2021.05.25 Java

〈생성자는 main에서만 사용 가능한가?〉

NO.

어디에서나 사용 가능하다.

```java
class Students {
    String name;
    int score;

    public Students (String name) {
        score = (int)(Math.random() * 40 + 61);
        this.name = name;
        //School class에서 받은 매개변수로
        //이름: 이주형 점수: xx을 각각 Students의 data field에 저장.
    }
    public String getName() { return name; }
    public int getScore() { return score; }
}
class School {
    final int MAX = 3;
    final String[] names = {"이주형", "이새연", "박동민"};
    Students[] arr;

    public School () {
        arr = new Students[MAX]; // arr[]의 index 생성하고
        for (int i = 0; i < MAX; i++) {
            arr[i] = new Students(names[i]);
            // Students class의 생성자를 이용해서 arr[]의 값들을 설정
            // names[]의 값들은 이름으로 String
            // 즉 Students Class의 생성자의 인자 값으로 활용 할 수 있음.
        }
    }
    public void printStudentsInfo () {
        for (int i = 0; i < MAX; i++) {
            System.out.printf("학생 이름은 %s, 점수는 %d 입니다.\n",
                    arr[i].getName(), arr[i].getScore());
        }
```

```java
public class _1st_ConstructorCallTest {
    public static void main(String[] args) {
        // Q: 생성자는 무조건 main()에서만 호출이 가능한가요 ?
        // A: 어디서나 가능합니다.
        School school = new School();
        school.printStudentsInfo();
```

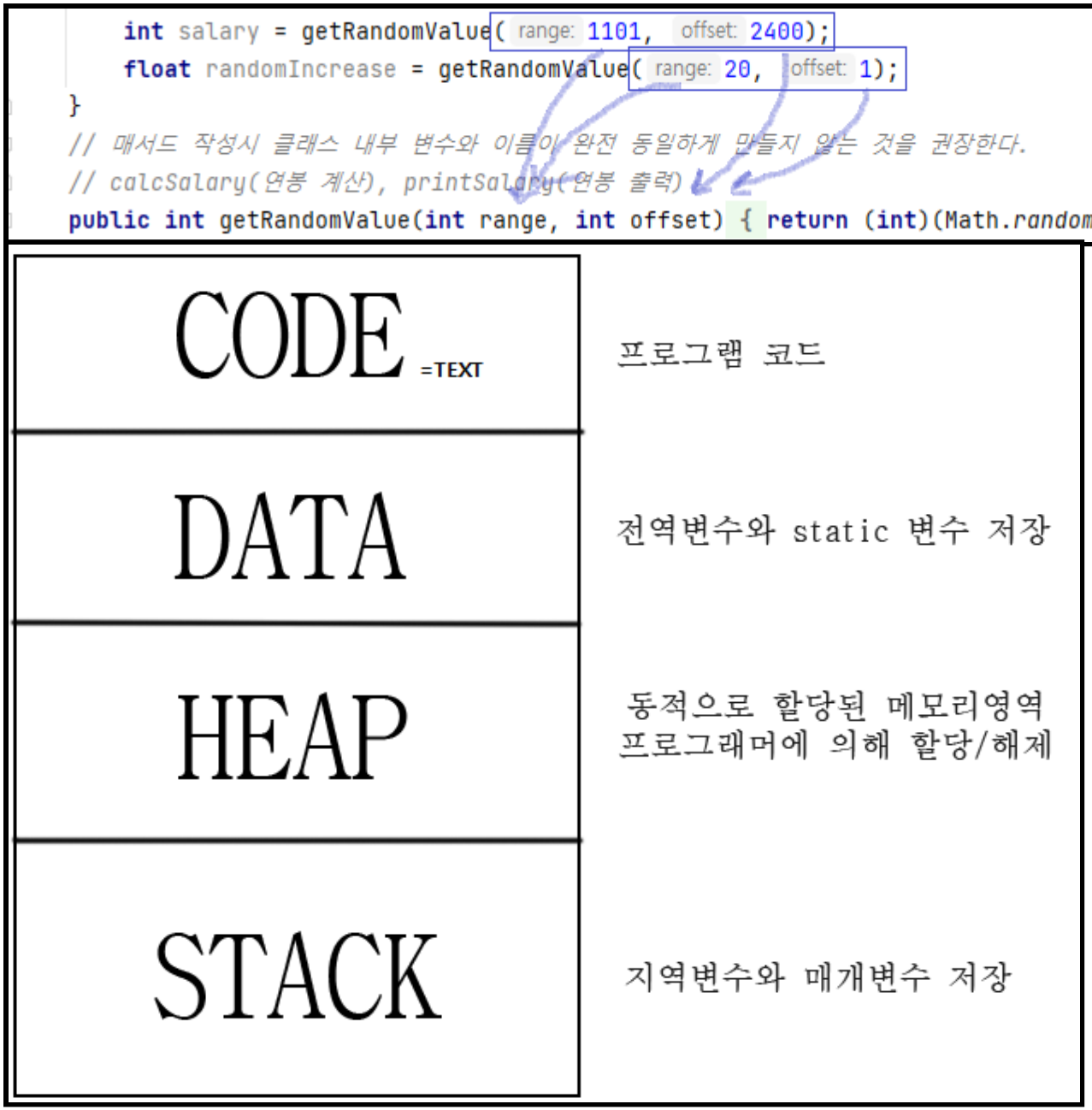
```
"C:\Program Files\Java\jdk-16\bin
학생 이름은 이주형, 점수는 99 입니다.
학생 이름은 이새연, 점수는 93 입니다.
학생 이름은 박동민, 점수는 90 입니다.
```

```java
    int salary = getRandomValue( range: 1101,  offset: 2400);
    float randomIncrease = getRandomValue( range: 20,  offset: 1);
}
// 메서드 작성시 클래스 내부 변수와 이름이 완전 동일하게 만들지 않는 것을 권장한다.
// calcSalary(연봉 계산), printSalary(연봉 출력)
public int getRandomValue(int range, int offset) { return (int)(Math.random() * range + offset); }
```

〈TIP〉

매개변수를 갖는

method 활용.



〈메모리 영역〉

변수 = 컴퓨터 안의 어떤 메모리 공간에 지어진 컨테이너.
그 메모리는 구분해서 사용되며 4가지 영역으로 나뉜다.

Code : 실행한 프로그램의 코드가 저장되어있다.
Data : 전역변수와 static변수가 할당되고 프로그램 종료시까지 남아있다.
Heap: 동적으로 할당된 메모리영역으로 프로그래머에 의해서 할당/해제된다.
Stack: 지역변수와 매개변수가 할당되고 함수를 빠져나가면 변수는 자동 소멸된다.

"동적 할당"과 "정적할당"을 구분하는 기준은
프로그램 실행중(런타임 : 값 대입도 런타임에 일어난다.)에 할당하는지.
컴파일(컴퓨터 언어로 바꾸는 과정)단계에서 공간을 할당하는지에 따라 구분된다.
동적 할당은 런타임 중에. 정적 할당은 컴파일 단계에서.

⟪정적 할당⟫

장점⟩
메모리 누수를 걱정할 필요가 없다. 컴파일해서 메모리 공간을 할당 받으면 어떤 변수가 지정된 공간 (예를 들어 함수)이 사라질 경우 운영체제가 알아서 해당 메모리 공간을 더 이상 사용하지 않는다고 판단해서 메모리를 회수해간다.
단점⟩⟩
메모리 크기가 정해져 있어서 프로그램이 실행되고 있는 도중에 수정할 수가 없다. 데이터 타입을 자세하게 공부할 때 int의 경우 4byte를 차지한다는 정보를 얻었다면 그 4byte가 여기서 사용된다는걸 알고 넘어가자.

위의 메모리 크기가 정해져 있다는게 왜 단점일까?
예를 들어서 처음에는 10개의 공간이 필요했다. 근데 프로그램을 실행하면서 점점 필요한 공간이 줄어드는 경우가 생긴다면? 그때 공간을 낭비하지 않기 위해서 공간을 2개로 바꿔줄 수 있을까? 답은 불가능. 정적 할당은 한 번 크기를 설정해놓은대로 그부분을 사용한다. 메모리 공간의 낭비가 발생할 수 있다. 반대로 처음에는 2개만 필요했는데 나중에는 10개가 필요한 상황이 되면 에러를 일으키게 된다.

《동적 할당》

동적 할당은 런타임중에 메모리 공간을 할당 받는다. 포인터를 사용해서 Heap영역을 가리켜 공간을 할당하는 것.

예를들어 배열의 크기가 확실하지 않고 변동이 있을 때 Heap영역을 활용한다.

공간 낭비가 우려됐던 정적 할당과는 달리 동적 할당은 사이즈를 늘리고 줄이고 할 수 있다. 런타임 중에 할당이 이루어지기 때문에 처음에 필요했던 공간과 프로그램이 어느정도 실행된 후 필요한 공간이 다를경우 공간을 낭비하지 않고 사용할 수 있다.

하지만 동적 할당은 메모리 누수가 발생할 수 있다는 치명적인 단점이 있다. 프로그래머가 직접 할당을 하는 만큼 해제도 직접 해주어야한다.

《STATIC》

메모리에 고정적으로 할당되어. 프로그램이 종료될 때 해제되는 변수를 static 변수라 한다.

```
public class Member {
    private String name = "BB_GG";

}
```

다음과 같은 클래스가 있을 때.
이 클래스를 통해 다른 클래스에서
n개의 Member를 생성하면 "BB_GG"라는 값을
가지는 메모리가 n개가 생겨버린다.
이는 메모리 효율에 매우 좋지않은 방법이므로 이를
해결하기 위해서 아래와 같이 static을 사용하는 것.

```
public class Member {
    public static final String name = "BB_GG";

}
```

다음과 같이 사용하게 되면 n개의
Member를 생성해도 똑같은
메모리에서 값을 참조해쓰기
때문에 메모리 효율에 도움이된다.

```java
class StudyStatic{

    public static void printout_Static(){
        System.out.println("this is static");
    }
    public void printOut_NoneStatic(){
        System.out.println("this is not a static");
    }
}

public class _2nd_Static {
    public static void main(String[] args) {

        StudyStatic.printout_Static();
        StudyStatic.printout_NoneStatic();

        StudyStatic instanceS = new StudyStatic();
        instanceS.printOut_NoneStatic();
    }
}
```

method에서의 static 활용 예제

static method의 경우 new를 통해 instance 생성하지 않아도 main에서 method 호출이 가능.

반대로 static method가 아닌 경우에는 instance를 생성 해야 instance의 method로 호출이 가능

```java
public class _2nd_Static {
    public static void main(String[] args) {

        StudyStatic.printout_Static();
        //StudyStatic.printout_NoneStatic();

        StudyStatic instanceS = new StudyStatic();
        instanceS.printOut_NoneStatic();
    }
}
```

```
_2nd_Static
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaage
this is static
this is not a static
```

static이라는 키워드는 메모리 효율에 있어서 도움을 줄 수 있고.

유틸리티 관련 메소드들을 활용할 때 적합하다.

static method의 대표 예

// int number = (int)Math.pow(2, 2);

// 굳이 특정한 클래스에 소속될 필요없이 필요하면 항상 사용하는 method.

// 이런 method를 utility method라고 하는데

// 이런 utility method들은 static method로 만들어서 사용하곤 한다.

// 필요하다면 class에 static method를 만든다.

// ex) Math.exp() ====> 오일러 상수 e^x 같은 것을 표현할때 쓰는거

// ex) Math.pow() ====> x^y 표현 등등

// 결론적으로 앞의 Math는<<< 수학 관련 라이브러리임을 알려주고

// 뒤에 exp. pow. sqrt 등이 어떤 method인지를 알려준다.

1)

메모리의 영역에는 static(data) 영역과 heap 영역이 존재하는데.

heap 영역은 Garbage Collection 이라고 하는 최적화 방법을 통해 관리 받는 반면. static 영역은 그 반대이다.

static 이라는 키워드를 통해 모든 객체가 공유할 수 있게 되지만. 이는 Garbage Collection 의 관리 영역 밖에 존재하기 때문에

static을 자주 사용하게 된다면 프로그램의 종료시 까지 메모리가 할당 된 채로 존재하게 되므로

시스템에 영향을 끼칠 수 있게되므로 static을 남용하지 않는 것이 좋다.

2)

일반적으로 Class들은 static 영역에 생성되고

new 연산을 통해 생성한 객체들은 heap 영역에 생성된다.

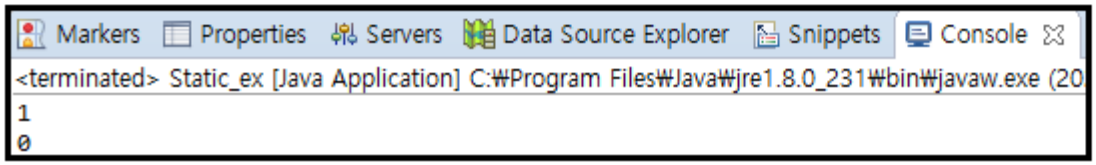
정적(Static) 필드 사용 예시

```java
class Number{
    static int num = 0; //클래스 필드
    int num2 = 0; //인스턴스 필드
}

public class Static_ex {

    public static void main(String[] args) {
        Number number1 = new Number(); //첫번째 number
        Number number2 = new Number(); //두번째 number

        number1.num++; //클래스 필드 num을 1증가시킴
        number1.num2++; //인스턴스 필드 num을 1증가시킴
        System.out.println(number2.num); //두번째 number의 클래스 필드 출력
        System.out.println(number2.num2); //두번째 number의 인스턴스 필드 출력
    }
}
```



Class의 data에 static을 사용했는지 안 했는지
여부에 따라서
class field / instance field로 구분해서
부르니까 뭔가 이해하기가 편하다.

//
number라는 class안에
class변수 num / instance 변수 num2 생성

number class의 instance인
number1 / number2를 생성

number1에서 num과 num2를 각각 1씩 ++

number2에서 num과 num2를 출력하면
num은 1 / num2는 0이 출력된다.

instance 변수는
instance가 생성될 때마다 생성되므로(초기화)
instance마다 각기 다른 값을 가지지만
static 변수는
모든 instance가 하나의 저장공간을 공유하기에
항상 같은 값을 가지기 때문에..

```
// 1. Dicegame의 instance 2개를 생성하고 User와 computer를 대표하도록 한다.
// 2. class 내부에서 위의 rule대로 실행하는 코드 setting.
// 3. 1000만원을 final int로 설정
// 4. Scanner로 판돈 비율 설정 // 이 것도 user와 computer가 같은 값을 공유해야됨.
// 5. 생성자에는 > 주사위를 굴린다
// 6. 주사위 값들을 static 변수로 지정해야 User instance에서 나온 두번째 주사위 값으로
//      computer insatnce에 영향을 줄 수 있을 것 같음
//      그냥 한 instance에서 user / computer 둘 다 처리하기로 바꿈.
// 7. main에는 승자를 결정하는 식만 간단하게
```

《Quiz. 45》

》 두번째 주사위로 첫번째 주사위 값에 영향을 주는 것과
》 단순 승자만 알 수 있는 코딩까지는 했는데
》 시간이 없어서 나머지 부분은 못 만들어 봤습니다
》 수업 참고해서 나머지 부분 마무리하도록 하겠습니다.

```java
import java.util.Scanner;

class DiceGame{
    private final int START = 1000;
    Scanner scan;

    public int user_Dice;
    public int user_Dice2;
    public int computer_Dice;
    public int computer_Dice2;

    public DiceGame(){
        scan = new Scanner(System.in);
        System.out.print("배팅 비율 설정: ");
        int bet_ratio = scan.nextInt();
        user_Dice = getRanDice( range: 6,  offset: 1);
        if_User_Dice();
        computer_Dice = getRanDice( range: 6,  offset: 1);
        if_Computer_Dice();
    }
```

《《《 DiceGame class와 생성자

```java
public void if_User_Dice(){
    if(user_Dice % 2 == 0){
        user_Dice2 =getRanDice( range: 6, offset: 1);
    }
    switch(user_Dice2){
        case 1 : user_Dice = user_Dice + 3;
            break;
        case 3 : computer_Dice = computer_Dice - 2;
            break;
        case 4 : user_Dice = 0;
            break;
        case 6 : user_Dice = (user_Dice + user_Dice2) * 2;
            break;
        default: break;
    }
}
```

<<< if_User_Dice method

```java
public void if_Computer_Dice(){
    if(computer_Dice % 2 == 0){
        computer_Dice2 =getRanDice( range: 6, offset: 1);
    }
    switch(computer_Dice2){
        case 1 : computer_Dice = computer_Dice + 3;
            break;
        case 3 : user_Dice = user_Dice - 2;
            break;
        case 4 : computer_Dice = 0;
            break;
        case 6 : computer_Dice = (computer_Dice + computer_Dice2) * 2;
            break;
        default: break;
    }
}
public int getRanDice(int range, int offset) { return (int)(Math.random() * range + offset); }
public int getUser_Dice() { return user_Dice; }
public int getComputer_Dice() { return computer_Dice; }
public int getUser_Dice2() { return user_Dice2; }
public int getComputer_Dice2() { return computer_Dice2; }
```

<<< if_Computer_Dice method

⟨⟨⟨ main과 승자출력

```java
public class _4th_Quiz45 {
    public static void main(String[] args) {
//...

        DiceGame casino = new DiceGame();

        if(casino.getUser_Dice() > casino.getComputer_Dice()){
            System.out.printf("user:%d / computer%d\n", casino.getUser_Dice(),casino.getComputer_Dice());
            System.out.println("user win!");
        } else if(casino.getUser_Dice() == casino.getComputer_Dice()) {
            System.out.printf("user:%d / computer%d\n", casino.getUser_Dice(),casino.getComputer_Dice());
            System.out.println("draw");
        } else {
            System.out.printf("user:%d / computer%d\n", casino.getUser_Dice(),casino.getComputer_Dice());
            System.out.println("computer win!");
        }
    }
}
```