JAVA 8강 생성자

강사 박주병

1. 생성자

가. 생성자의 특징

1) 객체 생성시 최초 1번만 실행되는 특별한 메서드
2) 클래스와 이름이 같아야 한다(대소문자 구분)
3) 리턴 타입이 없어야 한다.(void가 아님)
4) 객체는 무조건 생성자를 통해 생성된다.

```
public class Student
{
    public String name ;
    int kor ;
    int eng ;
    int math ;
    Student()
    {

    }

    Student(String name)
    {
        this.name = name;

    }

    Student(String name, int kor, int eng,int math)
    {
        this.name =name;
        this.kor =kor;
        this.eng = eng;
        this.math=math;

    }
```

그림 1

그림1의 Student 클래스 내부를 보면 Student() 메서드 3개가 정의되어 있다. 그런데 메서드의 구조가 이상한 것을 볼수 있다. 바로 반환타입이 적혀있지 않다. 보통 반환타입이 없다면 void를 작성해야 한다. 하지만 void가 아닌 아예 적혀져 있지 않다. 그리고 메서드의 이름또한 클래스의 이름과 완전히 동일한 것을 볼수 있다.

생성자를 만들기 위해서는 이와 같이 반환타입이 없어야 하고 클래스의 이름과 완전히 동일해야 한다.

그림1의 빨간네모 부분의 3개의 메서드가 모두 생성자인 것이다.

```
Student student1 = new Student();
student3.name = "박주병";
student3.kor = 30;
student3.eng = 50;
student3.math = 20;
```

그림 2 생성자의 사용예

생성자를 사용 하는 방법은 그림2와 같이 사용한다. 사실 지금까지 객체를 생성하는 방법과 크게 다르지 않다. 즉 우리는 지금까지 알게 모르게 생성자를 사용하고 있었던 것이다. 그림2에서 new Student(); 를 호출함으로써 매개변수가 하나도 없는 생성자를 호출하고 있는 것이다. 그리고 Student클래스의 내부를 보면 해당 생성자는 내부에서 아무것도 하지 않고 있다.

이렇게 매개변수가 하나도 없는 생성자를 기본 생성자(디폴트 생성자) 라고 부른다. 물론 매개변수는 없지만 내부를 채워 넣을 순 있다.

```
Student student1 = new Student();
Student student2 = new Student("홍길동");
Student student3 = new Student("박주병",30,50,20);
```

그림 3

그림3은 매개변수가 있는 생성자를 사용 하는 방법이다.

ParkJuByeong

매개변수가 있는 생성자의 내부를 보면 매개변수로 받은 데이터를 Student의 멤버변수로 저장하고 있는 것을 볼수 있다.
지금까지는 Student의 객체를 생성 후 멤버변수를 초기화하기 위해서는 그림2와 같이 참조변수를 통해 멤버변수에 접근하여 하나하나 대입을 해줬어야 했다. 하지만 그림3을 보면 생성자의 매개변수로 값들을 넘겨주면 생성자 내부에서 멤버변수로 값을 저장한다.

생성자를 통해 멤버변수를 초기화 하는 방법은 기존에 참조변수.멤버변수이름 으로 접근하여 하나씩 초기화 하는 방법보다 코드가 간결하고 가독성이 올라간다. 이렇게 생성자는 일반적으로 멤버변수를 초기화 하거나 객체가 생성될 때 초기에 수행해야 할 기능들을 수행하는 목적으로 사용 한다.

나. 기본 생성자(디폴트 생성자)

앞서 객체는 반드시 생성자를 통해 생성된다고 하였고 지금까지 알게 모르게 new Student()와 같이 기본 생성자를 이용해 객체를 생성해왔다. 그런데 한가지 의문점이 들것이다. 우리는 지금까지 클래스를 만들 때 기본 생성자 라는걸 만든적이 없을 것이다.

1) 기본생성자의 특징
   가) 명시적으로 만든 생성자가 하나도 없다면 자동으로 기본 생성자를 만들어준다.
   나) 명시적으로 한 개이상의 생성자가 만들어져 있다면 기본 생성자는 만들어주지 않는다.

```
public class Student
{
    String name;
    int kor;
    int eng;
    int math;


    Student(String name1)
    {
        name = name1;
    }

    Student(String name1, int kor1, int eng1,int math1)
    {
        name =name1;
        kor =kor1;
        eng = eng1;
        math=math1;

    }
```

그림 4

그림4를 보면 Student 클래스에는 명시적으로 매개변수가 있는 생성자 2개가 선언되어 있다.

이럴 경우 기본 생성자가 없는 상태이며 기본 생성자를 자동으로 만들어주지 않는다.

하지만 지금까지처럼 생성자를 아무것도 만들지 않았다면 기본 생성자를 자동으로 만들어주기에 new Student()를 이용해 객체를 생성 할 수 있다.

```
Student student1 = new Student();
Student student2 = new Student("홍길동");
Student student3 = new Student("박주병",30,50,20);
```

→ 기본 생성자가 없어 ERROR!

그림 5

그림5를 보면 기본 생성자를 만들어주지 않기 때문에 객체 생성할 때 사용할 수가 없다. 즉 무조건 매개변수를 넘겨주어야 객체를 생성 할 수 있는 것이다. 그렇다고 이런 상황이 안좋다는건 아니다. 객체를 생성할 때 반드시 값을 초기화 해주어야 하는 멤버변수가 있을시 일부러 기본 생성자를 제공하지 않아 무조건 값을 넣어야 객체를 생성할수 있게 유

ParkJuByeong

도 할 수 있다. 예를 들어 People 클래스의 경우 주민번호 같은경우이다.

2) 생성자 오버로딩

생성자 역시 메서드처럼 오버로딩이 가능하다. 오버로딩이랑 메서드의 이름이 동일하고 매개변수의 개수나 데이터타입을 달리하여 같은 이름으로 여러 가지 방법의 메서드 사용방법을 만드는 것이다.

생성자 역시 오버로딩이 가능하다. 일단 모든 생성자의 이름은 기본적으로 클래스의 이름과 일치 해야된다. 그리고 매개변수의 개수나 데이터타입을 다르게 주어 오버로딩이 가능하다.

```java
public class Student
{
    String name;
    int kor;
    int eng;
    int math;

    Student(String name1)
    {
        name = name1;
    }

    Student(String name1, int kor1, int eng1,int math1)
    {
        name =name1;
        kor =kor1;
        eng = eng1;
        math=math1;

    }
```

그림 6 생성자 오버로딩 예시

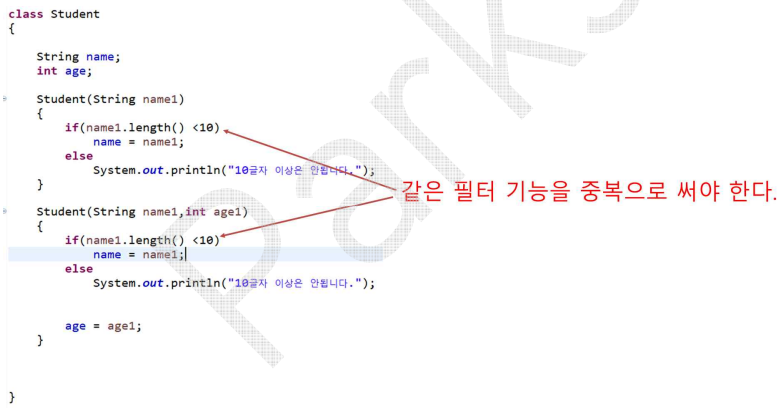

그림 7

그림6의 생성자 오버로딩을 보면 2개의 생성자 모두 내부에서 name 멤버변수를 초기화 하는 기능이 들어가 있다.
이 부분이 지금 예시에서는 간단한 코드 이지만

그림7과 같이 코드가 길어지게 되면 중복된 코드가 많아져 유지보수에 좋지 못한 코드가 된다.

이럴 때 만약 Student(String name,int age)에서 Student(String name) 생성자를 호출해서 쓰는 구조 이면 name을 멤버변수에 초기화 하는 코드는 Student(String name)에만 있으면 된다.

ParkJuByeong

3) 생성자에서 다른 생성자 호출

```
class Student
{

    String name;
    int age;

    Student(String name1)
    {
        if(name1.length() <10)
            name = name1;
        else
            System.out.println("10글자 이상은 안됩니다.");
    }

    Student(String name1,int age1)
    {
        Student(name1);


        age = age1;
    }


}
```

매개변수1개 생성자를 재활용하면
필터 기능을 다시 안만들어도 된다.

근데 왜 안될까...?

그림 8

코드 중복 문제를 해결하기 위해 생성자에서 다른 생성자를 이용해 멤버변수를 초기화 하려고 그림8과 같이 작성해보면 컴파일 오류가 발생하는 것을 볼 수 있다.
생성자에서 다른 생성자를 사용할때는 다음의 규칙이 있다.

가) 생성자의 이름을 this로 호출해야 한다.
나) 가장 첫줄에 작성해야한다.
다) 일반메서드에서는 호출할수 없다.

```
class Student
{

    String name;
    int age;

    Student(String name1)
    {
        if(name1.length() <10)
            name = name1;
        else
            System.out.println("10글자 이상은 안됩니다.");
    }

    Student(String name1,int age1)
    {
        this(name1);

        age = age1;
    }
}
```

Student(name1)이 아닌 this로 호출을 해야 한다.

그림 9

그림9와 같이 생성자에서 다른 생성자를 사용 할때는 this라는 이름으로 호출을 해야 한다.그렇다면 일반적인 메서드 호출하듯이 Student(name) 처럼은 왜 안되는것일까?

사실 생성자를 두고 계속 메서드라고 표현을 하고 있지만 이는 이해를 돕기 위한 설명일뿐 사실은 메서드가 아니고 객체가 생성될 때 실행되는 코드블럭이 정확한 표현이다.

그 외에 생성자이름을 통해 호출시 생성자내에서 또다른 객체를 생성하는것과 혼돈 될 수 있고 여러 가지 문제들로 인해 this라는 이름으로 호출하도록 되어 있다.

```
Student()
{
    System.out.println("생성자 호출");
    this("홍길동");
}
```

그림 10

그림10의 예시를 보면 기본 생성자 내에서 매개변수 1개를 가지는 생성자를 호출하고 있는데 첫줄이 아니면 에러가 발생하는 것을 볼 수 있다.
생성자의 기본적인 목적은 객체를 생성할 때 초기화 작업을 하는 것이다. 멤버변수가 될 수도 있고 각종 소스코드들도 포함될 것이다.그런데 이러한 초기화 코드들은 서로 실행되어야 하는 순서에 있어서 의존적일수 있다. 특정 변수가 초기하 되어야지만 실행 가능한 코드들이 있다던가 이런 상황에서 생성자들끼리 호출 순서가 아무렇게나 실행이 가능하다면 코드가 복잡해지고 오류가 발생할 가능성이크다. 물론 그림10과 같은 예시에서는 name의 초기화가 뒤에된다고 해서 문제가 발생하지 않기에 이해되지 않을수 있다. 하지만 자바에서는 초기화에 순서를 강제하면서 코드를 단순하게 만들게 유도하려는 의도로 해당 규칙이 존재하는 것이다.

ParkJuHyeong

2. this

앞서 봤던 생성자에서 다른 생성자를 호출할 때 this()처럼 호출한다고 하였다. 그러나 지금 말하고자 하는 this는 다른 의미이다. 앞서 배운 this()는 생성자를 의미하는것이고 이번에 다룰 내용은 this라는 멤버변수에 대해 말하고자 한다.

가. this변수의 특징

1) 클래스 생성시 자동으로 만들어지는 멤버변수
2) 객체 자신의 주소를 가지고 있다.

```
public class Bird {

    void fly()
    {
        System.out.println(this);
    }

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

만들지 않아도 이미 존재하는 멤버변수이다.

멤버변수이기에 클래스 내부 어디서든 사용가능하다.

그림 11

그림11의 Bird 클래스를 보면 메서드 내부에서 this라는 변수를 사용하는걸 볼 수 있다.
클래스 내부에서 자동으로만들어주는 변수이기 때문에 따로 생성하지 않아도 this라는 변수를 사용 할수 있고 멤버변수이기 때문에 fly()와 eat() 메서드에서 전부 사용가능하다.

```
public class Main {

    public static void main(String[] args) {

        Bird bird1 = new Bird();
        System.out.println("bird1 참조변수: "+bird1);

        bird1.eat();
    }
}

public class Bird {

    void fly()
    {
        System.out.println(this);
    }

    void eat()
    {
        System.out.println("this: "+this);
    }
}
```

```
Problems  Javadoc  Search  Console  Git Staging
<terminated> Main (9) [Java Application] C:₩Users₩zest1₩.p2₩pool₩plu
bird1 참조변수:joo.강의8.Bird@6f2b958e
this:joo.강의8.Bird@6f2b958e
```

bird1과 this는 같은 주소를 가지고 있다.

그림 12

나. this변수의 값

this변수는 그렇다면 어떤 값을 가지고 있는가? 그림12에서 main메서드에서 Bird 객체를 만들고 만들어진 객체의 주소값을 출력하고 있다. 이때 주소값은 6f2b958e 인 것을 알 수 있다. 그리고 eat()메서드를 호출하고 있는데 eat() 메서드 내부에서는 this의 값을 출력하고 있다. 그리고 그 값을 확인해보면 6f2b958e로 나오며 클래스 외부에서 bird1변수를 통해 객체의 주소값을 출력한 것과 동일 한 것을 볼 수 있다.

ParkJuByeong

즉 this라는 것은 객체의 주소를 가지고 있는 멤버변수 인 것이다. 그러면 이러한 변수는 왜 필요 한것일까?

클래스의 밖에서는 객체의 주소값을 가지는 참조변수를 통해 객체의 변수와 메서드를 사용할수 있다. 하지만 클래스 내부에서는 그렇게 하지 않아도 멤버변수와 메서드를 사용 할 수 있다. 그러면 this는 필요가 없을텐데 어디에 쓰이는 걸까?

```
public class Bird {

    int age;


    Bird()
    {
        age = 1;
    }

    Bird(int age)
    {
        age =age;
    }
```

그림13에서 매개변수로 나이를 받는 생성자를 보자. 매개변수로 받은 age 변수를 멤버변수 age에 저장을 하려 한다.그런데 과연 그림13의 코드가 그렇게 동작할까?

age라는 변수는 멤버변수에도 있지만 생성자의 매개변수인 지역변수로써도 존재한다. 이렇게 멤버변수와 지역변수가 이름이 같을 경우 지역변수가 우선시되어 age=age;는 결국 자기 자신의 값을 다시 자기 자신에게 넣는 의미 없는 코드가 된다.

그렇다면 이 상황에서 멤버변수 age를 가리킬수 있는 방법이 있을까?

그게 바로 this를 활용하는 방법이다.

그림 13

```
public class Bird {

    int age;


    Bird()
    {
        age = 1;
    }

    Bird(int age)
    {
        this.age =age;
    }
```

그림14에서는 this.age를 사용함으로써 age가 정확히 멤버변수라는 것을 말해주고 있다. 클래스 밖에서는 클래스 내부의 지역변수를 접근할수 없다. 그러므로 참조변수를 통해 접근할수 있는 변수는 오직 멤버변수뿐이다.

그런데 클래스 내부에서 this를 통해 마치 클래스 외부에서 쓰듯이 변수를 쓰면 그것은 무조건 멤버변수인 것이다. 따라서 내부에서 똑같은이름의 멤버변수와 지역변수를 구분할 때 this를 활용 할수 있다.

그 외에도 this는 객체의 주소를 가지고 있기 때문에 객체들을 구분할 때 쓰이거나 객체자신을 외부로 반환할때라던가 쓰임이 다양하다.

그림 14

다. static 메서드 내부에서 this 사용

```
static void test()
{
    this.eng = 30;
    this.getTotal();
}
```

그림 15

그림15의 소스코드는 과연 정상적으로 작동할까? static 메서드는 객체 생성없이 사용가능한 메서드이다. 그 이유에 대해서는 static을 다룰 때 자세히 봤었다. 결론은 test() 메서드가 실행되는 타이밍에는 아직 객체가 없을수 있다. 그러므로 static 메서드 내부에서는 this를 사용 할 수 없다.

ParkJiSeong

3. 멤버변수 초기화

클래스 내부의 멤버변수를 초기화 하는 방법에는 3가지 방법이 있다.

- 명시적 초기화
- 생성자를 이용한 초기화
- 초기화 블록

가. 명시적 초기화

```
public class Student
{
    String name ="박주병";
    int kor = 30;
    int eng = 50;
    int math =20;


    Student()
    {
        System.out.println("this: "+ this);
    }

```

그림 16

그림16과 같이 멤버변수를 선언할 때 바로 초기화를 해주는 것이다.

이렇게 초기화를 하면 객체를 생성할 때 멤버변수가 생성되며 그때 값역시 초기화가 된다.

즉 new Student()를 하는 순간 따로 멤버변수를 초기화 하지 않아도 값들이 들어가 있는 상태가 된다.

생성자를 이용한 초기화 방법에는 앞의 생성자 부분에서 다뤘으므로 따로 설명하진 않겠다.

나. 초기화 블럭

초기화 블록은 생성자가 실행되기전 선행으로 실행되는 코드블록을 의미한다.

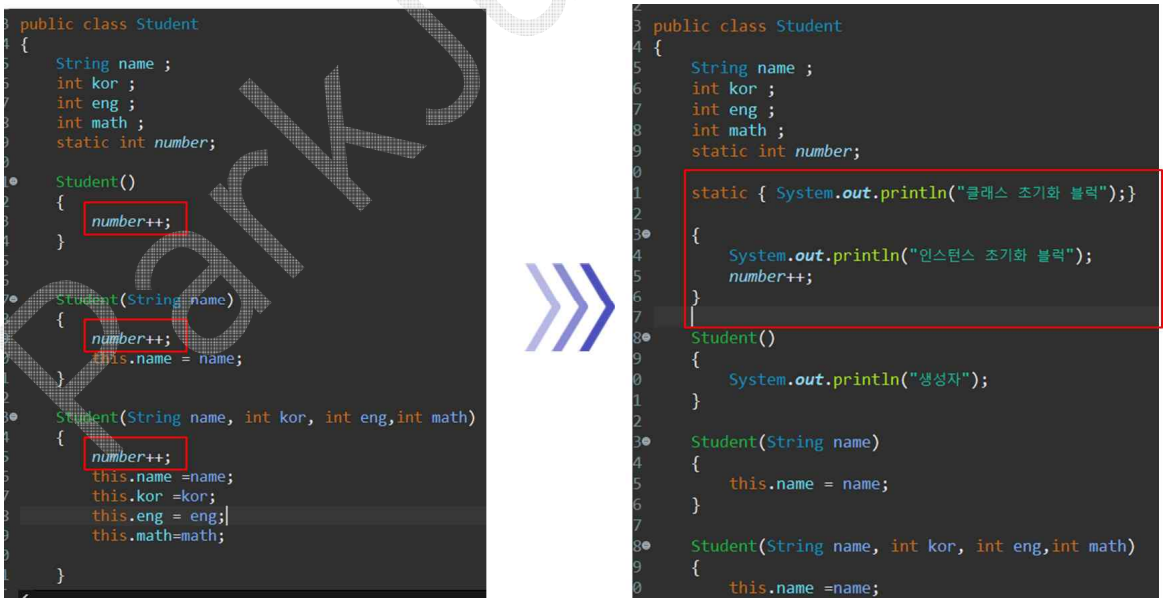

그림 17

그림17의 왼쪽을 보면 3개의 생성자 내부에서 number++를 하는 중복된 소스코드가 있다. 이렇게 모든 생성자에 공통적으로 작성되어야 하는 소스코드가 있을 때 왼쪽과 같이 작성하면 소스코드가 중복이며 변경이 필요할 경우 모든 생성자의 소스코드를 수정해야 한다.

그것을 오른쪽과 같이 초기화 블록 이라는 기능을 활용하여 해결할수 있다.

RankUByeong

초기화 블록에는 인스턴스초기화 블럭과 클래스 초기화 블록 두가지가 존재한다.

1) 클래스 초기화 블록

가) 클래스가 메모리에 처음 로딩 될 때 한번만 실행된다.(프로그램이 실행 될 때)

나) static 키워드를 이용하며 static { 소스코드...} 로 작성한다.

2) 인스턴스 초기화 블록

가) 객체(인스턴스)가 만들어 질 때 마다 생성자보다 먼저 실행된다.

```
public class Student
{
    String name ;
    int kor ;
    int eng ;
    int math ;
    static int number;

    static { System.out.println("클래스 초기화 블럭");}

    {
        System.out.println("인스턴스 초기화 블럭");
        number++;
    }

    Student()
    {
        System.out.println("생성자");
    }

    Student(String name)
    {
        this.name = name;
    }

    Student(String name, int kor, int eng,int math)
    {
        this.name =name;
```

그림 19

```
Student student1 = new Student();

Student student2 = new Student();

Student student3 = new Student();
```

그림 20

```
<terminated> main [Java Applicatio
클래스 초기화 블럭
인스턴스 초기화 블럭
생성자
인스턴스 초기화 블럭
생성자
인스턴스 초기화 블럭
생성자
```

그림20의 실행 결과

그림 18과 같이 인스턴스 초기화 블록과 클래스 초기화블럭을 선언해놨다 그리고 그림19와 같이 Student 객체를 3개 생성하였다.

클래스 초기화블럭이 실행된 타이밍은 객체를 만들기보다 이전인 프로그램이 실행될 때 이미 실행이 한번되었다.

그리고 Student student1 = new Student();를 실행할 때 인스턴스 초기화 블록이 실행되었고 그다음 생성자가 실행되며 객체가 만들어졌다.

그리고 객체가 생성될때마다 인스턴스 초기화 블록이 실행되며 그뒤에 생성자가 실행되어 객체가 만들어진다.

3) 클래스 초기화블럭에서 멤버변수 사용

클래스 초기화블럭 역시 클래스메서드와 동일하게 객체 생성이전에 실행된다. 그러므로 내부에서는 당연히 멤버변수를 사용 할수 없다.

```
public class Student
{
    String name ;
    int kor ;
    int eng ;
    int math ;
    static int number;

    static {
        System.out.println("클래스 초기화 블럭");

        name ="김길동";
    }
```

클래스 초기화 블록은 객체 생성보다 이전에 실행되므로 인스턴스 변수는 사용할수 없다.

그림 21

이렇게 초기화 블록은 생성자들에서 공통적으로 수행되어야 하는 코드들을 작성하는데 사용된다.

ParkJuByeong