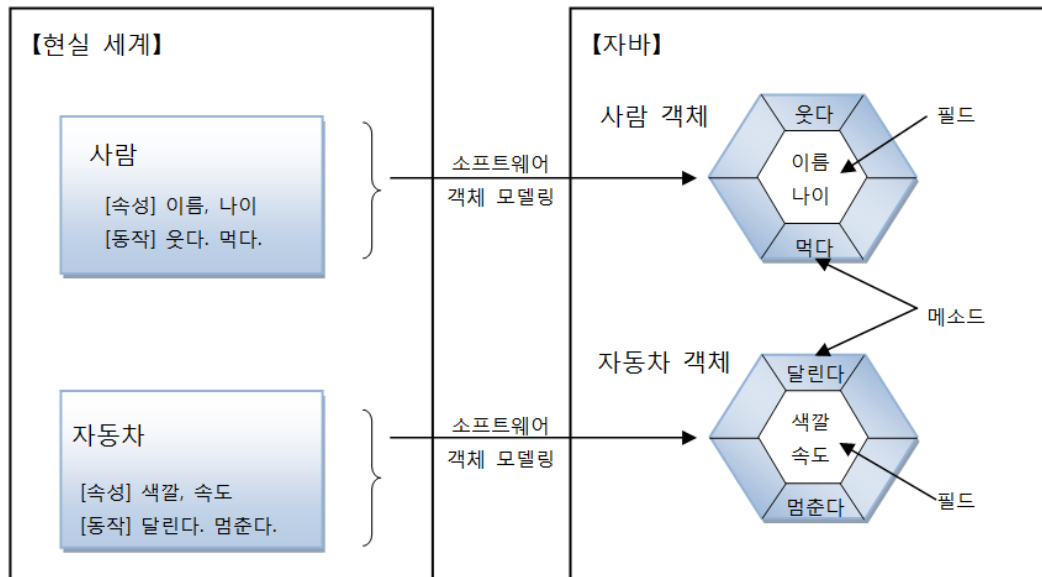


■ 객체 지향 프로그래밍

- OOP: Object Oriented Programming
- 부품 객체를 먼저 만들고 하나씩 조립해 완성된 프로그램을 만드는 기법

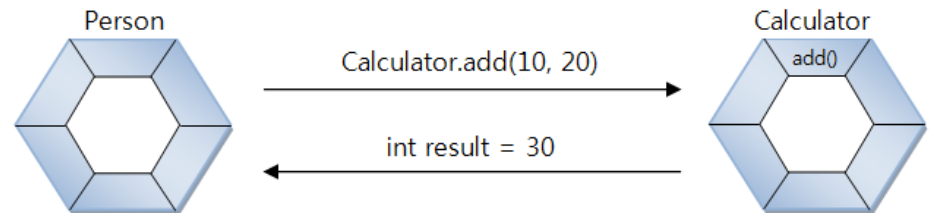
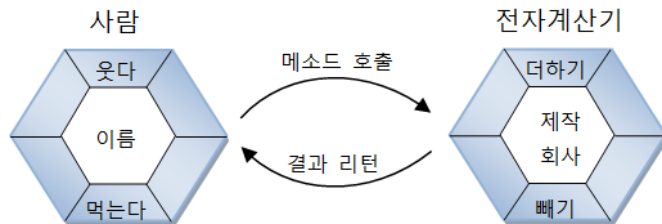
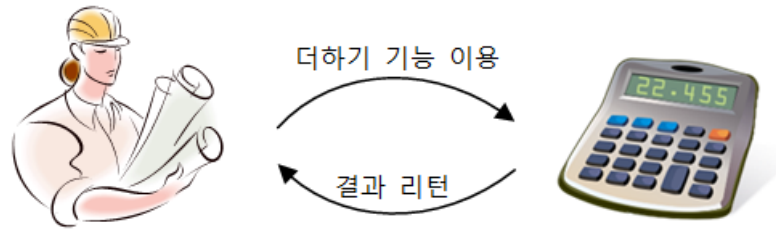
■ 객체(Object)란?

- 물리적으로 존재하는 것 (자동차, 책, 사람)
- 추상적인 것(회사, 날짜) 중에서 자신의 속성과 동작을 가지는 모든 것
- 객체는 필드(속성) 과 메소드(동작)로 구성된 자바 객체로 모델링 가능



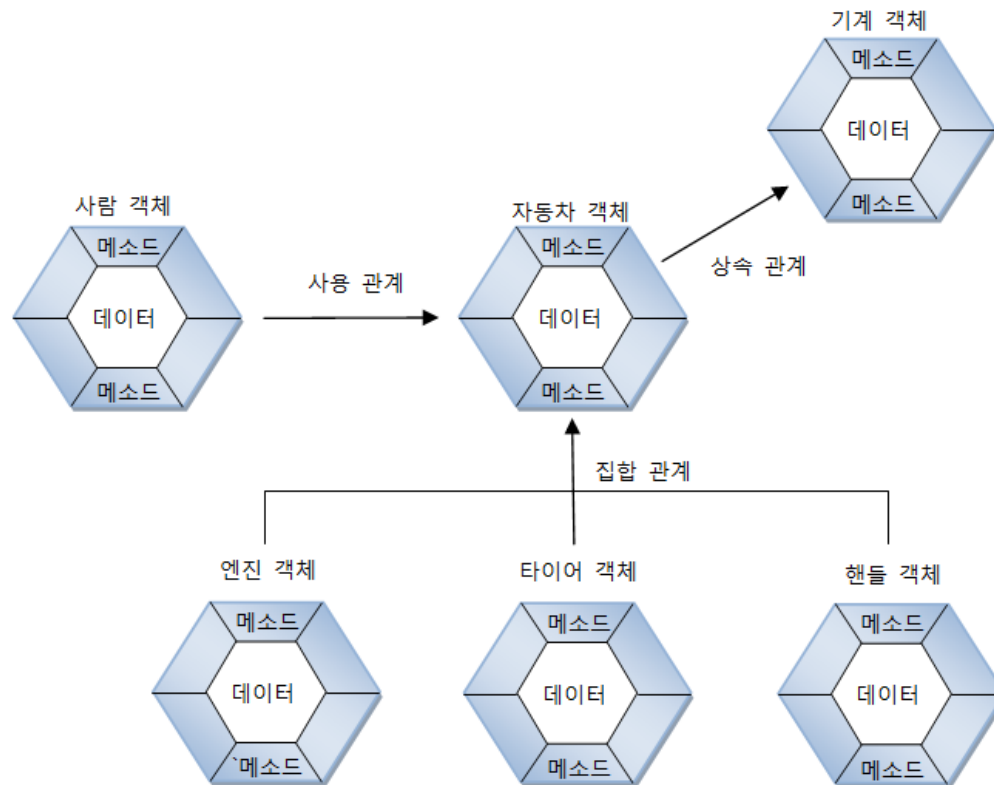
■ 객체의 상호 작용

- 객체들은 서로 간에 기능(동작)을 이용하고 데이터를 주고 받음



■ 객체간의 관계

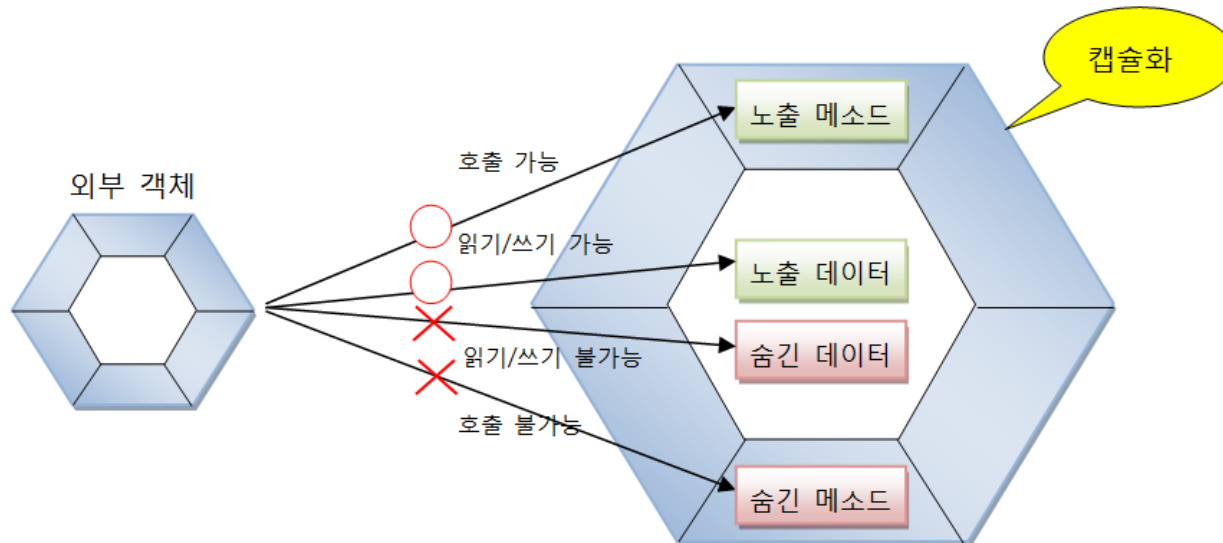
- 객체 지향 프로그램에서는 객체는 다른 객체와 관계를 맺음
- 관계의 종류
 - 집합 관계: 완성품과 부품의 관계
 - 사용 관계: 객체가 다른 객체를 사용하는 관계
 - 상속 관계: 종류 객체와 구체적인 사물 객체 관계



■ 객체 지향 프로그래밍 특징

● 캡슐화

- 객체의 필드, 메소드를 하나로 묶고, 실제 구현 내용을 감추는 것
- 외부 객체는 객체 내부 구조를 알지 못하며 객체가 노출해 제공하는 필드와 메소드만 이용 가능
- 필드와 메소드를 캡슐화하여 보호하는 이유는
외부의 잘못된 사용으로 인해 객체가 손상되지 않도록
- 자바 언어는 캡슐화된 멤버를 노출시킬 것인지 숨길 것인지 결정하기 위해 접근 제한자(Access Modifier) 사용



■ 객체 지향 프로그래밍 특징

● 상속

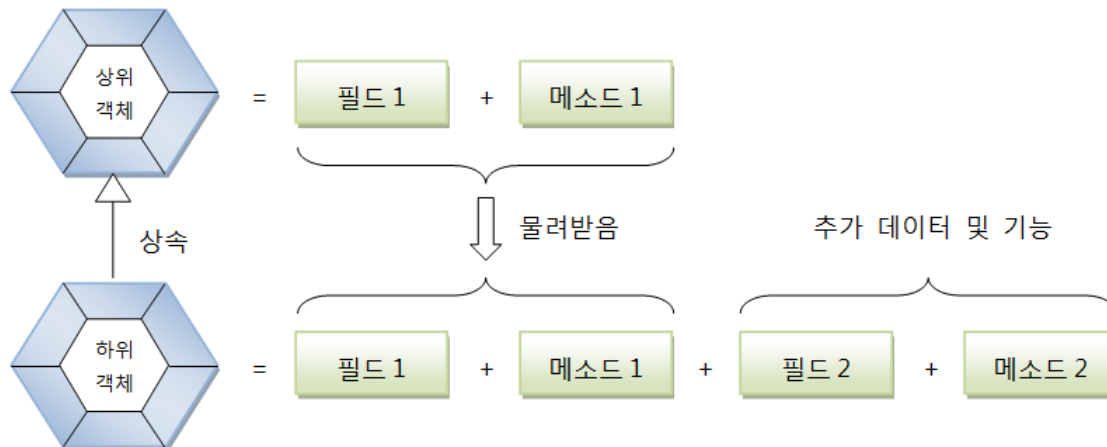
- 상위(부모) 객체의 필드와 메소드를 하위(자식) 객체에게 물려주는 행위
- 하위 객체는 상위 객체를 확장해서 추가적인 필드와 메소드를 가질 수 있음
- 상속 대상: 필드와 메소드
- 상속의 효과

상위 객체를 재사용해서 하위 객체를 빨리 개발 가능

반복된 코드의 중복을 줄임

유지 보수성 편리성 제공

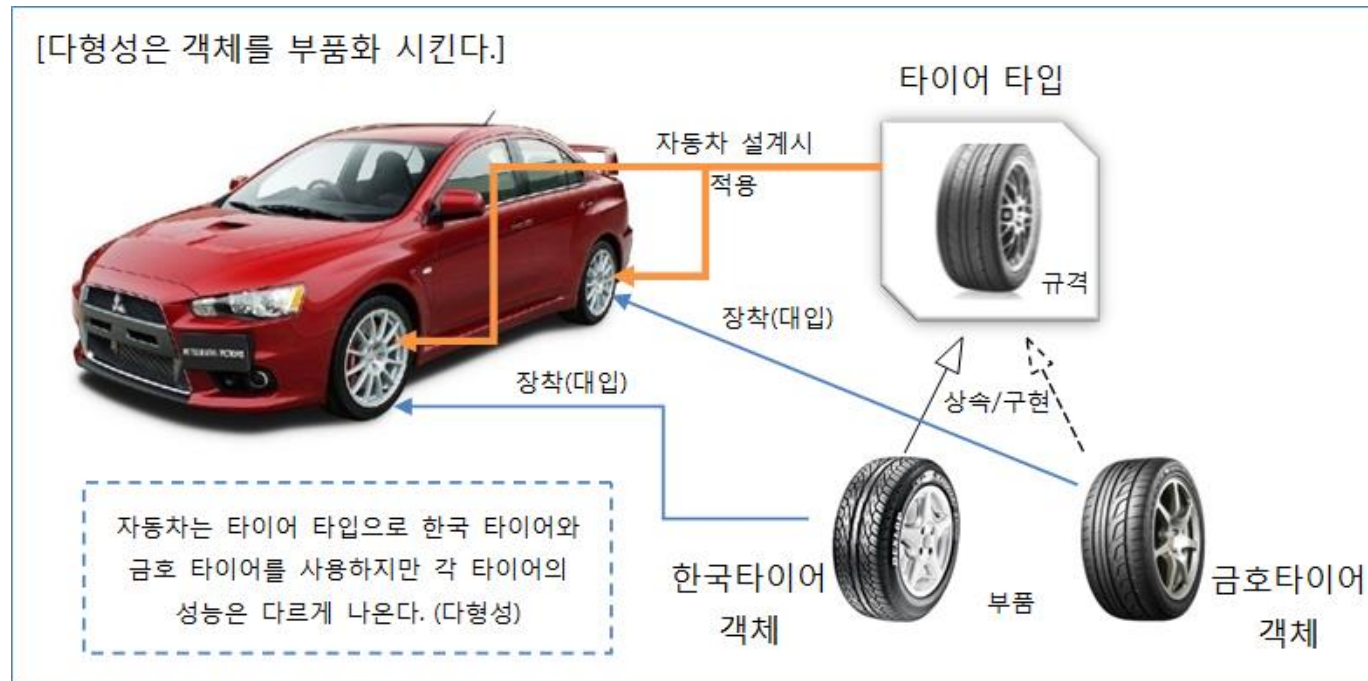
객체의 다형성 구현



■ 객체 지향 프로그래밍 특징

● 다형성 (Polymorphism)

- 같은 타입이지만 실행 결과가 다양한 객체를 대입할 수 있는 성질



■ 객체와 클래스

- 현실세계 : 설계도 → 객체
- 자바 : 클래스 → 인스턴스
- 클래스에는 객체를 생성하기 위한 필드와 메소드가 정의
- 클래스로부터 만들어진 객체를 해당 클래스의 인스턴스(instance) 라고 함
- 하나의 클래스로부터 여러 개의 인스턴스를 만들 수 있음

[객체를 생성하는 순서]



개발자

설계



클래스(설계도)

인스턴스화



인스턴스(객체)

■ 클래스 구성 요소

- 필드(Field)
- 생성자(Constructor)
- 메소드(Method)

- 필드(Field)
객체의 데이터가 저장되는 곳
- 생성자(Constructor)
객체 생성시 초기화 역할 담당
- 메소드(Method)
객체의 동작에 해당하는 실행 블록

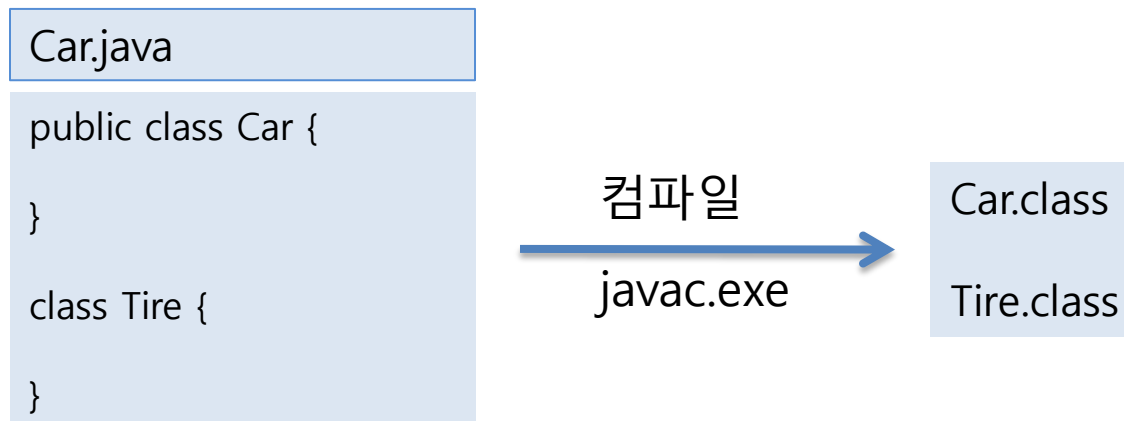
```
public class ClassName {  
  
    //필드  
    int fieldName;  
  
    //생성자  
    ClassName() { ... }  
  
    //메소드  
    void methodName() { ... }  
  
}
```


■ 클래스 선언과 컴파일

- 소스 파일 생성: 클래스이름.java (대소문자 주의)
- 소스 작성



- 소스 파일당 하나의 클래스를 선언하는 것이 관례
 - 두 개 이상의 클래스도 선언 가능
 - 소스 파일 이름과 동일한 클래스만 public으로 선언 가능
 - 선언한 개수만큼 바이트 코드 파일이 생성



■ 클래스 용도

- 라이브러리(API: Application Program Interface) 용
 - 자체적으로 실행되지 않음
 - 다른 클래스에서 이용할 목적으로 만든 클래스
- 실행용
 - main() 메소드를 가지고 있는 클래스로 실행할 목적으로 만든 클래스

1개의 애플리케이션 = (1개의 실행클래스) + (n개의 라이브러리 클래스)

■ new 연산자

- 객체 생성 역할

```
new 클래스();
```

- 클래스() 는 생성자를 호출하는 코드
- 생성된 객체는 힙 메모리 영역에 생성
- new 연산자는 객체를 생성한 후 객체 생성 주소 리턴

■ 인스턴스의 생성과 사용

▶ 인스턴스의 생성방법

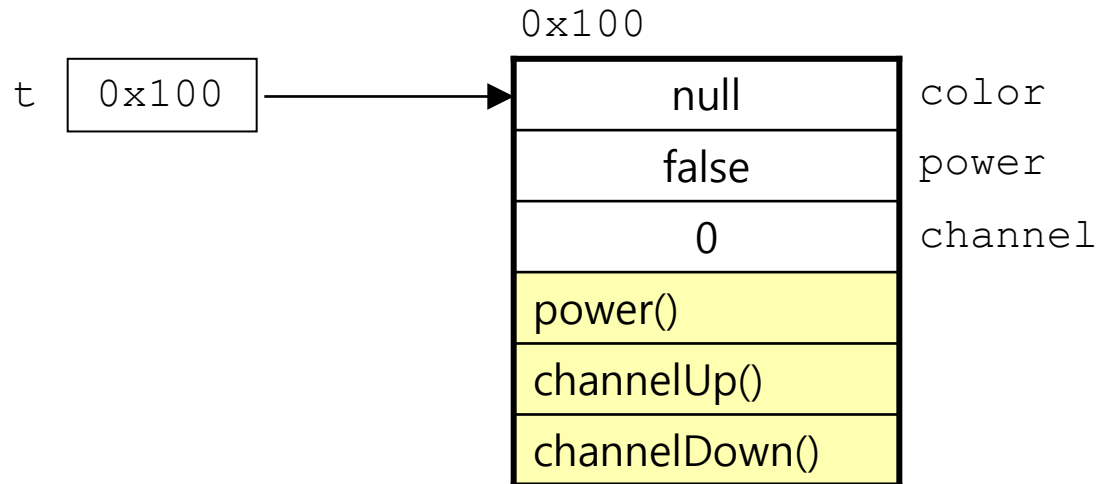
클래스명 참조변수명;
참조변수명 = new 클래스명();

```
public class Tv {  
    String color;  
    boolean power;  
    int channel;  
  
    void power() { power = !power; }  
    void channelUp() { channel++; }  
    void channelDown() { channel--; }  
}
```

```
Tv t;
```

```
t = new Tv();
```

```
Tv t = new Tv();
```



■ 인스턴스의 생성과 사용

```
Tv t;
```

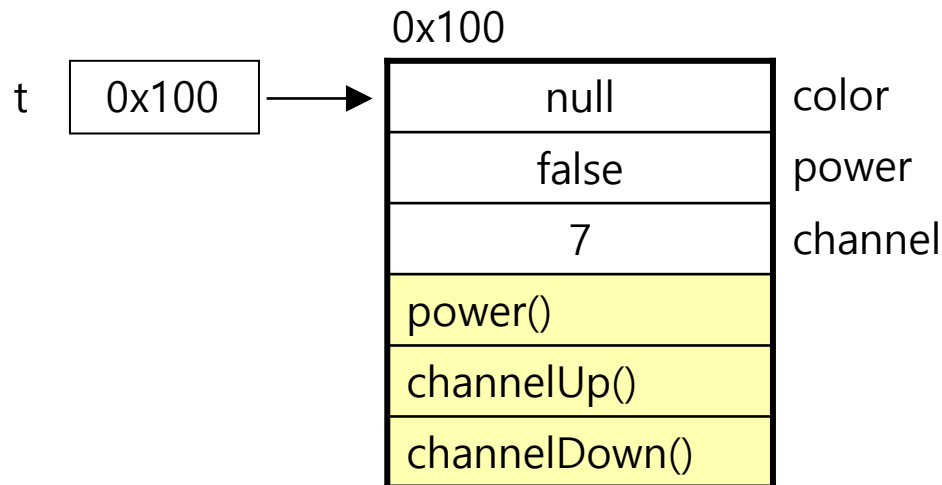
```
t = new Tv();
```

```
t.channel = 7;
```

```
t.channelDown();
```

```
System.out.println(t.channel);
```

```
public class Tv {  
    String color;  
    boolean power;  
    int channel;  
  
    void power() { power = !power; }  
    void channelUp() { channel++; }  
    void channelDown() { channel--; }  
}
```

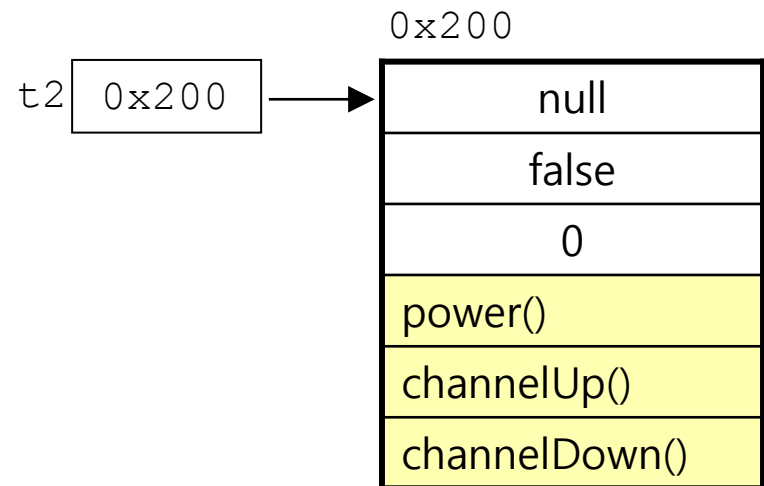
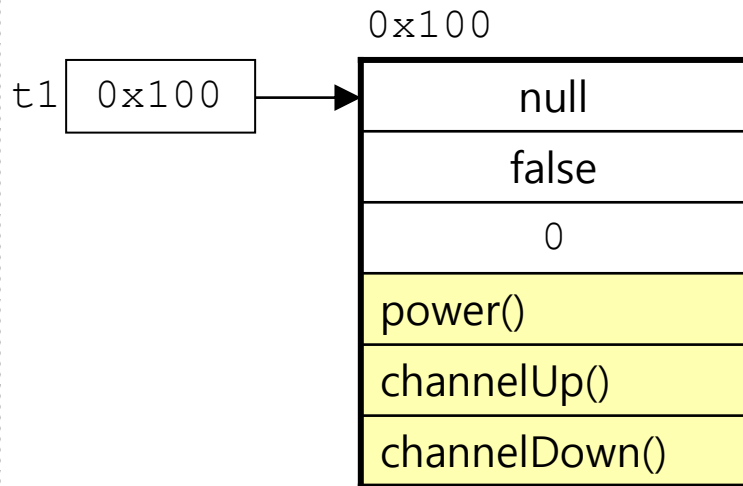


■ 인스턴스의 생성과 사용

```
Tv t1 = new Tv();
```

```
Tv t2 = new Tv();
```

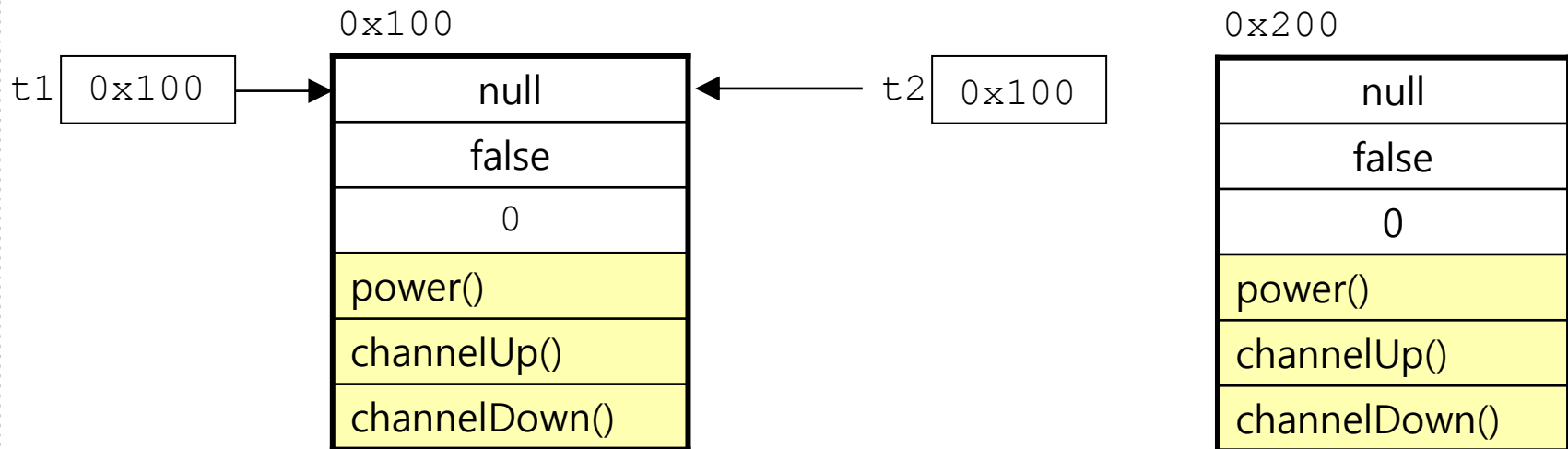
```
public class Tv {  
    String color;  
    boolean power;  
    int channel;  
  
    void power() { power = !power; }  
    void channelUp() { channel++; }  
    void channelDown() { channel--; }  
}
```



■ 인스턴스의 생성과 사용

```
Tv t1 = new Tv();  
Tv t2 = new Tv();  
t2 = t1;
```

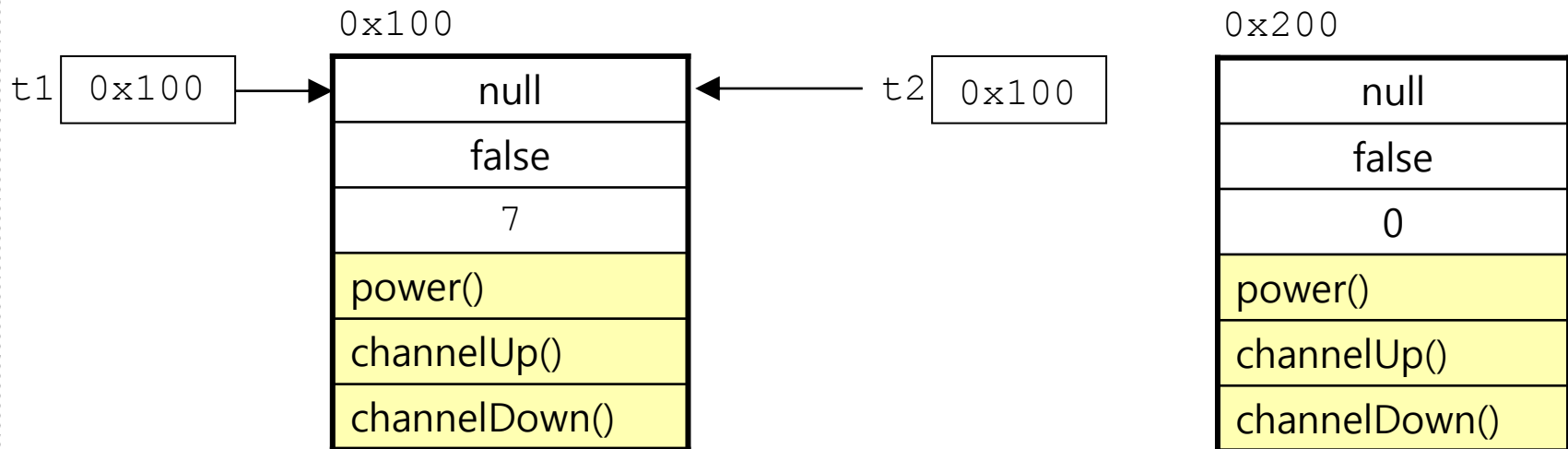
```
public class Tv {  
    String color;  
    boolean power;  
    int channel;  
  
    void power() { power = !power; }  
    void channelUp() { channel++; }  
    void channelDown() { channel--; }  
}
```



■ 인스턴스의 생성과 사용

```
Tv t1 = new Tv();  
Tv t2 = new Tv();  
t2 = t1;  
t1.channel = 7;
```

```
public class Tv {  
    String color;  
    boolean power;  
    int channel;  
  
    void power() { power = !power; }  
    void channelUp() { channel++; }  
    void channelDown() { channel--; }  
}
```



■ 인스턴스의 생성과 사용

```
Tv t1 = new Tv();
```

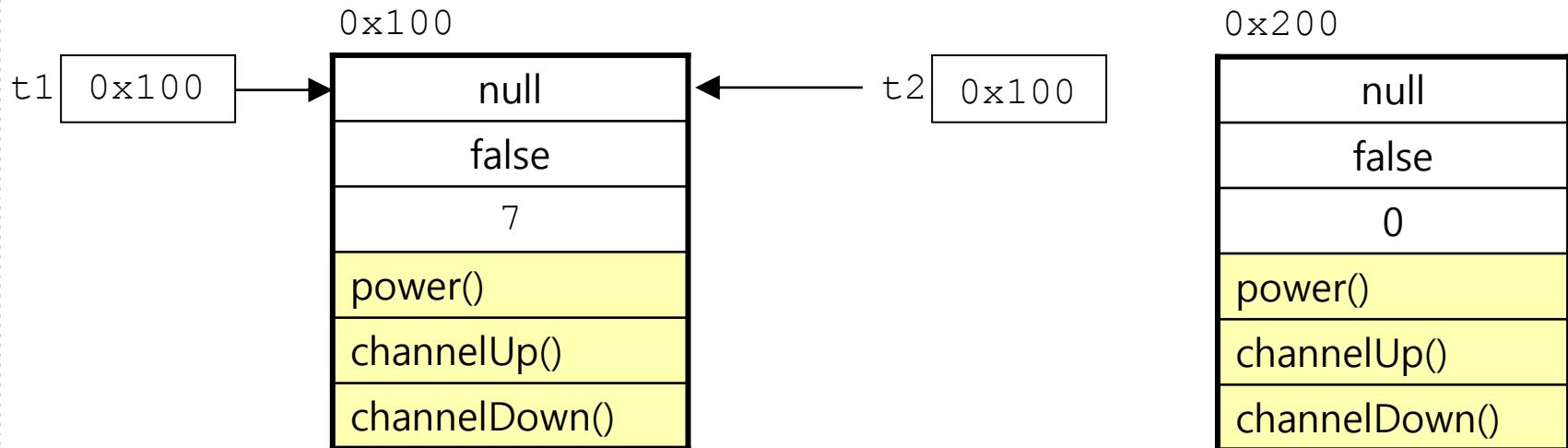
```
Tv t2 = new Tv();
```

```
t2 = t1;
```

```
t1.channel = 7;
```

```
System.out.println(t1.channel); ➔ 7
```

```
System.out.println(t2.channel); ➔ 7
```



■ 인스턴스 생성 및 사용 – 1 (1 / 2)

```
public class Person {  
    String name; // 이름  
    float height; // 키  
    float weight; // 몸무게  
}
```

```
public class PersonMain {  
    public static void main(String[] args) {  
        Person p = new Person();  
        p.name = "김덕호";  
        p.height = 170.4f;  
        p.weight = 70.0f;  
  
        Person p2 = new Person();  
        p2.name = "양준호";  
        p2.height = 171.1f;  
        p2.weight = 75.4f;  
    }  
}
```

■ 인스턴스 생성 및 사용 – 1 (2 / 2)

```
System.out.println(p.name);  
System.out.println(p2.name);
```

```
Person[] persons = {p, p2};  
for(int i = 0; i < persons.length; i++) {  
    System.out.println("이름 : " + persons[i].name +  
                        ", 키 : " + persons[i].height +  
                        ", 몸무게 : " + persons[i].weight);  
}  
}  
}
```

■ 인스턴스 생성 및 사용 – 2 (1 / 2)

```
public class SmartPhone {  
    String company; // 제조사  
    String name; // 단말기명  
    float size; // 핸드폰 사이즈 ex)4.0, 5.0, 5.3  
    int price; // 가격  
  
    public void volumeUp() {  
        System.out.println(name + " 소리 올림");  
    }  
    public void volumeDown() {  
        System.out.println(name + " 소리 내림");  
    }  
    public void powerOn() {  
        System.out.println(name + " 전원 켜짐");  
    }  
    public void powerOff() {  
        System.out.println(name + " 전원 꺼짐");  
    }  
}
```

■ 인스턴스 생성 및 사용 – 2 (2 / 2)

```
public class SmartPhoneMain {  
    public static void main(String[] args) {  
        SmartPhone sp = new SmartPhone();  
        sp.company = "삼성";  
        sp.name = "갤럭시";  
        sp.size = 4.0f;    sp.price = 100000;  
  
        SmartPhone sp2 = new SmartPhone();  
        sp2.company = "LG";  
        sp2.name = "넥서스";  
        sp2.size = 5.0f;    sp2.price = 200000;  
  
        sp.volumeUp();  
        sp.powerOff();  
  
        sp2.volumeDown();  
        sp2.powerOn();  
    }  
}
```

■ 변수의 초기화

- 변수를 선언하고 처음으로 값을 저장하는 것
- 멤버변수(인스턴스변수, 클래스변수)와 배열은 각 타입의 기본값으로 자동초기화되므로 초기화를 생략할 수 있다.
- 지역변수는 사용전에 반드시 초기화를 해주어야 한다.

■ 기본값 및 초기화

자료형	기본값	초기화
boolean	false	boolean isEmpty = false;
char	'\u0000'	char gender = ' ' or 'M'
byte	0	byte b = 10;
short	0	short s = 10;
int	0	int i = 10; (8 / 10 / 16진수)
long	0L	long l = 10L;
float	0.0F	float f = 10F;
double	0.0D 또는 0.0	double d = 10 or 10D;
참조 자료형	null	UserType u = null or new UserType();

■ 변수의 초기화

```
public class Initialize1 {  
    // 멤버(전역) 변수  
    int x;  
    int y = x;  
  
    public static void main(String[] args) {  
        // 메소드(지역) 변수  
        int a;  
        int b = a;  
    }  
}
```


■ 멤버변수(인스턴스변수)의 초기화

● 인스턴스변수의 초기화 방법

- 명시적 초기화(explicit initialization)

```
class Car {  
    int door = 4;           // 기본형 (primitive type) 변수의 초기화  
    Engine e = new Engine(); // 참조형 (reference type) 변수의 초기화  
  
    //...  
}
```

- 생성자(constructor)

```
Car(String color, String gearType, int door){  
    this.color = color;  
    this.gearType = gearType;  
    this.door = door;  
}
```

- 초기화 블록(initialization block)
 - 인스턴스 초기화 블록 : { }
 - 클래스 초기화 블록 : static { }

■ 멤버변수(인스턴스변수)의 초기화

```
public class Initialize2 {  
    // 1. 명시적 초기화  
    int a = 10;  
    int b = 20;  
  
    // 2. 초기화 블록을 이용  
    {  
        System.out.println("= 초기화 블록 =");  
        System.out.println(a);  
        System.out.println(b);  
        a = 40;  
        b = 50;  
    }  
}
```

// 3. 생성자를 이용

```
public Initialize2() {  
    System.out.println("= 생성자 =");  
    System.out.println(a);  
    System.out.println(b);  
    a = 20;  
    b = 30;  
}  
  
public static void main(String[] args) {  
    Initialize2 i = new Initialize2();  
  
    System.out.println("= 객체 생성 후 =");  
    System.out.println(i.a);  
    System.out.println(i.b);  
}  
}
```

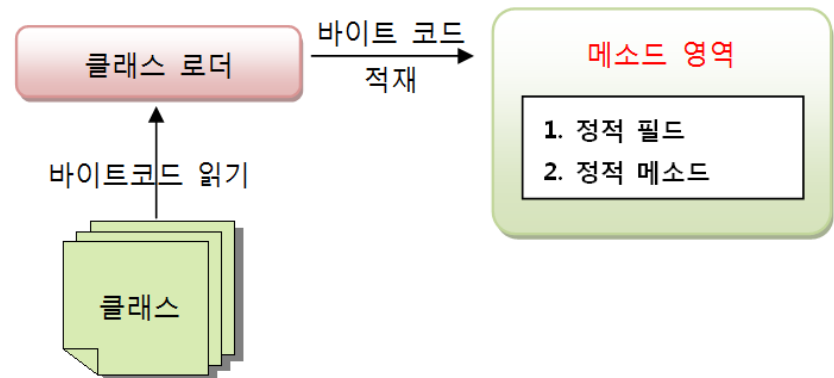
■ static 변수 (클래스 변수)

- 클래스에 고정된 필드와 메소드 - static 변수, static 메소드
- 정적 변수는 클래스에 소속
 - 객체 내부에 존재하지 않고, 메소드 영역에 존재
 - 정적 변수는 객체를 생성하지 않고 클래스로 바로 접근해 사용

■ static 변수 선언

- 변수 또는 메소드 선언할 때 **static** 키워드 붙임

```
public class 클래스 {  
    //정적 필드  
    static 타입 필드 [= 초기값];  
  
    //정적 메소드  
    static 리턴타입 메소드( 매개변수선언, ... ) { ... }  
}
```



■ static 변수 사용

● 클래스 이름과 함께 도트(.) 연산자로 접근

클래스.필드;

클래스.메소드(매개값, ...);

```
public class Calculator {  
    static double pi = 3.14159;  
    static int plus(int x, int y) { ... }  
    static int minus(int x, int y) { ... }  
}
```

[바람직한 사용]

```
double result1 = 10 * 10 * Calculator.pi;  
int result2 = Calculator.plus(10, 5);  
int result3 = Calculator.minus(10, 5);
```

[바람직하지 못한 사용]

```
Calculator myCalcu = new Calculator();  
double result1 = 10 * 10 * myCalcu.pi;  
int result2 = myCalcu.plus(10, 5);  
int result3 = myCalcu.minus(10, 5);
```

■ static 변수 사용 – 1 (1 / 2)

```
public class Static {  
    static char color = 'R';  
    static int num = 0;  
  
    static void staticMethod() {  
        System.out.println("static method");  
        num++;  
    }  
  
    void instanceMethod() {  
        System.out.println("instance method");  
    }  
}
```

■ static 변수 사용 – 1 (2 / 2)

```
public class StaticMain {  
    public static void main(String[] args) {  
        // 클래스명을 이용하여 접근  
        System.out.println(Static.color);  
  
        Static.staticMethod();  
        System.out.println(Static.num);  
  
        // 객체를 생성시킨 후 접근  
        Static s = new Static();  
        s.instanceMethod();  
        s.num++;  
  
        // 아래처럼 사용은 가능하지만 추천하지 않음  
        System.out.println(s.num);  
    }  
}
```

■ static 변수 사용 – 2

```
public class RandomExam {  
    public static void main(String[] args) {  
        double num = Math.random();  
        System.out.println(num);  
    }  
}
```

■ 인스턴스 선언 vs static 선언의 기준

● 변수

- 객체마다 가지고 있어야 할 데이터 ➔ 인스턴스 변수
- 공용적인 데이터 ➔ static 변수

```
public class Calculator {  
    String color;           //계산기 별로 색깔이 다를 수 있다.  
    static double pi = 3.14159; //계산기에서 사용하는 파이(π)값은 동일하다.  
}
```

● 메소드

- 인스턴스 필드로 작업해야 할 메소드 ➔ 인스턴스 메소드
- 인스턴스 필드로 작업하지 않는 메소드 ➔ static 메소드

```
public Calculator {  
    String color;  
    void setColor(String color) { this.color = color; }  
    static int plus(int x, int y) { return x + y; }  
    static int minus(int x, int y) { return x - y; }  
}
```


■ final 키워드의 용도

- final 필드: 수정 불가 필드
- final 클래스: 부모로 사용 불가한 클래스
- final 메소드: 자식이 재정의할 수 없는 메소드

■ 상속할 수 없는 final 클래스

- 자식 클래스 만들지 못하도록 final 클래스로 생성

```
public final class 클래스 { ... }
```

```
public final class String { .. }
```

```
public class NewString extends String { ... }
```

■ 오버라이딩 불가한 final 메소드

- 자식 클래스가 재정의 못하도록 부모 클래스의 메소드를 final로 생성

■ final 필드 (상수)

- 최종적인 값을 갖고 있는 필드 = 값을 변경할 수 없는 필드
- final 필드의 딱 한번의 초기값 지정 방법
 - 필드 선언 시
 - 생성자

```
public class Person {  
    final String nation = "Korea";  
    final String ssn;  
    String name;  
  
    public Person(String ssn, String name) {  
        this.ssn = ssn;  
        this.name = name;  
    }  
}
```

■ final 필드 (상수)

● 상수 = 정적 final 필드

- final 필드 : 객체마다 가지는 불변의 인스턴스 필드

- 상수(static final) : 객체마다 가지고 있지 않음

메소드 영역에 클래스 별 로 관리되는 불변의 정적 필드
공용 데이터로서 사용

● 상수 이름은 전부 대문자로 작성

● 다른 단어가 결합되면 _ 로 연결

■ final 필드 (상수) 사용 - 1

```
public class Constants {  
    public static final int HOBBY_SWIMMING = 0;  
    public static final int HOBBY_TENNIS = 1;  
    public static final int HOBBY_EAT = 2;  
}
```

```
public class ConstantsMain {  
    public static void main(String[] args) {  
        System.out.println(Constants.HOBBY_SWIMMING);  
    }  
}
```

■ final 필드 (상수) 사용 - 2

```
public class Window {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("윈도우");  
  
        frame.setBounds(200, 200, 200, 200);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        frame.setVisible(true);  
    }  
}
```