
```

은행 거래 프로젝트

- 거래(transaction) 추가하기

```
//거래 추가
public void addTransaction(TransactionType type, int amount) {
    for(int i = 0; i < transactions.length; i++) {
        if(transactions[i] == null) {
            //트랜잭션 인스턴스 생성
            transactions[i] = new Transaction(type, amount);
            break; //인스턴스 저장후 즉시 빠져나옴
        }
    }
}
```

은행 거래 프로젝트

- 거래 내역(history) 조회

```
public void getTransactionHistory() { //거래 내역 조회
    boolean hasTransaction = false; //토글 변수
    for(int i = 0; i < transactions.length; i++) {
        if(transactions[i] != null) {
            hasTransaction = true;
            System.out.print(" |" + (transactions[i].type == TransactionType.입금 ?
                    "입금" : "출금"));
            System.out.println("| " + transactions[i].amount + "원");
        }
    }
    if(!hasTransaction) {
        System.out.println("거래 내역이 없습니다.");
    }
}
```

은행 거래 프로젝트

- 입금(deposit)

```
public void deposit(int amount) { //입금
    if(amount <= 0) {
        System.out.println("유효한 금액을 입력하세요.");
    }
    else {
        this.balance += amount; //잔액 + 입금액
        System.out.println(amount + "원이 입금되었습니다. 현재 잔액: " +
                this.balance + "원");

        addTransaction(TransactionType.입금, amount); //입금 거래
    }
}
```

은행 거래 프로젝트

- 출금(withdraw)

```
public void withdraw(int amount) { //출금
    if(amount <= 0) {
        System.out.println("유효한 금액을 입력하세요.");
    }else if(amount > balance) {
        System.out.println("잔액이 부족합니다.");
    }else {
        this.balance -= amount; //잔액 - 출금액
        System.out.println(amount + "원이 출금되었습니다. 현재 잔액: " +
                this.balance + "원");

        addTransaction(TransactionType.출금, amount); //출금 거래
    }
}
```

은행 거래 프로젝트

- **계좌 정보 출력**

```
public void displayInfo() { //계좌 정보 출력
    System.out.println("계좌 정보");
    System.out.println("    계좌 번호: " + accountNumber);
    System.out.println("    계좌주: " + owner);
    System.out.println("    잔고: " + balance);
}
```

은행 거래 프로젝트

- **Main 테스트**

```
BankAccount[] accounts = new BankAccount[3];

//계좌 인스턴스 생성
accounts[0] = new BankAccount("101-1234", "이우주");
accounts[1] = new BankAccount("102-1234", "정은하");
accounts[2] = new BankAccount("103-1234", "한강");

//입금
accounts[0].deposit(10000);
accounts[1].deposit(20000);

//출금
accounts[1].withdraw(5000);
accounts[1].withdraw(30000);

//정보 출력
for(int i = 0; i < accounts.length; i++) {
    if(accounts[i] != null) {
        accounts[i].displayInfo();
        accounts[i].getTransactionHistory();
    }
}
```