

# Java Overloading constructor

## 오버로딩(Overloading)

- 과적하다 -> 많이시다
- 하나의메서드이름으로여거기능을구현하기때문에붙여진이름
  - \*\* 보통하나의메서드이름에하나의기능만을구현해야한다.
- 자바에서는한클래스내에이미사용하려는이름과같은이름을가진메서드가있더라도,
- 매개변수의개수또는타입이다른면,
- 같은이름을사용해서메서드를정의할수있다.
  - >메서드오버로딩or오버로딩

# Overloading

# 오버로딩(Overloading)

## 오버로딩의조건

1. 메서드의이름이같아야한다.
2. 매개변수의개수또는타입이달라야한다.

## 오버로딩의장점

1. 오버로딩을통해여러메서드들이하나의이름으로정의
2. 오류의가능성을많이줄일수있다.
3. 메서드의이름을보고기능을유추할수있다.
4. 메서드의이름절약

```
public void print(boolean b) { this.write(String.valueOf(b)); }

public void print(char c) { this.write(String.valueOf(c)); }

public void print(int i) { this.write(String.valueOf(i)); }

public void print(long l) { this.write(String.valueOf(l)); }

public void print(float f) { this.write(String.valueOf(f)); }

public void print(double d) { this.write(String.valueOf(d)); }

public void print( @NotNull char[] s) { this.write(s); }

public void print( @Nullable String s) { this.write(String.valueOf(s)); }

public void print( @Nullable Object obj) { this.write(String.valueOf(obj)); }

public void println() { this.newLine(); }

public void println(boolean x) {
    synchronized(this) {
        this.print(x);
        this.newLine();
    }
}
```

# 생성자(Constructor)

## 생성자의 정의

1. 인스턴스가 생성될 때 호출되는 '인스턴스 초기화 메서드'
  2. 인스턴스 변수의 초기화 작업에 주로 사용
  3. 인스턴스 생성 시에 실행되어야 할 작업을 위해서도 사용
- \*인스턴스 초기화 -> 인스턴스 변수들을 초기화하는 것.

## 생성자

1. 생성자의 이름은 클래스의 이름과 같아야 한다.
2. 생성자는 리턴 값 X
3. 생성자는 메서드처럼 클래스 내에 선언
4. 구조도 메서드와 유사하지만 리턴 값 X
5. 생성자도 메서드이기 때문에 void를 붙여야 하지만, 모든 생성자가 리턴 값이 없으므로 void를 생략할 수 있게 한 것.

```
public class Item {  
  
    private Long id;  
    private String itemName;  
    private Integer price;  
    private Integer quantity;  
  
    1 related problem  
    public void Item() {  
    }  
  
    public Item(String itemName, Integer price, Integer quantity) {  
        this.itemName = itemName;  
        this.price = price;  
        this.quantity = quantity;  
    }  
}
```

Item.java C:\study2\item-service\item-service\src\main\java\hello\

- ⚠ Unused import statement :4
- ⚠ Unused import statement :5
- ⚠ Found problems related to 'hello.itemservice.domain.item.Item' :9
- ⚠ Found problems related to 'Item()' :16
- ⚠ Method 'Item()' is never used :16
- ⚠ Method name 'Item' is the same as its class name :16

## 생성자(Constructor)

- 생성자는 오버로딩이 가능
- 따라서 하나의 클래스에 여러 개의 생성자가 존재할 수 있다.
- 연산자 new가 인스턴스를 생성하는 것이지
- 생성자가 인스턴스를 생성하는 것은 X

```
public class Item {  
  
    private Long id;  
    private String itemName;  
    private Integer price;  
    private Integer quantity;  
  
    public Item() {    기본생성자  
    }  
  
    public Item(String itemName, Integer price, Integer quantity) {  
        this.itemName = itemName;    파라미터(매개변수)가 있는 생성자  
        this.price = price;           - 인스턴스 생성 시 수행될 코드,  
        this.quantity = quantity;     - 주로 인스턴스 변수의 초기화 코드를 적는다  
    }  
}
```

# 생성자(Constructor)

## 기본생성자

1. 모든 클래스에는 반드시 하나 이상의 생성자가 정의
2. 기본생성자가 컴파일러에 의해서 추가되는 경우  
→ 클래스에 정의된 생성자가 하나도 없을 때

## 매개변수가 있는 생성자

1. 생성자도 메서드처럼 매개변수를 선언하여 호출시 값을 넘겨받아서 인스턴스의 초기화 작업에 사용가능
2. 각기 다른 값으로 초기화되어야 하는 경우 → 매개변수를 사용한 초기화는 매우 유용  
→ 인스턴스 생성과 동시에 원하는 값으로 초기화

```
public class Item {  
  
    private Long id;  
    private String itemName;  
    private Integer price;  
    private Integer quantity;  
  
    public Item() { 기본생성자  
    }  
  
    public Item(String itemName, Integer price, Integer quantity) {  
        this.itemName = itemName; 파라미터(매개변수)가 있는 생성자  
        this.price = price; -인스턴스 생성시 수행될 코드,  
        this.quantity = quantity; -주로 인스턴스 변수의 초기화 코드를 적는다  
    }  
}
```

# 생성자(Constructor)

## 기본생성자

1. 모든 클래스에는 반드시 하나 이상의 생성자가 정의
2. 기본생성자가 컴파일러에 의해서 추가되는 경우  
→ 클래스에 정의된 생성자가 하나도 없을 때

## 매개변수가 있는 생성자

1. 생성자도 메서드처럼 매개변수를 선언하여 호출시 값을 넘겨받아서 인스턴스의 초기화 작업에 사용가능
2. 각기 다른 값으로 초기화되어야 하는 경우 → 매개변수를 사용한 초기화는 매우 유용  
→ 인스턴스 생성과 동시에 원하는 값으로 초기화

```
public class Item {  
  
    private Long id;  
    private String itemName;  
    private Integer price;  
    private Integer quantity;  
  
    public Item() { 기본생성자  
    }  
  
    public Item(String itemName, Integer price, Integer quantity) {  
        this.itemName = itemName; 파라미터(매개변수)가 있는 생성자  
        this.price = price; -인스턴스 생성시 수행될 코드,  
        this.quantity = quantity; -주로 인스턴스 변수의 초기화 코드를 적는다  
    }  
}
```

## 생성자(Constructor) 사용 예제

```
import lombok.Getter;
import lombok.Setter;

@Getter @Setter
public class Member {

    private Long id; //id는 회원을 데이터베이스에 저장하면,
    private String username;
    private int age;

    public Member() {
    } //기본 생성자

    public Member(String username, int age) {
        this.username = username;
        this.age = age;
    }
}
```

```
@PostMapping(value = "/save")
@RequestMapping(value = "/save", method = RequestMethod.P
public String save(
    @RequestParam("username") String username,
    @RequestParam("age") int age,
    Model model) {

    Member member = new Member(username, age);
    memberRepository.save(member);

    model.addAttribute(attributeName: "member", member);
    return "save-result";
}
```



## 생성자(Constructor)

생성자에서 다른 생성자 호출 -> this(), this

1. 생성자 간에도 서로 호출이 가능

생성자 호출 조건

1. 생성자의 이름으로 클래스 이름 대신 this 를 사용
2. 한 생성자에서 다른 생성자를 호출할 때는  
반드시 첫 줄에서만 호출 가능

```
public Member(String username) {  
    age = 2;  
    Member(1, username, 2)  
}
```

```
import lombok.Getter;  
import lombok.Setter;  
  
@Getter @Setter  
public class Member {  
  
    private Long id; //id는 회원을 데이터베이스에 저장하면, 그게  
    private String username;  
    private int age;  
  
    public Member() {  
    } //기본 생성자  
  
    public Member(String username, int age) {  
        this.username = username;  
        this.age = age;  
    }  
}
```

## 생성자(Constructor)

생성자에서 다른 생성자 호출 -> this(), this

1. 생성자 간에도 서로 호출이 가능

생성자 호출 조건

1. 생성자의 이름으로 클래스 이름 대신 this 를 사용
2. 한 생성자에서 다른 생성자를 호출할 때는  
반드시 첫 줄에서만 호출 가능

```
public Member(String username) {  
    age = 2;  
    Member(1, username, 2)  
}
```

예러 1. 생성자의 두 번째 줄에서 다른 생성자 호출

예러 2. this(1, username, 2); 로 해야 함

\* 초기화 도중에 다른 생성자를 호출하면, 이미 호출된 다른 생성자 내 멤버변수들의 값을 초기화하기 때문에 금지된 것.

```
import lombok.Getter;  
import lombok.Setter;  
  
@Getter @Setter  
public class Member {  
  
    private Long id; //id는 회원을 데이터베이스에 저장하면, 그게  
    private String username;  
    private int age;  
  
    public Member() {  
    } //기본 생성자  
  
    public Member(String username, int age) {  
        this.username = username;  
        this.age = age;  
    }  
}
```

## this와 this()의 차이점

this는 인스턴스 자신을 가리키는 참조 변수  
this()는 생성자

this

1. 인스턴스 변수와 지역변수를 구분하기 위해서 사용
2. Car 생성자 안에서 this.color는 인스턴스 변수이고, color는 매개변수로 정의된 지역변수

\* static 메서드에서는 this를 사용하지 못한다

```
package hello.servlet.domain.member;

class Car {
    String color; // 인스턴스 변수
    String gearType;
    int door;

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

## this와 this()의 차이점

this는 인스턴스 자신을 가리키는 참조 변수  
this()는 생성자

this()

1. this()는 같은 클래스의 다른 생성자를 호출할 때 사용.
2. 코드의 Car() 생성자와 Car(String color) 생성자는 this()를 통해 모두 Car(String color, String gearType, int door) 생성자를 호출하고 있는 것.

```
package hello.servlet.domain.member;

class Car{
    String color; // 인스턴스 변수
    String gearType;
    int door;

    Car(){
        this( color: "white", gearType: "auto", door: 4);
        // Car(String color, string gearType, int door)를 호출
    }

    Car(String color){
        this(color, gearType: "auto", door: 4);
    }

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

# 변수의 초기화

## 정의

1. 변수를 선언하고 처음으로 값을 저장하는 것을 의미
2. 경우에 따라서 필수적 or 선택적
3. 가능하면 선언과 동시에 적절한 값으로 초기화 할 것
4. 멤버변수는 초기화를 하지 않아도 자동적으로 변수의 자료형에 맞게 초기화
5. 지역변수는 사용하기 전에 반드시 초기화

### \* 각 타입의 기본값

boolean -> false  
byte, short, int -> 0  
long -> 0L  
double -> 0.0d or 0.0

char -> '\u0000'  
float -> 0.0f  
참조형 변수 -> null

```
package hello.itemservice.domain.item;

public class InitTest {
    int x;
    int y = x; //문제 없음

    void method() {
        int i;
        int j = i; // 컴파일 에러: 지역변수를 초기화하지 않고 사용함
    }
}
```

# 변수의 초기화

## 멤버변수의 초기화 방법

1. 명시적 초기화
2. 생성자
3. 초기화 블록
  - 인스턴스 초기화 블록: 인스턴스변수를 초기화 하는데 사용
  - 클래스 초기화 블록: 클래스변수를 초기화 하는데 사용

---

## 명시적 초기화(explicit initialization)

- 변수를 선언과 동시에 초기화 하는 것
- 가장 기본적이면서 간단한 초기화 방법  
-> 가장 우선적으로 고려
- 보다 복잡한 초기화 작업이 필요할 때는  
-> 초기화 블록 또는 생성자를 사용

```
package hello.itemservice.domain.item;

class Car {
    int door = 4;
    Engine e = new Engine();

    //...
}
```

# 변수의 초기화

## 초기화 블록

1. 클래스 초기화 블록
  - > 클래스변수의 복잡한 초기화에 사용
  - > 인스턴스 초기화 블록 앞에 단순히 static을 덧붙이기만 하면 된다.
2. 인스턴스 초기화 블록
  - > 인스턴스변수의 복잡한 초기화에 사용
  - > 클래스내에 블록{ } 만들고 그 안에 코드를 작성하면 된다.

\*초기화 블록 내에는 메서드 내에서와 같이  
조건문, 반복문, 예외처리 구문 등을 자유롭게 사용 가능

- 클래스 초기화블록은 클래스가 메모리에 처음 로딩될 때 한번만 수행
- 인스턴스 초기화블록은 생성자와 같이 인스턴스를 생성할 때마다 수행  
\* 생성자 보다 인스턴스 초기화 블록이 먼저 수행

```
package hello.itemservice.domain.item;

class StaticBlockTest { 명시적 초기화로 배열 arr 생성
    static int[] arr = new int[10];

    static { 클래스 초기화 블록을 이용해서 random()으로 값을 채운다
        for(int i=0;i<arr.length;i++) {
            arr[i] = (int)(Math.random() * 10) + 1;
        }
    }

    public static void main(String args[]) {
        for(int i=0;i<arr.length;i++) {
            System.out.println("arr["+i+"] : " + arr[i]);
        }
    }
}
```

arr[0] : 3  
arr[1] : 10  
arr[2] : 2  
arr[3] : 2  
arr[4] : 5  
arr[5] : 7  
arr[6] : 10  
arr[7] : 1  
arr[8] : 6  
arr[9] : 4

Process finished with exit code 0

## 변수의 초기화

클래스 변수의 초기화 시점 : 클래스가 처음 로딩 될 때 단 한번 초기화됨

인스턴스 변수의 초기화 시점 : 인스턴스가 생성될 때마다 각 인스턴스 별로 초기화가 이루어 진다.

클래스 변수의 초기화 순서 : 기본값 -> 명시적 초기화 -> 클래스 초기화 블록

인스턴스 변수의 초기화 순서 : 기본값 -> 명시적 초기화 -> 인스턴스 초기화 블록 -> 생성자

```
public class TestStudy
{
    4 usages
    static int var = 10;
    1 usage
    int hello = 3;
    {
        System.out.println("hello -> 5 초기화");
        hello = 5;
    }
    static {
        System.out.println("var -> 20 초기화");
        var = 20;
    }
}
```

일반적인 초기화 블록

정적 초기화 블록

/Library/Java/JavaVirtualMachines/adoptopenjdk-11.

정적 변수 var -> 20 초기화

인스턴스 생성

멤버 변수 hello -> 5 초기