

2021.05.14 Java

<2020513_Quiz23 풀이 리뷰>

```
public class _4th_Quiz23 {
    public static void main(String[] args) throws InterruptedException {

        // Quiz 23.
        // 22번 문제(1st_NonDuplicateWithoutArrayTest)의 경우엔
        // random number(난수)의 범위가 0 ~ 9 였다.

        // 이번에는 2개의 범위를 가지는 random number 2개를 중복없이 제어해볼 것.
        // 하나는 5 ~ 10의 범위를 가지고 다른 하나는 7 ~ 10의 범위를 가진다.
        // 22번 예제를 응용하여 풀 수 있는 문제고 다소 난이도가 높은 문제다.
        // ex) 5, 6, 7, 8, 9, 10이 모두 출력되어야 하고 또한 7, 8, 9, 10이 출력되어야 한다.
        // 여기서 중복이 발생하면 안됨

        final int FIRST_BIAS = 5; // 변수 BIAS 활용하는 이유는 아래 설명
        final int SECOND_BIAS = 1;

        // int는 4바이트이므로 32비트에 해당하는 데이터를 저장할 수 있음
        // 우리가 testBit를 가지고 제어할 수 있는 비트의 개수는 32개
        // 숫자가 많은것이 아니므로 32개의 공간을 최대한 효율적으로 활용해야 할 것임
        // (32개라서 1개가 아까운 상황이라고 보면 됨)
        // 이 상황에서 되도록이면 0번 비트부터 순차적으로 활용하고 싶을것이고 중간에 비는 공간이 없길 원할 것임
        // 이 이유로 위의 변수 BIAS 활용

        final int FIRST_RANGE = 6; // 5~10 6개
        final int FIRST_OFFSET = 5;
        // 0<=Math.random() * 6< 6 이기 때문에 +5로 5, 6, 7, 8, 9, 10의 정수가 나오도록함.

        final int SECOND_RANGE = 4; // 7~10 4개
        final int SECOND_OFFSET = 7;
        // 0<=Math.random() * 4< 4 이기 때문에 +7로 7, 8, 9, 10의 정수가 나오도록함.
```

```
final int BIN = 1;
int testBit = 0;
int randNum;
```

```
// 5 ~ 10 ---> 6개 A조
for (int i = 0; i < FIRST_RANGE; i++) {
    // 5 ~ 10
    // 실제 출력값은 5~10을 사용하고, 비트연산에서는 0번~5번 비트를 사용>>> BIAS 이용
    // 2^5, 2^6, 2^7, 2^8, 2^9, 2^10
    // 2^0, 2^1, 2^2, 2^3, 2^4, 2^5
    randNum = (int)(Math.random() * FIRST_RANGE + FIRST_OFFSET);
    while ((testBit & (BIN << (randNum - FIRST_BIAS))) != 0) {
        randNum = (int)(Math.random() * FIRST_RANGE + FIRST_OFFSET);
    }
    System.out.printf("5 ~ 10 randNum = %d\n", randNum);
    testBit |= (BIN << (randNum - FIRST_BIAS));
}
System.out.println("testBit = " + testBit);

// 7 ~ 10 ---> 4개 B조
for (int i = 0; i < SECOND_RANGE; i++) {
    // 7 ~ 10
    // 출력되는 숫자는 7 ~ 10을 사용하고, 비트 연산에는 6번 ~ 9번 비트를 사용.
    // 2^7, 2^8, 2^9, 2^10
    // 2^6, 2^7, 2^8, 2^9
    randNum = (int)(Math.random() * SECOND_RANGE + SECOND_OFFSET);

    while ((testBit & (BIN << (randNum - SECOND_BIAS))) != 0) {
        randNum = (int)(Math.random() * SECOND_RANGE + SECOND_OFFSET);
    }

    System.out.printf("7 ~ 10 randNum = %d\n", randNum);

    testBit |= (BIN << (randNum - SECOND_BIAS));
}
System.out.println("testBit = " + testBit);
}
```

```
// 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
// 0 0 0 0 0 0 0 0 0 0
```

변수 BIAS를 활용해서 A조가 사용
변수 BIAS를 활용해서 B조가 사용

하나의 testBit 변수를 이용하기 위해서

변수 BIAS의 값을 적절하게 정할 필요가 있음.

잘못하면 A조 while문 끝나고 무한루프 걸림.

또한, BIAS 값을 잘못 설정해서 (randNum - BIAS)의 값이 음수로 나오게 되면 코드 망가질 수 있음.

```
Run: _4th_Quiz23 x
"C:\Program Files\Java\jdk-16\bin
5 ~ 10 randNum = 10
5 ~ 10 randNum = 7
5 ~ 10 randNum = 8
5 ~ 10 randNum = 9
5 ~ 10 randNum = 6
5 ~ 10 randNum = 5
testBit = 63
7 ~ 10 randNum = 10
7 ~ 10 randNum = 8
7 ~ 10 randNum = 9
7 ~ 10 randNum = 7
testBit = 1023

Process finished with exit code 0
```

```

public class _4th_Quiz23_Selfstudy2 {
    public static void main(String[] args) {

        int randNum1, randNum2;
        int testBit1 = 0, testBit2 = 0;
        int BIN = 1;

        for(int i = 0; i < 6; i++){
            randNum1 = (int)(Math.random() * 6 + 5);
            while((testBit1 & (BIN << randNum1)) != 0){
                randNum1 = (int)(Math.random() * 6 + 5);
            }
            System.out.println(randNum1);
            testBit1 |= (BIN << randNum1);
        }

        for(int i = 0; i < 4; i++){
            randNum2 = (int)(Math.random() * 4 + 7);
            while((testBit2 & (BIN << randNum2)) != 0){
                randNum2 = (int)(Math.random() * 4 + 7);
            }
            System.out.println(randNum2);
            testBit2 |= (BIN << randNum2);
        }
    }
}

```

나는 변수 BIAS를 활용해서 2진수 자리 배치하고 중복 피하는게 너무 헛갈려서

그냥 testBit1 / testBit2 나눠서 사용함.

그리고 OFFSET / RANGE / BIAS를 변수로 지정해서 사용하는게 아직은 헛갈려서

그냥 정수로 표기함.

```

_4th_Quiz23_Selfstudy2 ×
"C:\Program Files\Java\jdk-1
10
7
6
5
9
8
10
7
9
8

```

```

public class _4th_Quiz23_Selfstudy3 {
    public static void main(String[] args) throws InterruptedException {

        int randNum1, randNum2;
        int testBit= 0;
        int BIN = 1;

        for(int i = 0; i < 6; i++){
            randNum1 = (int)(Math.random() * 6 + 5);
            while((testBit & (BIN << randNum1)) != 0){
                randNum1 = (int)(Math.random() * 6 + 5);
            }
            System.out.println(randNum1);
            testBit |= (BIN << randNum1);
        }

        testBit = 0;

        for(int i = 0; i < 4; i++){
            randNum2 = (int)(Math.random() * 4 + 7);
            while((testBit & (BIN << randNum2)) != 0){
                randNum2 = (int)(Math.random() * 4 + 7);
            }
            System.out.println(randNum2);
            testBit |= (BIN << randNum2);
        }
    }
}

```

꼭 testBit를 하나만 쓰려면
그냥 A조 for문 끝나고 testBit를
0으로 초기화 하는 방법도 있음.

내가 헛갈려서 머리가 깨지는 것 보다
그냥 컴퓨터 보러 열심히 일 하라고 하는게 편한 것 같음

Run: _4th_Quiz23_Selfstudy3

6
7
5
8
10
9
9
7
10
8

```

import java.util.Scanner;

public class _1st_SwitchTest {
    public static void main(String[] args) {
        Boolean isTrue = true; // Boolean: 참, 거짓을 표현하는 Datatype.
        Scanner scan = new Scanner(System.in);
        int num;

        while (isTrue) {
            System.out.print("숫자를 눌러 물건을 담으세요: ");
            num = scan.nextInt();
            // 입력된 키보드 값에 따라 적절한 처리를 하게 된다.
            // 키보드 값에 따라 처리하는 루틴은 case x에 해당한다.
            // 0번이 눌렀다면 case 0, 1번이라면 case1과 같은 형식.

            switch (num) {
                case 0:
                    System.out.println("탈출합니다.");
                    isTrue = false;
                    break;
                case 1:
                    System.out.println("비누를 장바구니에 담았습니다.");
                    break;
                    // break; 는 더 이상 다음을 실행하지 않고 그 시점에서 종료하는 코드
                    // 여기서 break가 없으면 비누/신발 다 담아짐.
                    // Switch case의 경우 break 사용 잘 활용 하는거 항상 인지.
                case 2:
                    System.out.println("신발을 장바구니에 담았습니다.");
                    break;
                case 3:
                    System.out.println("에어팟을 장바구니에 담았습니다.");
                    break;
                default:
                    // default: 기본값. 우리가 예상하지 못한 입력이 존재할 수 있는데 이 때 활용됨.
                    System.out.println("그런건 없습니다");
                    break;
            }
        }
    }
}

```

<Switch_Case>

```

_1st_SwitchTest x
"C:\Program Files\Java\jdk-16\bin\java.exe" -
저희 상점에 방문해주셔서 감사합니다. 물건을 고르세요
숫자를 눌러 물건을 담으세요: 1
비누를 장바구니에 담았습니다.
숫자를 눌러 물건을 담으세요: 2
신발을 장바구니에 담았습니다.
숫자를 눌러 물건을 담으세요: 3
에어팟을 장바구니에 담았습니다.
숫자를 눌러 물건을 담으세요: 4
그런건 없습니다
숫자를 눌러 물건을 담으세요: 0
탈출합니다.

```

<Switch_Case with String>

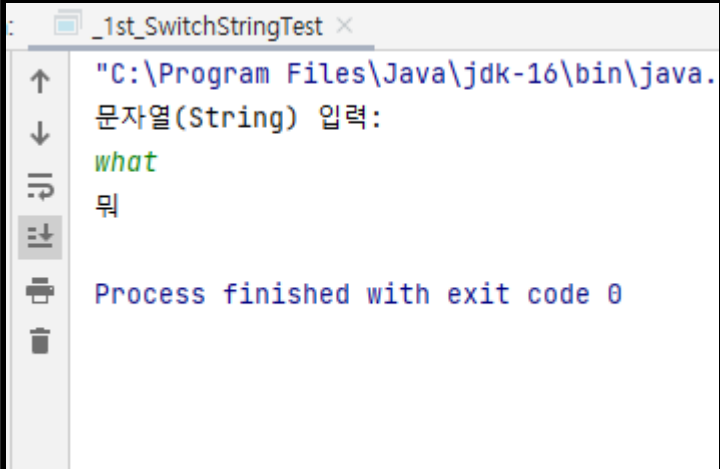
```
import java.util.Scanner;

public class _1st_SwitchStringTest {
    public static void main(String[] args) {

        System.out.println("문자열(String) 입력: ");
        Scanner scan = new Scanner(System.in);

        String str = scan.nextLine();

        // 주의점: switch에 사용되는 DataType과
        // case에서 사용하는 DataType을 일치시켜야 한다.
        switch (str){
            case "hi":
                System.out.println("안녕");
                break;
            case "what":
                System.out.println("뭐");
                break;
            default:
                System.out.println("???");
                break;
        }
    }
}
```



```
_1st_SwitchStringTest x
"C:\Program Files\Java\jdk-16\bin\java.
문자열(String) 입력:
what
뭐
Process finished with exit code 0
```

```
_2nd_ContinueTest.java × _99th_0512ChallengeQuiz10.java × _4th_ArrayTest.java × _3rd_Quiz25
1 ▶ public class _2nd_ContinueTest {
2 ▶     public static void main(String[] args) {
3     for (int i = 0; i < 10; i++){
4         if (i % 2 == 0){
5             // continue를 만나면 아래쪽에 진행해야하는 코드가 남아있더라도
6             // 무조건 for loop의 최상단으로 이동하게 한다.
7             // 따라서 증감식이 진행됨.
8             continue;
9         }
10        System.out.println("i = " + i);
11    }
12 }
13
14
```

```
_2nd_ContinueTest ×
"C:\Program Files\Java\jdk-16\bin\java
i = 1
i = 3
i = 5
i = 7
i = 9

Process finished with exit code 0
```

<Continue>

```

public class _3rd_Quiz25 {
    public static void main(String[] args) {
        // 1 ~ 100 까지의 숫자 중,
        // 2의 배수와 11의 배수는 더하고 5의 배수는 뺀다.
        // 단, 상위 조건 중 중복으로 적용되는 수는 무시한다.
        // ex) 22는 제외
        // 모든 값을 처리한 이후 결과값은 무엇인지 프로그래밍 한다.
        int sum = 0;

        for(int i = 1; i <= 100; i++){
            if(i % 2 == 0 && i % 5 == 0 && i % 11 == 0){
                ;
            } else if(i % 2 == 0 && i % 5 == 0){
                ;
            } else if(i % 5 == 0 && i % 11 == 0){
                ;
            } else if(i % 2 == 0 && i % 11 == 0){
                ;
            } else if(i % 2 == 0){
                sum+=i;
            } else if(i % 5 == 0){
                sum-=i;
            } else if(i % 11 == 0){
                sum+=i;
            }
        }
        System.out.println("최종 합산: " + sum);
    }
}

```

<Quiz25>

Q. 1~200(또는 그 이상)인 경우를 예를 들었을 때
 2의 배수 & 5의 배수 & 11의 배수에 해당하는 숫자는
 어짜피 2의 배수 & 5의 배수인 조건에 해당하는 숫자이고
 제외되기 때문에
 굳이 $(i \% 2 == 0 \ \&\& \ i \% 5 == 0 \ \&\& \ i \% 11 == 0)$ 가 필요한지?

2의 배수 & 5의 배수 & 11의 배수에 해당하는 수 제외
 >> ;는 아무 것도 실행하지 않는다

2의 배수 & 5의 배수 해당하는 수 제외

5의 배수 & 11의 배수에 해당하는 수 제외

2의 배수 & 11의 배수에 해당하는 수 제외

2의 배수인 것은 sum에 더하고

5의 배수인 것은 sum에서 빼고

11의 배수인 것은 sum에 더한다.


```

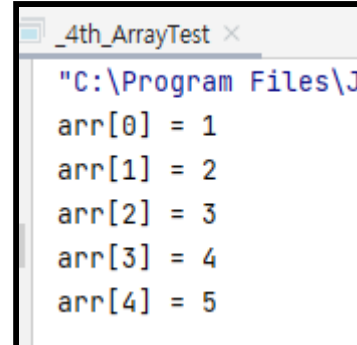
public class _4th_ArrayTest {
    public static void main(String[] args) {

        // 배열을 사용하는 이유:
        //     >> 동일한 데이터 타입의 변수가 여러개 필요할 때
        // 배열을 만드는 방법
        // 1. stack에 할당하는 방법
        //     1-1. 배열의 데이터 타입을 적는다 >> int
        //     1-2. 배열의 이름이 될 변수명을 적는다 >> arr
        //     1-3. 배열임을 알리기 위해 []을 변수 옆에 적는다. >> []
        //     1-4. 필요하다면 배열의 값들을 초기화한다. {1, 2, 3, 4, 5};
        //         (이 때 원소로 지정한 숫자에 따라 배열의 길이가 지정된다.)
        //     *stack에 할당한다는 것은 지역변수로 처리함을 의미
        //         따라서 나중에 method나 class를 학습한 이후 스택에 할당하면
        //         해당 method 또는 class 내부에서만 해당 배열이 활성화된다.
        //     *가변으로 구성하고 싶다면 new를 사용해야 하는데 이것은 다음에 학습.
        int arr[] = { 1, 2, 3, 4, 5 };
        // 이 data는 아래와 같은 형식으로 저장된다.
        //     -----
        // arr | 1 | 2 | 3 | 4 | 5 |
        //     -----
        //     [0] [1] [2] [3] [4]
        // 배열의 index(방) 번호는 0번부터 시작함에 주의한다.
        // 방 번호가 순차적으로 증가하기 때문에 for문이나 while문과의 혼합구성에 있어 매우 좋다.

        for (int i = 0; i < 5; i++){
            System.out.printf("arr[%d] = %d\n", i, arr[i]);
        }
    }
}

```

<배열 Array>



```

_4th_ArrayTest x
"C:\Program Files\J
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5

```

stack(지역변수)에 할당한다는 것은
지역변수로 처리함을 의미

그렇기 때문에 나중에 method나 class를
학습한 이후
스택에 할당하면 해당 method 혹은 class
내부에서만 해당 배열이 활성화됨.

<Quiz10 review>

```
public class _99th_0512ChallengeQuiz10 {  
    public static void main(String[] args) {  
        // 1000개의 case가 있다  
        // A case 700개 / B case 240개  
        // C case 30개 / D case 350개 / F case 500개  
        // 'C 또는 F면 통과'라고 할 때, 이를 판정하는 횟수.  
        // if (case F || case C)  
        // F case 500 + 나머지 case 500 + 추가검사 500개  
        // if (case C || case F)  
        // C case 30개 + 나머지 case 970개 + 추가검사 970개
```

```
// 위의 시나리오를 숫자중의 배수 판별하는 시나리오로 바꾸면  
// 1 ~ 1000까지의 숫자중 2의 배수는 F  
// 1 ~ 1000까지의 숫자중 33의 배수는 C  
int bigFrontCnt = 0, smallFrontCnt = 0;  
  
for (int i = 1; i <= 1000; i++) {  
    // if (i % 2 == 0 || i % 33 == 0) {  
    //     cnt++;  
    // }  
    // 주석으로 막은 if문에서는, 하나만 검사할때도 cnt가 증가,  
    // 둘 다 검사할때도 cnt가 증가.  
    // 확인하고자 하는 것은 검사가 총 몇 번 발생가? 이다.  
    // 그러므로 각 검사마다 값의 cnt(카운트)값을 증가시켜야함.  
  
    // 따라서 각 검사마다 카운트를 할 수 있도록  
    // 카운트를 하는 코드와 검사 코드를 하나로 묶는다.  
    // 카운트 하는 코드는 전위연산자(++a)로 배치하여 무조건 0이 아니게 만들면 참이다. >> cnt 1씩 증가  
    // AND 연산의 특성상 뒤의 조건을 확인해야하므로, 무조건 cnt는 증가하고 뒤의 조건을 확인하게 된다.  
    if (((++bigFrontCnt != 0) && (i % 2 == 0)) || ((++bigFrontCnt != 0) && (i % 33 == 0))) {  
        ;  
    }  
    if (((++smallFrontCnt != 0) && (i % 33 == 0)) || ((++smallFrontCnt != 0) && (i % 2 == 0))) {  
        ;  
    }  
}  
System.out.println("큰놈이 앞에 있을때: " + bigFrontCnt);  
System.out.println("작은놈이 앞에 있을때: " + smallFrontCnt);
```

```
_99th_0512ChallengeQuiz10 ×  
"C:\Program Files\Java\jdk-16\bin\java.exe"  
큰놈이 앞에 있을때: 1500  
작은놈이 앞에 있을때: 1970  
  
Process finished with exit code 0
```

즉, 큰(많은) case를 먼저 했을 때 더 효율적

```

public class _99th_0512ChallengeQuiz10_Selfstudy {
    public static void main(String[] args) {
        |
        int bigFrontCnt = 0, smallFrontCnt = 0;
        for (int i = 1; i <= 10; i++) {
            if (((++bigFrontCnt != 0) && (i % 2 == 0)) || ((++bigFrontCnt != 0) && (i % 3 == 0))) {
                ;
            }
            if (((++smallFrontCnt != 0) && (i % 3 == 0)) || ((++smallFrontCnt != 0) && (i % 2 == 0))) {
                ;
            }
        }
        System.out.println("큰놈이 앞에 있을때: " + bigFrontCnt);
        System.out.println("작은놈이 앞에 있을때: " + smallFrontCnt);
    }
}

```

수를 1~10으로 줄이고 2의 배수와 3의 배수를 이용해서 비교해봄.

```

"C:\Program Files\Java\jdk-16\bin\java.
큰놈이 앞에 있을때: 15
작은놈이 앞에 있을때: 17

Process finished with exit code 0

```

5 // 2의배수인가? | 3의 배수인가? (체크하는지 안 하는지의 여부)

//1 Check Check

//2 Check

//3 Check Check

//4 Check

//5 Check Check

//6 Check

//7 Check Check

//8 Check

//9 Check Check

//10 Check

// 총 15번 Check

// 3의배수인가? | 2의 배수인가? (체크하는지 안 하는지의 여부)

//1 Check Check

//2 Check Check

//3 Check

//4 Check Check

//5 Check Check

//6 Check

//7 Check Check

//8 Check Check

//9 Check

//10 Check Check

6 // 총 17번 Check