● **메서드 오버로딩(method overloading/간단히 오버로딩)**: 한 클래스 내에 같은 이름의 메서드를 여러 개 정의하는 것. 자바에서는 한 클래스 내에 이미 사용하려는 이름과 같은 이름을 가진 메서드가 있더라도 매게변수의 개수 또는 타입이 다르면, 같은 이름을 사용해서 메서드를 정의할 수 있다. 단, 오버로딩된 메서드들은 매개변수에 의해서만 구현될 수 있으므로 반환 타입은 오버로딩을 구현하는데 아무런 영향을 주지 못한다. 오버로딩은 하나의 메서드 이름으로 다든 타입의 매게변수를 사용해 같은 기능(메서드)를 이용할 수 있기 때문에 메서드를 이름을 절약할 수 있고 사용자 입장에서 메서드 이름을 기억하기 쉬워 기능(메서드)를 사용하기 쉽다는 장점이 있다.

☞ Ex) println: 하나의 이름을 가졌지만 실제로는 println메서드를 호출할 때 매개변수로 지정하는 값의 타입에 따라서 호출되는 println메서드가 달라진다.

· 매개변수의 이름만 다를 뿐 매개변수의 타입이 같으면 오버로딩이 성립되지 않아 컴파일 시 --is already defined라고 메시지가 나타나며 에러가 발생한다.

· 리턴 타입만 다를 경우에도 오버로딩은 성립되지 않는다.

· 만약 두 메서드가 두 개의 다든 타입의 매개변수가 하나씩 선언되어 있지만 서로 순서가 다든 경우에는 오버로딩으로 간주한다. (ex. int a, long b // long a, int b) 단, 이 경우에는 3,3을 인자로 호출할 경우 두 메서드 중 어느 메서드가 호출될 것인지 알 수 없기 때문에 컴파일 에러가 발생하기 때문에 단지 매개변수의 순서만을 다르게 하여 오버로딩을 구현할 때는 주의해야 한다.

● **가변인자(varargs: variable arguments)**: 메서드의 매개변수의 개수를 동적으로 지정해주는 기능. 가변인자는 **'타입...변수명'**으로 선언하며, 가변인자 외에도 매개변수가 더 있다면, 가변인자를 매개변수 중에서 제일 마지막에 선언해야 한다. 그렇지 않으면, 가변인자인지 아닌지 구분할 방법이 없어 컴파일 에러가 발생한다. 가변인자를 사용하면 **인자가 아예 없어도 되고 배열도 인자로 사용할 수 있다**. 가변인자가 선언된 메서드를 호출할 때마다 배열이 새로 생성되고, 내부적으로 배열을 이용하기 때문이다. 때문에 가변인자는 비효율적이라 꼭 필요한 경우에만 가변인자를 사용해야 한다. 매개변수의 타입을 배열로 하면 반드시 인자를 지정해줘야 하기 때문에, 인자를 생략할 수 없어서 인자를 null이나 길이가 0인 배열로 지정해주어야 하는 불편함이 있는 것이 가변인자와의 차이이다.

```java
// 여러 문자를 하나로 결합하여 반환하는 concatenate 메서드를 작성하려면,
// 원래대로라면 매번 원하는 매개변수의 개수를 다르게 해서 여러 개의 메서드를 작성해야 하지만,
// 가변인자를 사용하면 메서드 하나로 간단히 작성할 수 있다.

class Ex {
    public static void main(String[] args) {
        System.out.println(concatenate("안녕"));
        System.out.println(concatenate("안녕", "만나서", "반가워"));
        System.out.println(concatenate());
        System.out.println(concatenate("끝"));
        String[] strArr = {"나는", "에스포레소이다"};
        System.out.println(concatenate(strArr));
    }

    static String concatenate(String... sen) { // 가변인자 sen. 인자로 배열을 생성한다.
        String result = "";

        for (String str : sen) {
            result += str;
        }

        return result;
    }
}
```

```
"C:\Program Files\Java\jdk-17.0.1\
안녕
안녕만나서반가워

끝
나는에스포레소이다

Process finished with exit code 0
```

- 가변인자를 선언한 메서드를 오버로딩 하면, 메서드를 호출했을 때 구별되지 못하는 경우가 발생하기 쉽기 때문에 주의해야 한다. 가능하면 가변인자를 사용한 메서드는 오버로딩 하지 않는 것이 좋다.

```java
// 매개변수로 입력된 문자열에 구분자를 사이에 포함시켜 결합해서 반환하는 concatenate 메서드
import java.util.*;

class Ex {
    public static void main(String[] args) {
        String[] strArr = {"100", "200", "300"};

        System.out.println(concatenate("", "100", "200", "300"));
        System.out.println(concatenate("-", strArr));
        System.out.println(concatenate(",", new String[]{"1", "2", "3"}));
        System.out.println("[" + concatenate(",", new String[0]) + "]");
        System.out.println("[" + concatenate(",") + "]");

    }

    static String concatenate(String delim, String... sen) {
        String result = "";

        for (String str : sen) {
            result += str + delim;
        }

        return result;
    }

//    static String concatenate(String...args) {
//        return concatenate("", args);
//    }
    // 첫번째로 오는 인자가 첫번째 메서드의 delim인지 아니면 두번째 메서드의 문자 중 하나인지
    // 구분되지 않기 때문에 컴파일 시 에러가 발생한다.
}
```

```
"C:\Program Files\Java\jdk-17.0.1\b
100200300
100-200-300-
1,2,3,
[]
[]

Process finished with exit code 0
```

● **생성자(Constructor)** : 인스턴스가 생성될 대 호출되는 '인스턴스 초기화 메서드'. 인스턴스 초기화란, 인스턴스 변수들을 초기화하는 것을 뜻한다. 인스턴스 변수의 초기화 작업에 주로 사용되며, 인스턴스 생성 시에 실행되어야 할 작업을 위해서도 사용된다. 생성자 역시 클래스 내에 선언된다. 생성자의 이름은 클래스 이름과 같아야 하며, 생성자는 리턴 값이 없다. 생성자도 오버로딩이 가능하므로 하나의 클래스에 여러 개의 생성자가 존재할 수 있다. 단, 연산자 new가 인스턴스를 생성하는 것이지 생성자가 인스턴스를 생성하는 것이 아니다. 생성자는 단순히 인스턴스변수들의 초기화에 사용되는 특별한 메서드일 뿐이다.

▶ **기본 생성자(default constructor)**: 컴파일 할 때, 소스파일(.java)의 클래스에 생성자가 하나도 정의되어 있지 않은 경우, 컴파일러는 클래스이름() {} 과 같이 매개변수도 없고 아무런 내용도 없는 기본 생성자를 추가하여 컴파일하기 때문에, 클래스에 생성자를 정의하지 않고도 인스턴스를 생성할 수 있다. 특별히 인스턴스 초기화 작업이 요구되어지지 않는다면 생성자를 정의하지 않고 컴파일러가 제공하는 기본 생성자를 사용하는 것도 좋다. 클래스의 접근 제어자(Access Modifier)가 public인 경우에는 기본 생성자로 'public 클래스이름() {}'이 추가된다. 기본 생성자가

컴파일러에 의해 추가되는 경우는 클래스에 정의된 생성자가 하나도 없을 때 뿐이다. 만약 생성자가 따로 있는 상황에서 매개변수가 하나도 없는 생성자도 원한다면 사용자가 직접 클래스이름() {} 으로 생성자를 작성해 주어야 한다.

```
// Card클래스의 인스턴스를 생성하는 코드
Card c = new Card();

// 1. 연산자 new에 의해서 메모리(heap)에 Card클래스의 인스턴스가 생성된다.
// 2. 생성된 Card()가 호출되어 수행된다.
// 3. 연산자 new의 결과로, 생성된 Card인스턴스의 주소가 반환되어 참조변수 c에 저장된다.
```

▶ 매개변수가 있는 생성자: 매개변수를 선언하여 호출 시 값을 넘겨받아서 인스턴스의 초기화 작업에 사용할 수 있다.

```
// Card인스턴스를 생성할 때, 기본생성자 Car()을 사용한다면 인스턴스 생성 후 인스턴스변수들을
// 따로 초기화해야하지만, 매개변수가 있는 생성자 Car(String color, String gearType, int door
// 를 사용한다면 인스턴스를 생성하는 동시에 인스턴스변수들을 원하는 값으로 초기화할 수 있다.

import java.util.*;

class Car {
    String color;
    String gearType;
    int door;

    Car() {};

    Car(String color, String gearType, int door) {
        // 참조변수 this로 매개변수와 인스턴스변수 구분하기
        this.color = color;
        this.gearType = gearType;
        this.door = door;
    }
}

class Ex {
    public static void main(String[] args) {
        Car car1 = new Car();
        car1.color = "white";
        car1.gearType = "auto";
        car1.door = 4;

        Car car2 = new Car("black", "manual", 3);

        System.out.println("car1의 옵션 : " + car1.color + car1.gearType + car1.door);
        System.out.println("car2의 옵션 : " + car2.color + car2.gearType + car2.door);

    }
}
```

▶ this: **인스턴스 자신은 가르키는 참조변수, 인스턴스의 주소가 저장되어 있다.** 모든 인스턴스 메서드에 지역변수로 숨겨진 채로 존재한다. 생성자의 매개변수로 선언된 변수의 이름이 인스턴스변수의 이름과 같은 경우에는 두 변수를 서로 구분하기 위해서 인스턴스변수 앞에 'this'를 사용한다. this는 참조변수(메서드 내 지역변수)로 인스턴스 자신을 가리킨다. 'this'로 인스턴스 변수에 접근할 수 있는 것이다. this를 사용할 수 있는 것은 인스턴스멤버 뿐으로,

static메서드에서는 this역시 사용할 수 없다. 사실 생성자를 포함한 모든 인스턴스메서드에는 자신이 관련된 인스턴스를 가리키는 참조변수 'this'가 지역변수로 숨겨진 채로 존재한다.

● **생성자에서 다른 생성자 호출하기 - this(), this(매개변수): 생성자, 같은 클래스의 다른 생성자를 호출할 때 사용한다. 다른 생성자의 이름으로 클래스 이름대신 this를 사용하고, 첫 줄에 다른 생성자를 호출**한다면 한 생성자에서 다든 생성자를 호출할 수 있다.

```
import java.util.*;

class Car {
    String color;
    String gearType;
    int door;

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

    Car() {   // 아무런 옵션(매개변수)를 주지 않으면,
        // 기본적으로 흰색에 자동변속기어, 문이 4개인 자동차를 생산한다
       this("White", "auto", 4);
    };

    Car(String color) {  // 색깔 옵션만 줄 경우 색깔만 바꾸고 나머지 옵션은 동일
        this(color, "auto", 4);
    }

}

class Ex {
    public static void main(String[] args) {
        Car car1 = new Car();
        Car car2 = new Car("Red");
        Car car3 = new Car("Black", "manual", 3);

        System.out.println("car1의 옵션 : " + car1.color + car1.gearType + car1.door);
        System.out.println("car2의 옵션 : " + car2.color + car2.gearType + car2.door);
        System.out.println("car3의 옵션 : " + car3.color + car3.gearType + car3.door);

    }
}
```

```
"C:\Program Files\Java\jdk-17.0.1\b
car1의 옵션 : Whiteauto4
car2의 옵션 : Redauto4
car3의 옵션 : Blackmanual3

Process finished with exit code 0
```

● 생성자를 이용한 인스턴스의 복사: **현재 사용하고 있는 인스턴스와 같은 상태를 갖는(모든 인스턴스 변수, 즉, 상태가 동일한 값을 갖고 있는) 인스턴스를 하나 더 만들고자 할 때**, 생성자를 이용할 수 있다. 방법은 **클래스의 참조변수를 매개변수로 선언한 생성자를 만들면 된다**. 매개변수로 넘겨진 참조변수가 가르키는 인스턴스의 인스턴스변수의 값을 인스턴스 자신으로 복사하는 것이다. (참고: Java API의 많은 클래스들이 인스턴스 복사를 위한 생성자를 정의해놓았다. Object 클래스에 정의된 clone 메서드를 이용하면 간단히 인스턴스를 복사할 수 있다).

```java
import java.util.*;

class Car {
    String color;
    String gearType;
    int door;

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

    Car() {   // 아무런 옵션(매개변수)를 주지 않으면,
        // 기본적으로 흰색에 자동변속기어, 문이 4개인 자동차를 생산한다
       this("White", "auto", 4);
    };

    // 인스턴스 복사를 위한 생성자
//    Car(Car c) {
//        color = c.color;
//        gearType = c.gearType;
//        door = c.door;
//    }

    Car(Car c) {
        this(c.color, c.gearType, c.door);
    }

}

class Ex {
    public static void main(String[] args) {
        Car car1 = new Car();
        Car car2 = new Car("Black", "manual", 3);
        Car car3 = new Car(car2);  // car2를 복사


        System.out.println("car1의 옵션 : " + car1.color + car1.gearType + car1.door);
        System.out.println("car2의 옵션 : " + car2.color + car2.gearType + car2.door);
        System.out.println("car3의 옵션 : " + car3.color + car3.gearType + car3.door);

        System.out.println("변경");

        car2.color = "Orange";

        System.out.println("car2 차의 색 : " + car2.color);
        System.out.println("car3 차의 색 : " + car3.color);

        // 인스턴스 car3는 car2를 복사하여 생성된 것으로 처음에는 서로 같은 상태를 갖지만,
        // 서로 독립적으로 메모리공간에 존재하는 별도의 인스턴스이므로 car2의 값들이 변경되어도
        // car3는 영향을 받지 않는다.


    }
}
```

```
"C:\Program Files\Java\jdk-17.0.1\
car1의 옵션 : Whiteauto4
car2의 옵션 : Blackmanual3
car3의 옵션 : Blackmanual3
변경
car2 차의 색 : Orange
car3 차의 색 : Black

Process finished with exit code 0
```

정리하자면, 인스턴스를 생성할 대는 다음의 두 가지 사항을 결정해야 한다 — 1) 클래스: 어떤 클래스의 인스턴스를 생성할 것인가? 2) 생성자: 선택한 클래스의 어떤 생성자로 인스턴스를 생성할 것인가?

● 변수의 초기화: 변수를 선언하고 처음으로 값을 저장하는 것. 멤버변수는 초기화를 하지 않아도 자동적으로 변수의 자료형에 맞는 기본값으로 초기화하지 않고 사용해도 되지만, 지역변수는 사용하기 전에 반드시 초기화해야 한다. 초기화 전에 지역변수를 사용하려면 컴파일 에러가 발생한다. 다시 말해, **멤버변수(클래스 변수와 인스턴스 변수)와 배열의 초기화는 선택적이지만, 지역변수의 초기화는 필수적이다.**

선언예	설 명
`int i=10;` `int j=10;`	int형 변수 i를 선언하고 10으로 초기화 한다. int형 변수 j를 선언하고 10으로 초기화 한다.
`int i=10, j=10;`	같은 타입의 변수는 콤마(,)를 사용해서 함께 선언하거나 초기화 할 수 있다.
`int i=10, long j=0;`	**에러.** 타입이 다른 변수는 함께 선언하거나 초기화할 수 없다.
`int i=10;` `int j=i;`	변수 i에 저장된 값으로 변수 j를 초기화 한다. 변수 j는 i의 값인 10으로 초기화 된다.
`int j=i;` `int i=10;`	**에러.** 변수 i가 선언되기 전에 i를 사용할 수 없다.

● 멤버변수의 초기화 방법

1. **명시적 초기화**(explicit initialization): 변수를 선언과 동시에 초기화하는 것이다. 가장 기본적이면서도 간단한 방법으로 여러 초기화 방법 중에서 가장 우선적으로 고려되어야 한다.

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

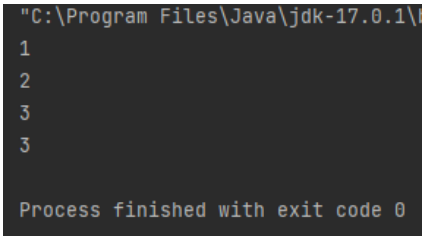
보다 복잡한 작업이 필요할 때는, 초기화 블럭 또는 생성자를 사용해야 한다.

2. 생성자(constructor)

3. **초기화 블럭**(initialization block): 클래스변수의 복잡한 초기화에 사용되는 클래스 초기화 블럭과, 인스턴스변수의 복잡한 초기화에 사용되는 인스턴스 초기화 블럭이 있다. 인스턴스 초기화 블럭은 클래스 내에 블럭{}을 만들고 그 안에 코드를 작성하면 되고, 클래스 초기화 블럭은 앞에 static을 붙이고 블럭 {}을 만들면 된다. 초기화 블럭 내에는 조건문, 반복문, 예외처리구문 등을 자유롭게 사용할 수 있으므로, 초기화 작업이 복잡하여 명시적 초기화만으로는 부족할 경우 초기화 블록을 사용한다. **클래스 초기화 블럭은 클래스가 메모리에 처음 로딩될 때 한 번만 수행되며, 인스턴스 초기화 블럭은 인스턴스를 생성할 때마다 수행된다.** 그리고 **생성자보다 인스턴스 초기화 블럭이 먼저 수행된다** (클래스가 처음 로딩될 때 클래스변수들이 자동적으로 메모리에 만들어지고, 곧바로 초기화 블럭이 수행되어

클래스변수들을 초기화하고, 그 후에 main메서드가 수행되면서 인스턴스가 생성되면서 인스턴스 초기화 블럭이 수행되어 인스턴스변수들을 초기화하고, 끝으로 생성자가 수행된다). **인스턴스 변수의 초기화는 주로 생성자를 사용하고, 인스턴스 초기화 블럭은 모든 생성자에서 공통으로 수행해야 하는 코드를 넣는데 사용한다**.

```java
import java.util.*;

// 모든 생성자에 같은 코드가 중복될 경우, 코드의 재사용성을 높이고 중복을 제거하기 위해
// 중복된 코드를 인스턴스 블럭에 넣어준다.
/* 예시)
Car() {
    count++;
    serialNo = count;
    color = "White";
    gearType = "Auto";
}
Car(String color, String gearType) {
    count++;
    serialNo = count;
    this.color = color;
    this.gearType = gearType;
}
 */

class Car {
    static int count;   //  총 생성된 인스턴스 수를 저장하기 위한 변수
    int serialNo;  // 인스턴스 고유의 번호
    String color;
    String gearType;

    { // 인스턴스 초기화 블럭
        // 모든 생성자에서 공통으로 수행되어야 하는 내용이다.
        // 인스턴스가 만들어질 때마다 사용된다.
        // 즉, 어떤 생성자를 사용해서 인스턴스를 만들더라도 해당 블럭이 먼저 수행된다.
        count++;   // Car인스턴스가 만들어질 때마다 클래스변수 count의 값을 1 증가시킨 다음,
        serialNo = count;  // 그 값을 인스턴스변수 serialNo에 저장한다.
        // 만약 count을 인스턴스 변수로 선언했다면, 인스턴스가 생성될 때마다 0으로 초기화돼서
        // 모든 Car인스턴스의 serialNo값은 항상 1이 될 것이다.
    }

    Car() {
        color = "White";
        gearType = "Auto";
    }

    Car(String color, String gearType) {
        this.color = color;
        this.gearType = gearType;
    }
}
class Ex {
    public static void main(String[] args) {
        Car car1 = new Car();
        System.out.println(car1.serialNo);
        Car car2 = new Car("Black","Auto");
        System.out.println(car2.serialNo);
        Car car3 = new Car();
        System.out.println(car3.serialNo);
        System.out.println(Car.count);
    }
}
```




```java
class Ex {
    static int[] arr = new int[10];    // 명시적 초기화를 통해 배열 arr을 생성

    static {    // 클래스 초기화 블럭
        for (int i = 0; i < arr.length; i++) {
            // 1과 10 사이의 임의의 값을 배열 arr에 저장
            arr[i] = (int) (Math.random() * 10) + 1;
            // 이처럼 배열이나 예외처리가 필요한 초기화에서는
            // 명시적 초기화로는 복잡한 초기화 작업을 할 수 없기 때문에
            // 클래스 초기화 블럭을 사용한다.
        }
    }
    public static void main(String[] args) {
       for (int i = 0; i < arr.length; i++) {
           System.out.println("arr[" + i + "] :" + arr[i]);
       }
    }
}
```

● 멤버변수의 초기화 시기와 순서

멤버변수	초기화시점	초기화순서
클래스변수	클래스가 처음 로딩될 때 단 한번 초기화된다. (*클래스에 대한 정보가 요구될 때, 예를 들면 클래스 멤버를 사용했을 때, 인스턴스를 생성할 때 등, 클래스는 메모리에 로딩된다—JVM의 종류에 따라 클래스가 필요할 때 바로 메모리에 로딩되도록 설계되어 있는 것도 있고, 실행효율을 높이기 위해 사용될 클래스들을 프로그램이 시작될 때 미리 로딩하도록 되어 있는 것도 있다)	기본값 → 명시적 초기화 → 클래스 초기화 블럭
인스턴스변수	인스턴스가 생성될 때마다 각 인스턴스별로 초기화가 이루어진다	기본값 → 명시적 초기화 → 인스턴스 초기화 블럭 → 생성자

```java
// Example

class InitTest {
    // 클래스변수와 인스턴스변수의 명시적 초기화
    static int cv = 1;
    int iv = 1;

    // 클래스 초기화 블럭
    static {
        cv = 2;
    }

    // 인스턴스 초기화 블럭
    {
        iv = 2;
    }

    // 생성자
    InitTest() {
        iv = 3;
    }
}
```

위의 InitTest클래스는 클래스변수(cv)와 인스턴스변수(iv)를 각각 하나씩 가지고 있다. 'new InitTest();'와 같이 하여 인스턴스를 생성했을 때, cv와 iv가 초기화되어가는 과정을 단계별로 자세히 살펴보도록 하자.

클래스 초기화			인스턴스 초기화			
기본값	명시적 초기화	클래스 초기화블럭	기본값	명시적 초기화	인스턴스 초기화블럭	생성자
cv 0	cv 1	cv 2	cv 2 iv 0	cv 2 iv 1	cv 2 iv 2	cv 2 iv 3
1	2	3	4	5	6	7

▶ **클래스변수 초기화 (1~3)** : 클래스가 처음 메모리에 로딩될 때 차례대로 수행됨.

▶ **인스턴스변수 초기화(4~7)** : 인스턴스를 생성할 때 차례대로 수행됨

| 중요 | 클래스변수는 항상 인스턴스변수보다 항상 먼저 생성되고 초기화 된다.

1. cv가 메모리(method area)에 생성되고, cv에는 int형의 기본값인 0이 cv에 저장된다.
2. 그 다음에는 명시적 초기화(int cv=1)에 의해서 cv에 1이 저장된다.
3. 마지막으로 클래스 초기화 블럭(cv=2)이 수행되어 cv에는 2가 저장된다.
4. InitTest클래스의 인스턴스가 생성되면서 iv가 메모리(heap)에 존재하게 된다. iv 역시 int형 변수이므로 기본값 0이 저장된다.
5. 명시적 초기화에 의해서 iv에 1이 저장되고
6. 인스턴스 초기화 블럭이 수행되어 iv에 2가 저장된다.
7. 마지막으로 생성자가 수행되어 iv에는 3이 저장된다.

```java
class Document {
    static int count = 0;   // 이름없는 문서 숫자. 클래스변수로 선언되었다.
    String document_name;

    Document(String document_name) {
        this.document_name = document_name;
        System.out.println("문서 " + this.document_name + "가 생성되었습니다.");
    }

    Document() { // 문서 생성 시 문서명을 지정하지 않았을 때
        this("제목없음" + ++count);
        // count가 1올라가고 그것이 문자열로 변환되어 제목없음과 합쳐진 후에
        // 그것이 인자로 지정되어 위의 생성자의 매개변수로 복사된다.
    }

}
class Ex {
    public static void main(String[] args) {
        Document d1 = new Document();
        Document d2 = new Document("에스프레소의 공부 일기.txt");
        Document d3 = new Document();
        Document d4 = new Document();
    }
}
```

```
"C:\Program Files\Java\jdk-17.0.1\bin\java
문서 제목없음1가 생성되었습니다.
문서 에스프레소의 공부 일기.txt가 생성되었습니다.
문서 제목없음2가 생성되었습니다.
문서 제목없음3가 생성되었습니다.

Process finished with exit code 0
```

Resource: 자바의 정석