

<12.24 복습>

- 문제1. 65 ~ 122 사이의 랜덤한 문자를 생성하도록 한다.

여기서 소문자나 대문자가 아니라면 다시 생성하도록 프로그램을 만들어보자

- 구현 전략

문자와 숫자가 대응되는 ASCII 코드 이용

반복문 while 안에서 난수를 발생시키기

```
while (isChar) {  
    int rand = (int) (Math.random() * range + MIN);
```

난수 발생 범위 : 65 ~ 122 사이 -> (int)(Math.random() * (122-65 +1) + 65))

```
final int MAX = 122;  
final int MIN = 65;  
  
int range = MAX - MIN + 1;
```

while의 종료 조건: 소문자나 대문자에 해당하는 난수가 발생 (대문자조건 or 소문자조건)

[대문자 조건 : 65<= 난수 <= 90]

[소문자 조건 : 97<= 난수 <=122]

난수가 발생한 뒤 조건을 판별하기 위해 if문 사용

-> 발생된 난수에 해당하는 문자가 [소문자 OR 대문자] [문자가 아닌지] 판별

```
boolean condition1 = rand >= 65 && rand <= 90; //대문자조건  
boolean condition2 = rand >= 97 && rand <= 122; //소문자조건  
  
if (condition1 || condition2) {  
    if(condition1) {  
        System.out.printf("rand는 영문자 대문자중 하나임: %c(%d)\n", rand, rand);  
        isChar = false;  
        // break를 걸면 loop(반복)을 바로 빠져나감  
        break;  
    }  
    if(condition2) {  
        System.out.printf("rand는 영문자 소문자중 하나임: %c(%d)\n", rand, rand);  
        isChar = false;  
        break;  
    }  
}  
  
System.out.printf("문자가 아님: %c(%d)\n", rand, rand);
```

- 문제2. 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

일명 피보나치 수열의 20번째 항을 구하도록 프로그램 해보자!

- 구현 전략

피보나치 수열 : 첫째 및 둘째 항이 1이며 그 뒤의 모든 항은 바로 앞 두 항의 합인 수열

고정된 값은 2개, 고정된 2개의 값을 통해 다음 항의 값을 계산

3개의 변수를 사용, 첫 번째 변수의 초기값 = 1, 두 번째 변수의 초기값 = 1, 세 번째 변수 초기값 = 0

```
int first = 1;
int second = 1;
int result = 0, i;
```

20번째 항까지 값을 구하기 위해 반복해서 더해주는 작업이 필요 -> for문 이용

```
for (i = START; i < 20; i++) {
```

for문의 초기값은 2로 설정

: 첫 번째 항과 두 번째 항의 값은 1로 고정되어 있기 때문에 세 번째 항부터 시작해야 함.

하지만 컴퓨터에서 배열의 시작은 0, 반복 시작을 0을 기준으로 맞춰야 한다.

-> 초기값 2가 의미하는 것은 결국 세 번째 항

```
final int START = 2;
```

for문 내부에서 반복 수행할 동작은 피보나치 수열의 규칙대로 작성

```
result = first + second;
```

3항에 대한 계산이 완료되었기 때문에 다음 항에 대한 처리를 위해 변수값 재설정

```
first = second;
second = result;
```

- 문제3. 1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, 41, 60, 88, 129, ...

이와 같은 숫자의 규칙을 찾아 25번째 항을 구하도록 프로그램 해보자!

- 구현 전략

초황금 수열(supergolden sequence)

: 처음 세 항은 1이고, 다음 항의 값은 바로 이전 항과 이전 항보다 2개 전 항을 더한 값이다.

고정된 값은 3개, 고정된 3개의 값을 통해 다음 항의 값을 계산

4개의 변수를 사용

```
int first = 1;
int second = 1;
int third = 1;
int result = 0, i;
```

25번째 항까지 값을 구하기 위해 반복해서 더해주는 작업이 필요 -> for문 이용

```
for (i = START; i < END; i++) {
```

for문의 초기값은 3로 설정

: 첫 번째 항, 두 번째 항, 세 번째 항의 값은 1로 고정되어 있기 때문에 네 번째 항부터 시작해야 함.

하지만 컴퓨터에서 배열의 시작은 0, 반복 시작을 0을 기준으로 맞춰야 한다.

-> 초기값 3이 의미하는 것은 결국 네 번째 항

```
final int START = 3;
final int END = 25;
```

for문 내부에서 반복 수행할 동작은 초황금 수열의 규칙대로 작성

```
result = first + third;
```

4항에 대한 계산이 완료되었기 때문에 다음 항에 대한 처리를 위해 변수값 재설정

```
first = second;
second = third;
third = result;
```

- 문제4. 구구단 7단을 출력해보자.

- 구현 전략

구구단 7단을 모두 출력하기 위해 반복문 for 사용

7 X 1 ~ 7 X 9에 대한 결과를 모두 보여주기 위해 for문 제어변수의 값을 1부터 9까지 증가시켜야 한다.
for문 제어변수의 초기값은 1, 증가된 변수의 값이 10보다 작을 때까지 반복

```
final int STAGE = 7;  
final int START = 1;  
final int END = 10;
```

for문에서 반복시킬 동작 -> 출력문 printf

```
for (int i = START; i < END; i++) {  
    System.out.printf("%d x %d = %d\n", STAGE, i, STAGE * i);  
}
```

- 문제5. 1~100까지 숫자 중 짝수만 출력해보자!

- 구현 전략

1~100까지 값을 증가시켜야 한다. -> for문 사용

```
final int START = 1;  
final int END = 100;
```

1부터 100까지 짝수에 해당하는 숫자가 있는지 확인하기 위한 조건이 필요하다.

-> 나머지를 구하는 연산자(%)를 이용-> 변수 % 2 == 0 에 부합하면 짝수.

DECISION = 2, REMAIN = 0

```
final int DECISION = 2;  
final int REMAIN = 0;  
if (i % DECISION == REMAIN) {
```

그리고 이 조건에 대한 판별은 값을 하나 증가시킬 때마다 수행되어야 한다.

-> for문 내부에 if문을 배치해 반복시킨다.

if문 조건이 참일 경우 수행할 동작 : 짝수에 해당하는 숫자를 출력

```
for (int i = START; i <= END; i++) {  
    if (i % DECISION == REMAIN) {  
        System.out.println("짝수 i = " + i);  
    }  
}
```

- 문제6. 1~100까지 숫자 중 3의 배수만 출력해보자!

- 구현 전략

1~100까지 값을 증가시켜야 한다. -> for문 사용

```
final int START = 1;  
final int END = 100;
```

1부터 100까지 3의 배수에 해당하는 숫자가 있는지 확인하기 위한 조건이 필요하다.

-> 나머지를 구하는 연산자(%)를 이용-> 변수 % 3 == 0 에 부합하면 3의 배수.

DECISION = 3, REMAIN = 0

```
final int DECISION = 3;  
final int REMAIN = 0;  
if (i % DECISION == REMAIN) {
```

그리고 이 조건에 대한 판별은 값을 하나 증가 시킬 때마다 수행되어야 한다.

-> for문 내부에 if문을 배치해 반복시킨다.

if문 조건이 참일 경우 수행할 동작 : 3의 배수에 해당하는 숫자를 출력

```
for (int i = START; i <= END; i++) {  
    if (i % DECISION == REMAIN) {  
        System.out.println("3의 배수 i = " + i);  
    }  
}
```

- 문제7. 1~100까지 숫자 중 4의 배수를 더한 결과를 출력해보자!

- 구현 전략

1~100까지 값을 증가시켜야 한다. -> for문 사용

```
final int START = 1;  
final int END = 100;
```

4의 배수들의 합을 저장시킬 변수 필요 -> sum = 0

```
int sum = 0;
```

1부터 100까지 4의 배수에 해당하는 숫자가 있는지 확인하기 위한 조건이 필요하다.

-> 나머지를 구하는 연산자(%)를 이용-> 변수 % 4 == 0 에 부합하면 4의 배수.

```
final int DECISION = 4;  
final int REMAIN = 0;  
  
if (i % DECISION == REMAIN) {
```

그리고 이 조건에 대한 판별은 값을 하나 증가 시킬 때마다 수행되어야 한다.

-> for문 내부에 if문을 배치해 반복시킨다.

if문 조건이 참일 경우 수행할 동작 : sum에 현재 값 더하기

```
for (int i = START; i <= END; i++) {  
    if (i % DECISION == REMAIN) {  
        System.out.printf("%d의 배수 i = %d\n", DECISION, i);  
        sum += i;  
    }  
}
```

1~100까지의 반복이 끝나면 더해진 값을 출력시킨다.

```
System.out.printf("%d ~ %d까지 %d의 배수들의 합은 %d\n", START, END, DECISION, sum);
```

- 문제8. 1~100까지 숫자를 순회한다.

2~10 사이의 랜덤한 숫자를 선택하고 이 숫자의 배수를 출력해보도록 한다.

- 구현 전략

어떤 수의 배수를 출력시킬지 2~10 사이 난수 발생을 통해 결정

2~10 사이의 난수를 발생시키기 위해 -> (int)(Math.random() * (10-2 +1) + 2))

```
final int MAX = 10;
final int MIN = 2;

int range = MAX - MIN + 1;
int decision = (int) (Math.random() * range + MIN);
```

1~100까지 값을 증가시켜야 한다. -> for문 사용

1부터 100까지 선택된 난수의 배수에 해당하는 숫자가 있는지 확인하기 위한 조건이 필요하다.

-> 나머지를 구하는 연산자(%)를 이용-> 변수 % 선택된 난수 == 0 에 부합하면 선택된 난수의 배수.

```
final int START = 1;
final int END = 100;
final int REMAIN = 0;

if (i % decision == REMAIN) {
```

그리고 이 조건에 대한 판별은 값을 하나 증가 시킬 때마다 수행되어야 한다.

-> for문 내부에 if문을 배치해 반복시킨다.

if문 조건이 참일 경우 수행할 동작 : 선택된 난수의 배수를 출력

```
for (int i = START; i <= END; i++) {
    if (i % decision == REMAIN) {
        System.out.printf("%d의 배수 i = %d\n", decision, i);
    }
}
```


- 문제9. 1 ~ 100까지의 숫자를 순회한다.

2 ~ 10 사이의 랜덤한 숫자를 선택하고 이 숫자의 배수를 출력한다.

다음 루프에서 다시 랜덤 숫자를 선택하고 해당 숫자의 배수를 출력한다.

그 다음 루프에서 다시 작업을 반복한다.

끝까지 순회 했을때 출력된 숫자들의 합은 얼마인가 ?

- 구현 전략

1~100까지의 숫자를 순회하며, 다음 숫자들 중 선택된 난수의 배수값 중 최소값으로 이동하고 이 값을 출력한다. 마지막에는 출력된 값들의 합을 보여준다.

1부터 100까지 값이 증가하며, 이를 위해 for문을 사용한다.

```
final int START = 1;
final int END = 100;
```

매 루프마다 1. 2~10 사이의 난수 발생작업과

2. 현재값이 선택된 난수의 배수인지 판별하는 작업이 필요하다.

난수 발생 후 현재 증가된 값이 선택된 난수의 배수가 아니라면,

다음에 수행되는 루프들에서 난수를 다시 뽑지 않고, 이전에 선택된 난수의 배수가 나올 때까지 값을 증가시킨다.

선택된 난수의 배수가 나오면 해당 숫자를 출력하고 다음 루프로 넘어가 난수를 다시 발생시킨다.

출력된 값들의 합을 저장하기 위한 변수 sum이 필요하다.

```
int sum = 0;
```

작동을 위해 2가지 조건이 필요하다.

1. 난수발생조건

: 난수 발생은 한 번 실행되고 난 뒤, 증가되는 값에서 해당 난수의 배수를 찾기 전까지 다시 실행되면 안 된다.

매 루프마다 난수발생 구문을 제어하기 위해 true나 false 값을 갖는 논리형 변수가 필요하다.

```
boolean isRandomAllocCheck = false;
```

2. 현재값(i)이 선택된 난수의 배수에 해당하는지 판별하는 조건

- 2~10 사이의 난수를 발생시키기 위해 -> (int)(Math.random() * (10-2 +1) + 2))

- 나머지를 구하는 연산자(%)를 이용-> 현재값 % 선택된 난수 == 0 에 부합하면 선택된 난수의 배수.

```
final int MAX = 10;
final int MIN = 2;
int range = MAX - MIN + 1;

int decision = 0;
decision = (int) (Math.random() * range + MIN);

if (i % decision == 0)
```

for문의 매 루프에서 수행될 동작 :

1. 난수 발생
2. 현재값이 선택된 난수의 배수에 해당하는지 판별하는 작업

- 1. 난수 발생 구문은 앞에 선언한 논리형 변수를 통해 편리하게 제어할 수 있는 if나 while을 사용해 구성한다.

```
for (int i = START; i <= END; i++) {  
    if (!isRandomAllocCheck) {  
        decision = (int) (Math.random() * range + MIN);  
        isRandomAllocCheck = true;  
    }  
}
```

- 2. 현재값이 선택된 난수의 배수에 해당하는지 판별하는 구문은 if(현재값 % 선택된 난수 == 0) 최초 난수 발생 후에 수행된다.

```
for (int i = START; i <= END; i++) {  
    if (!isRandomAllocCheck) {  
        decision = (int) (Math.random() * range + MIN);  
        isRandomAllocCheck = true;  
    }  
  
    if (i % decision == REMAIN) {  
        System.out.printf("%d의 배수 i = %d\n", decision, i);  
        isRandomAllocCheck = false;  
    }  
}
```

조건2에 부합할 때 수행할 동작 :

선택된 난수의 배수값인 현재값을 출력시키고, sum에 현재값을 더한다.

```
if (i % decision == REMAIN) {  
    System.out.printf("%d의 배수 i = %d\n", decision, i);  
    isRandomAllocCheck = false;  
  
    sum += i;  
}
```

난수를 발생시키는 while문이 먼저 수행될지 / 현재값이 선택된 난수의 배수에 해당하는지 판별하는 if문이 먼저 수행될지는 난수가 발생했을 때와 현재값이 선택된 난수의 배수에 해당할 때 논리형 변수의 값을 바꿈으로써 결정할 수 있다.

```
isRandomAllocCheck = true;
```

반복이 종료되면 printf를 이용해 출력된 값들의 합인 sum의 값을 보여준다.

```
System.out.println("현재까지 나타난 숫자들의 합 = " + sum);
```

- 문제10. 1 ~ 100까지의 숫자를 순회한다.

9번과 유사하게 2 ~ 10을 가지고 작업을 진행한다.

다만 이번에는 배수를 찾는게 아니라 랜덤한 숫자가 나온만큼만 이동하고 이동했을때 나온 숫자들의 합을 계산하도록 만들어보자!

- 구현 전략

1~100까지의 숫자를 순회시키기 위해 for문을 사용한다.

```
final int START = 1;
final int END = 100;
```

이동한 값들의 합을 저장시키기 위한 변수인 sum을 선언한다.

```
int sum = 0;
```

현재 위치값에서 얼마나 이동할지는 매 루프마다 수행되는 난수 발생을 통해 결정한다.

2~10 사이의 난수를 발생시키기 위해 Math.random()을 이용한다

-> decision = (int)(Math.random() * (10-2 +1) + 2))

```
final int MAX = 10;
final int MIN = 2;

int range = MAX - MIN + 1;
int decision = 0;
decision = (int) (Math.random() * range + MIN);
```

- for문 내부 동작

난수를 발생시키고 이동할 값과 현재 위치값을 출력한다. sum에 현재 위치값을 더한다.

```
System.out.println("-----");
System.out.printf("현재 위치 = %d\n", i);
decision = (int) (Math.random() * range + MIN);
System.out.printf("뽑은 난수 = %d\n", decision);
//System.out.printf("뽑은 난수 = %d, 현재 위치 = %d\n", decision, i);

sum += i;
System.out.println("현재까지 sum = " + sum);
System.out.println("-----");
```

- 위치값이 100을 넘지 않게 하기 위한 for문 구성

for (int i = START; i <= END; i += decision)

for문 내부 동작 후 수행되는 증감식: 현재 위치와 난수의 값을 더해준다.


증감식 이후 수행되는 조건식: 현재 위치와 난수의 값이 더해진 값을 끝점인 100과 비교해준다.

```
for (int i = START; i <= END; i += decision) {
```

- for문 동작방식

- : 1. 초기값을 가지고 2. for문 내부동작을 수행한다.
3. 증감식으로 이동해 현재 위치값에서 선택된 난수의 범위만큼 이동한다.
4. 이전 위치와 난수의 값이 더해진 i값이 100보다 작거나 같다면 5. 다음 루프를 실행하고,
100보다 크다면 반복이 종료된다.

```
for (int 1i = START; 4i <= END; 3i += decision) {  
    System.out.println("-----");  
    System.out.printf("현재 위치 = %d\n", 2, 5i);  
    decision = (int) (Math.random() * range + MIN);  
    System.out.printf("뽑은 난수 = %d\n", decision);  
    //System.out.printf("뽑은 난수 = %d, 현재 위치 = %d\n", decision, i);  
    /*  
    if (i == DEATH) {  
        i = START;  
    }  
    */  
    sum += i;  
    System.out.println("현재까지 sum = " + sum);  
    System.out.println("-----");  
}
```



반복문이 다 끝나면 최종 sum의 값을 출력한다.

```
System.out.println("현재까지 나타난 숫자들의 합 = " + sum);
```