

## 04. 참조자료형



강동기

# 참조자료형

---

- 참조자료형이란??
  - Reference type
  - 실제 데이터 값을 저장하지 않고 데이터가 저장된 메모리의 주소(참조값, reference)를 저장하는 자료형
  - 배열(Array), 클래스(Class), 인터페이스(Interface) 등

# 배열

---

- 동일한 자료형을 묶어 저장하는 참조자료형
- 생성할 때 크기를 지정해야 함
- 한번 크기를 지정하면 변경 불가함

# 배열

- 배열 생성하기
  - 배열을 나타내는 대괄호([])를 사용하여 생성
  - 대괄호는 자료형 뒤 (일반적임) 혹은 변수명 뒤에 위치할 수 있음

## 1차원 배열의 선언 방법

자료형[] 변수명

자료형 변수명[]

예 `int[] a;`  
`double[] b;`  
`String[] c;`

`int a[];`  
`double b[];`  
`String c[];`

# 배열

- 힙 메모리에 배열 객체 생성하기
  - 모든 참조 자료형의 실제 데이터(객체)는 힙 메모리에 생성됨
  - 힙 메모리에 객체를 생성하기 위해 "new" 키워드를 사용함
  - 배열의 객체를 생성할때는 길이를 지정해야 함

## 배열의 객체 생성

new 자료형[배열의 길이]

예    new int[3];  
      new String[5];

# 배열

- 배열 자료형 변수에 객체 대입하기
  - 변수 선언과 값의 대입을 한번에 작성
  - 변수 선언과 값의 대입을 나눠서 작성

## 배열 참조 자료형 변수에 생성한 객체 대입

```
자료형[] 변수명 = new 자료형[배열의 길이];
```

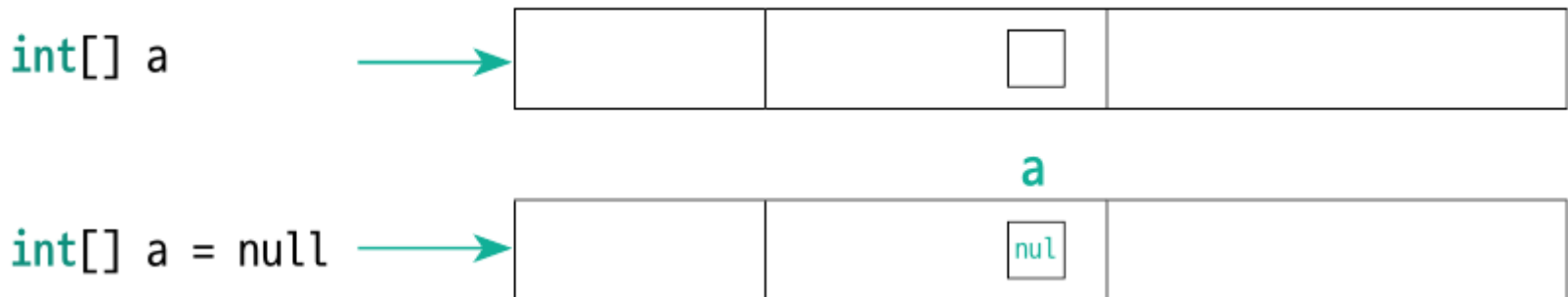
예 `int[] a = new int[3];`

```
자료형[] 변수명;  
변수명 = new 자료형[배열의 길이];
```

예 `int[] a;`  
`a = new int[3];`

# 배열

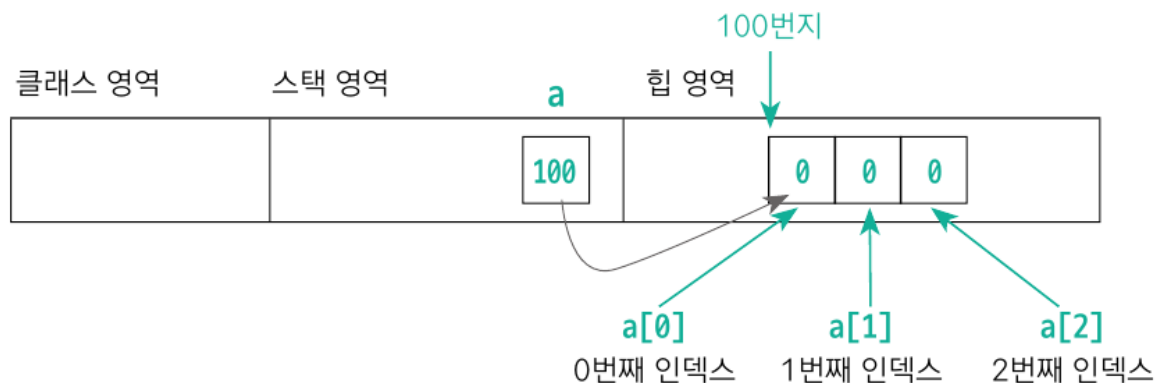
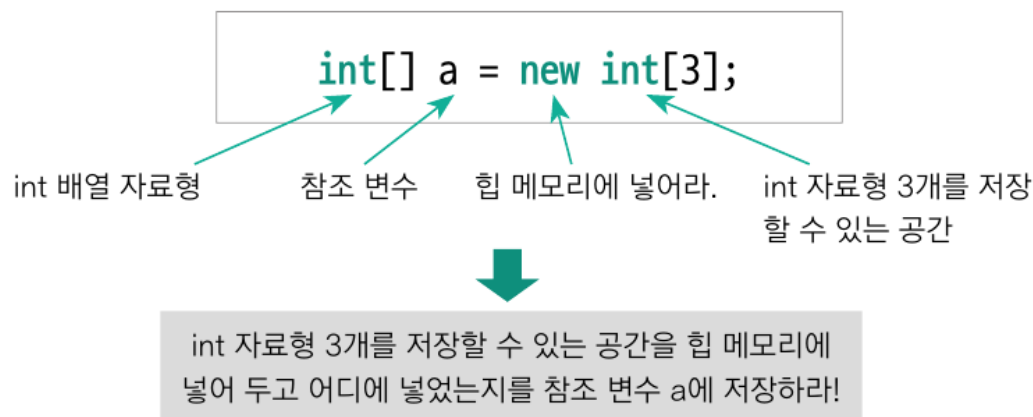
- 배열 생성과 공간
  - 배열 선언 시 스택 메모리에 변수의 공간만 생성됨
  - 배열의 실제 데이터인 객체 생성 전까지, 공간은 비어 있음  
(선언만 하는 경우 컴파일 에러 발생)
- null(널)
  - 스택 메모리에 위치한 참조 자료형 변수의 빈 공간을 초기화시 사용 가능함 (null 값을 넣은 경우 컴파일 에러가 발생하지 않음)
  - 힙 메모리의 특정 위치(번지)를 가리키고 있지 않다는 의미를 내포함  
(연결된 실제 데이터가 없음)



# 배열

- 배열과 메모리

- `int[] a = new int[3];`
- 스택과 달리 힙 메모리에서는 값을 주지 않는 경우 강제 초기화가 진행됨





# 배열

- 배열 선언 후 값 대입하기
  - 배열의 값을 저장하는 각 공간은 인덱스(index) 번호를 가짐
  - 인덱스는 0부터 시작하여 1씩 증가함  
(공간이 3개라면 인덱스는 0,1,2가 됨)

## 배열 객체에 값 대입하기

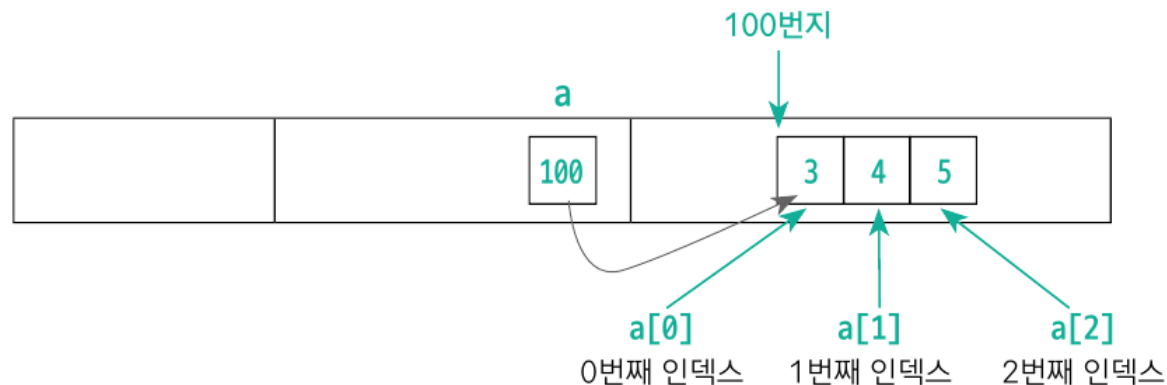
참조 변수명[인덱스] = 값;

예 `int[] a = new int[3];`

`a[0] = 3;`

`a[1] = 4;`

`a[2] = 5;`



# 배열

- 배열 선언과 동시에 값 대입하기
  - new 키워드 없이 배열 객체 생성과 함께 값 대입가능
  - 중괄호를 이용하여 초기화를 수행

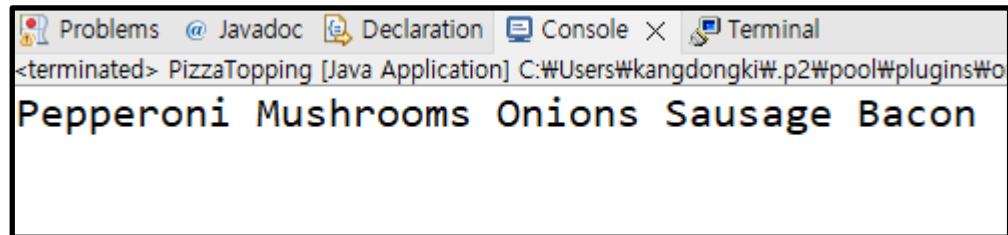
## 대입할 값만 입력하기

자료형[] 참조 변수명 = {값, 값, ..., 값};

예 `int[] a = {3, 4, 5};`

- PizzaTopping.java

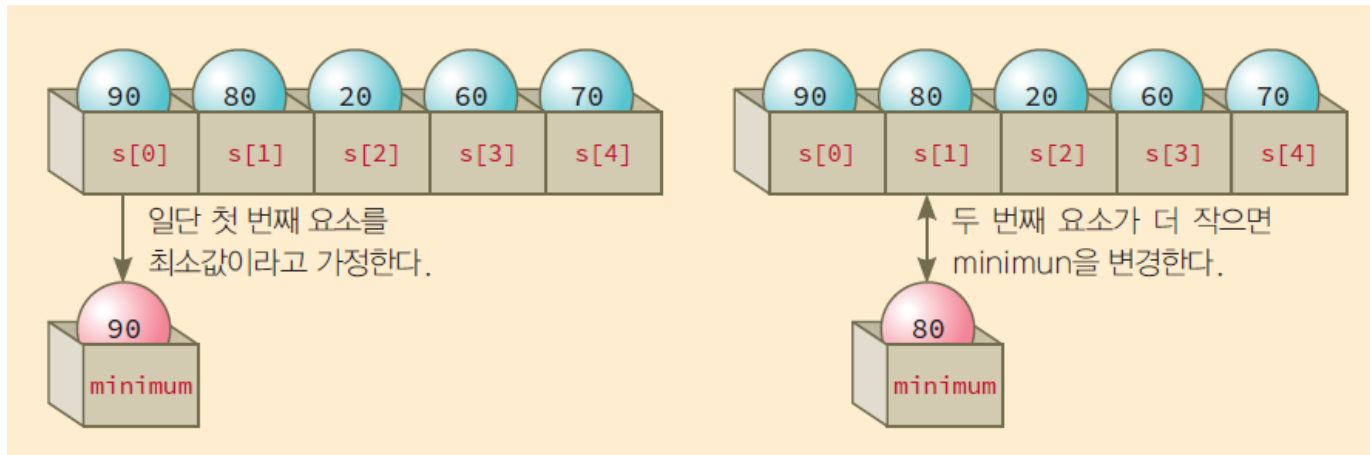
```
public class PizzaTopping {  
    public static void main(String[] args) {  
        String[] toppings =  
            { "Pepperoni", "Mushrooms", "Onions", "Sausage", "Bacon" };  
  
        for(int i=0; i<toppings.length; i++) {  
            System.out.print(toppings[i] + " ");  
        }  
    }  
}
```



The screenshot shows a Java IDE window with a terminal tab. The terminal output is: `<terminated> PizzaTopping [Java Application] C:\Users\kangdongki\p2\pool\plugins\Wo` followed by the toppings: `Pepperoni Mushrooms Onions Sausage Bacon`.

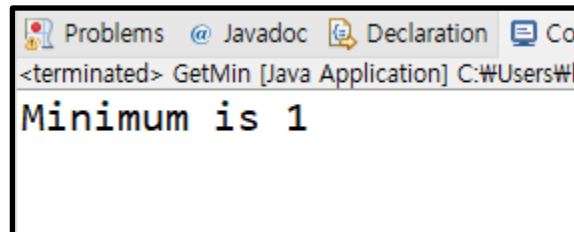
# 배열

- GetMin.java
  - 최소값을 구하는 예제
  - 배열의 첫번째 요소를 최소값이라고 가정
  - 순서대로 배열 요소 값을 검사하면서 최소값 변경



- GetMin.java

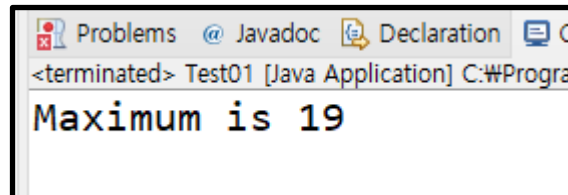
```
public class GetMin {  
    public static void main(String[] args) {  
        int s[] = { 12, 3, 19, 6, 18, 8, 12, 4, 1, 19 };  
        int minimum;  
  
        minimum = s[0];  
  
        for (int i = 1; i < s.length; i++) {  
            if (s[i] < minimum)  
                minimum = s[i];  
        }  
        System.out.print("Minimum is " + minimum);  
    }  
}
```



# 연습문제 1

- GetMax.java
  - 최소값 대신 최대값을 찾아보고 싶다.
  - 코드를 수정해보자.

```
public class GetMax {  
    public static void main(String[] args) {  
        int s[] = { 12, 3, 19, 6, 18, 8, 12, 4, 1, 19 };  
        int maximum;  
  
        ?????  
  
        System.out.print("Maximum is " + maximum);  
    }  
}
```



# 배열과 참조자료형

---

- 데이터 복사 시 기본 자료형 및 참조자료형 변수의 차이점은?
- 기본 자료형 변수를 복사하면 데이터값 자체가 복사됨
- 복사된 데이터값을 변경해도 원본값은 바뀌지 않음
- 참조자료형 변수를 복사하면 데이터가 저장된 주소값이 복사됨
- 데이터값을 변경하면 다른 참조 변수의 데이터도 변하게 됨

# 배열과 참조자료형

- 기본 자료형 변수 복사

## 기본 자료형 변수 복사

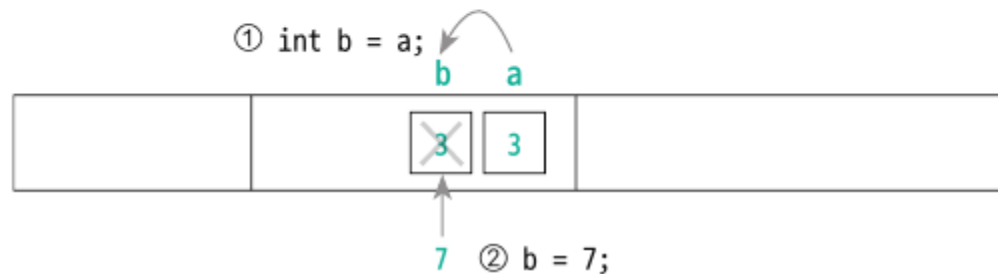
```
int a = 3;
```

```
int b = a; ①
```

```
b = 7; ②
```

```
System.out.println(a); // 3
```

```
System.out.println(b); // 7
```





# 배열과 참조자료형

- 참조 자료형 변수 복사

## 참조 자료형 변수 복사

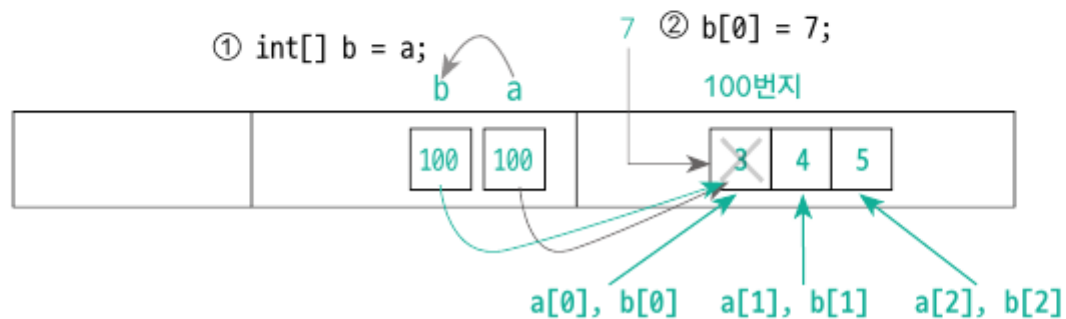
```
int[] a = {3, 4, 5};
```

```
int[] b = a; ①
```

```
b[0] = 7; ②
```

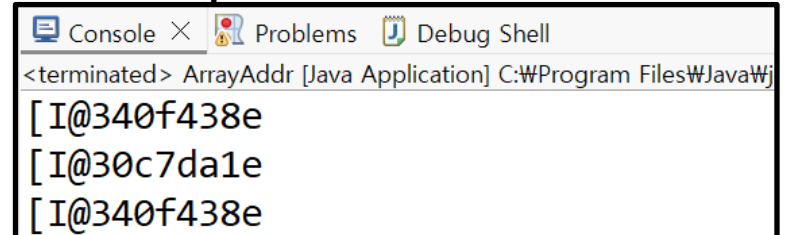
```
System.out.println(a[0]); // 7
```

```
System.out.println(b[0]); // 7
```



- ArrayAddr.java

```
public class ArrayAddr {  
    public static void main(String[] args) {  
        int[] a = new int[3];  
        a[0] = 3;  
        a[1] = 4;  
        a[2] = 5;  
  
        int[] b = new int[3];  
        b[0] = 3;  
        b[1] = 4;  
        b[2] = 5;  
  
        int[] c = a;  
  
        System.out.println(a);  
        System.out.println(b);  
        System.out.println(c);  
    }  
}
```



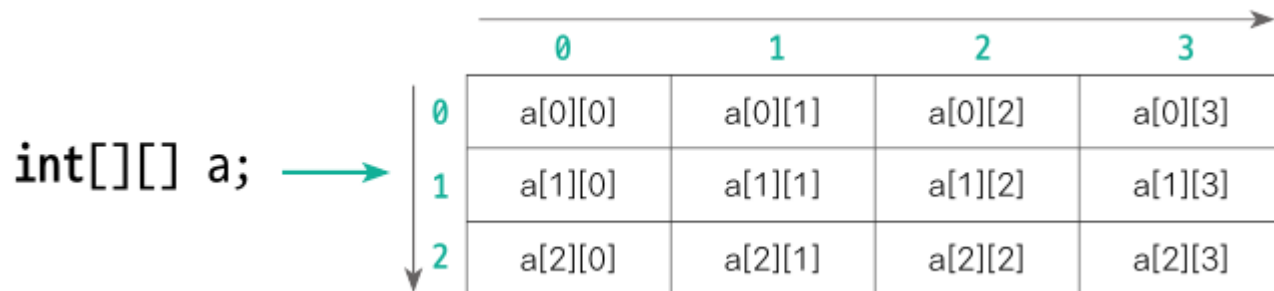
Console × Problems Debug Shell  
<terminated> ArrayAddr [Java Application] C:\Program Files\Java\j  
[I@340f438e  
[I@30c7da1e  
[I@340f438e

# 2차원 배열

- 2차원 배열 (2D Array)
  - 가로, 세로 방향의 2차원으로 데이터를 저장하는 배열
  - 대괄호 2개를 이용하여 표시함

## 2차원 배열의 선언 방법

	자료형[][] 변수명	자료형 변수명[][]	자료형[] 변수명[]
예	<code>int[][] a;</code> <code>double[][] b;</code> <code>String[][] c;</code>	<code>int a[][];</code> <code>double b[][];</code> <code>String c[][];</code>	<code>int[] a[];</code> <code>double[] b[];</code> <code>String[] c[];</code>



## 2차원 배열

- 배열 객체 선언 후 값 대입하기 (new 키워드를 사용함)

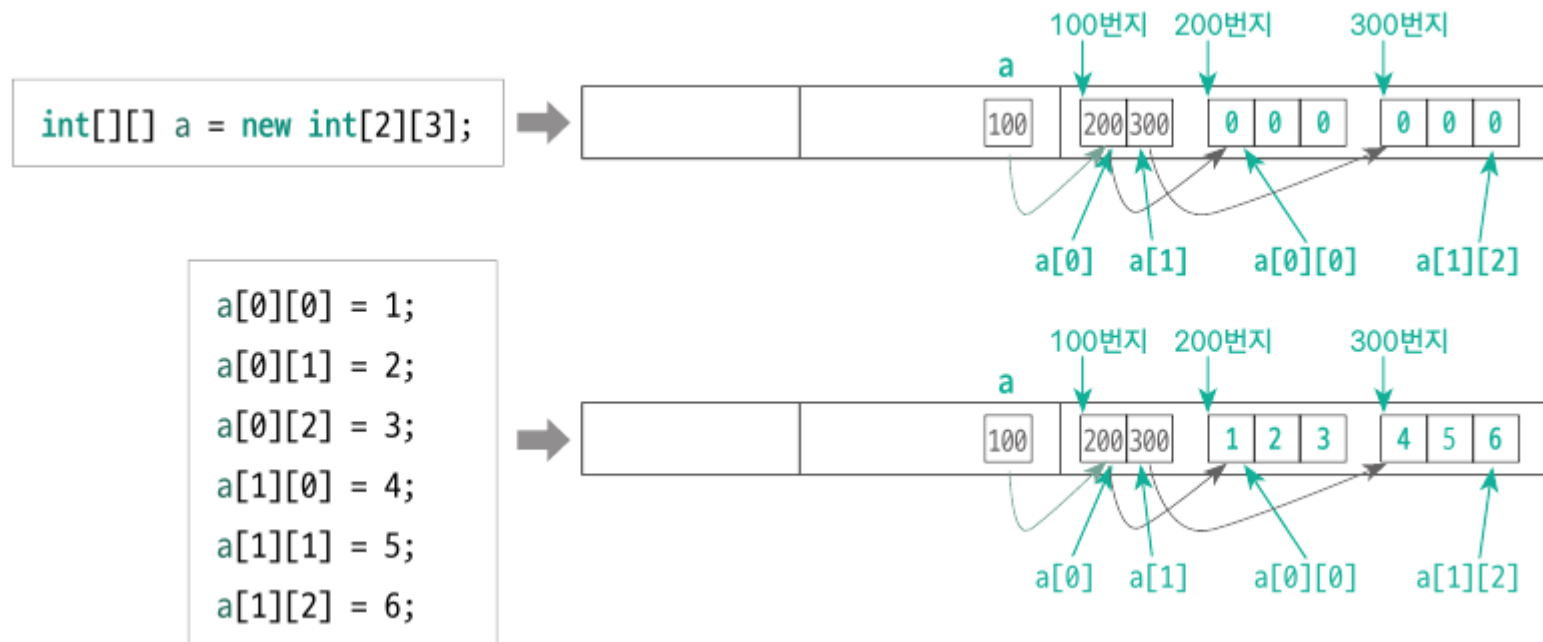
### 2차원 정방 행렬 배열 객체를 생성하고 값 대입하기

```
자료형[][] 참조 변수명 = new 자료형[행의 길이] [열의 길이];  
참조 변수명[0][0] = 값;  
참조 변수명[0][1] = 값;  
...  
참조 변수명 [행의 길이 -1] [열의 길이 -1] = 값;
```

```
예  int[][] a = new int[2][3];  
    a[0][0] = 1; a[0][1] = 2; a[0][2] = 3;  
    a[1][0] = 4; a[1][1] = 5; a[1][2] = 6;
```

## 2차원 배열

- 배열 객체 선언 후 값 대입하기 메모리 구조



## 2차원 배열

- 배열 객체 선언과 동시에 값 대입하기 (new 키워드를 사용하지 않음)

### 대입할 값만 입력하기

```
자료형[][] 참조 변수명 = {{값, 값, ..., 값}, ..., {값, 값, ..., 값}};
```

0번째 행 데이터

마지막 행 데이터

예 `int[][] a = {{1, 2, 3}, {4, 5, 6}};`

배열의 선언과 객체의 대입을 분리해 표현 불가능

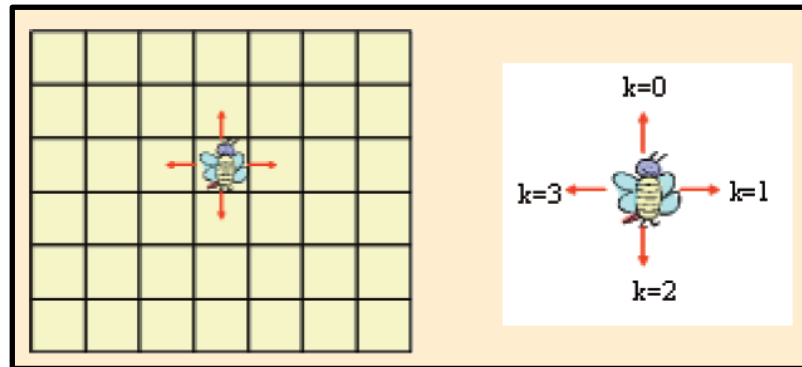
```
int[][] a = {{1, 2, 3}, {4, 5, 6}}; // (○)
```

```
int[][] b;
```

```
b = {{1, 2, 3}, {4, 5, 6}}; // (X)
```

## 2차원 배열

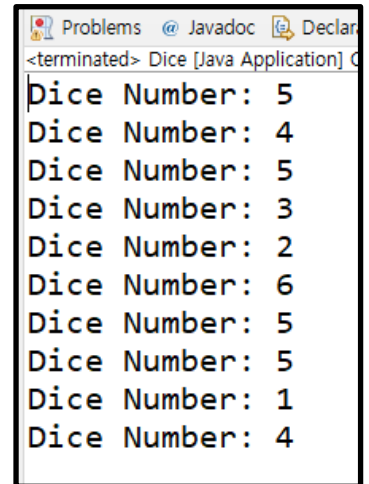
- RandomWalk.java
  - 11\*11 타일로 구성된 방안에 딱정벌레 1마리가 있다.
  - 딱정벌레는 임의의 위치를 선택하여 돌아다닌다.
  - 총 10번 이동할 수 있다고 가정하자.
  - 딱정벌레가 지나간 경로를 표시해보자.



## 2차원 배열

- Dice.java (RandomWalk.java 구현을 위한 부예제)
  - 주사위 숫자 1~6 중 하나를 랜덤하게 선택
- Math.random() 메소드
  - 0.0 이상 1.0 미만의 double 형 실수를 랜덤하게 출력해주는 메소드

```
public class Dice {  
    public static void main(String[] args) {  
        int diceNum;  
        for(int i=0; i<10; i++) {  
            diceNum = (int) (Math.random() * 6) + 1;  
            System.out.println("Dice Number: " + diceNum);  
        }  
    }  
}
```



```
Problems @ Javadoc Decla  
<terminated> Dice [Java Application] C  
Dice Number: 5  
Dice Number: 4  
Dice Number: 5  
Dice Number: 3  
Dice Number: 2  
Dice Number: 6  
Dice Number: 5  
Dice Number: 5  
Dice Number: 1  
Dice Number: 4
```



## 2차원 배열

- RandomWalk.java (1/2)

```
import java.util.Scanner;

public class RandomWalk {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int x = 5, y = 5;
        boolean[][] tile = new boolean[11][11];
        int steps;
        tile[y][x] = true;

        for (steps = 0; steps < 10; steps++) {
            int direction = (int) (Math.random() * 4);
            if (direction == 0 && x > 0) // go left
                x--;
            else if (direction == 1 && x < 10) // go right
                x++;
            else if (direction == 2 && y > 0) // go up
                y--;
            else if (y < 10) // go down
                y++;
            tile[y][x] = true;
        }
    }
}
```

## 2차원 배열

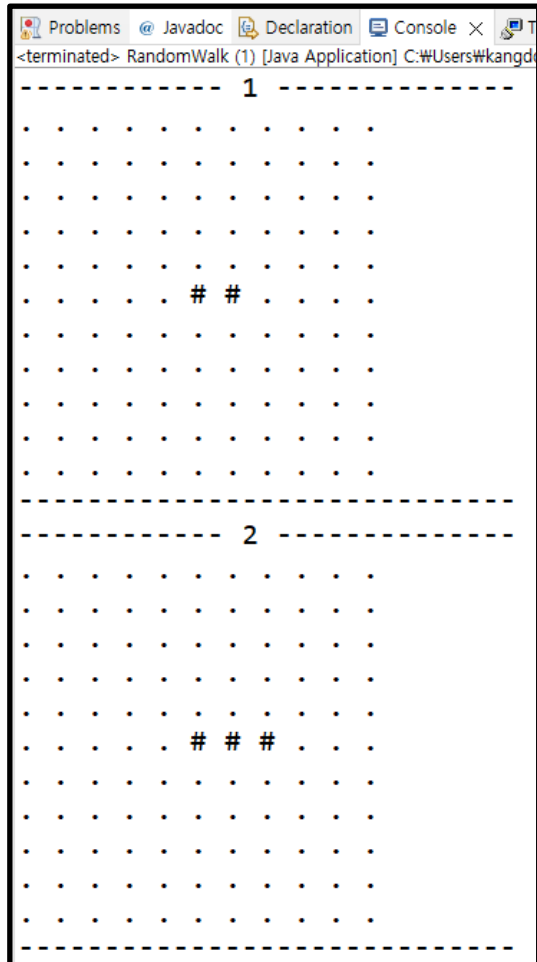
- RandomWalk.java (2/2)

```
System.out.print("Enter.");
input.nextLine();

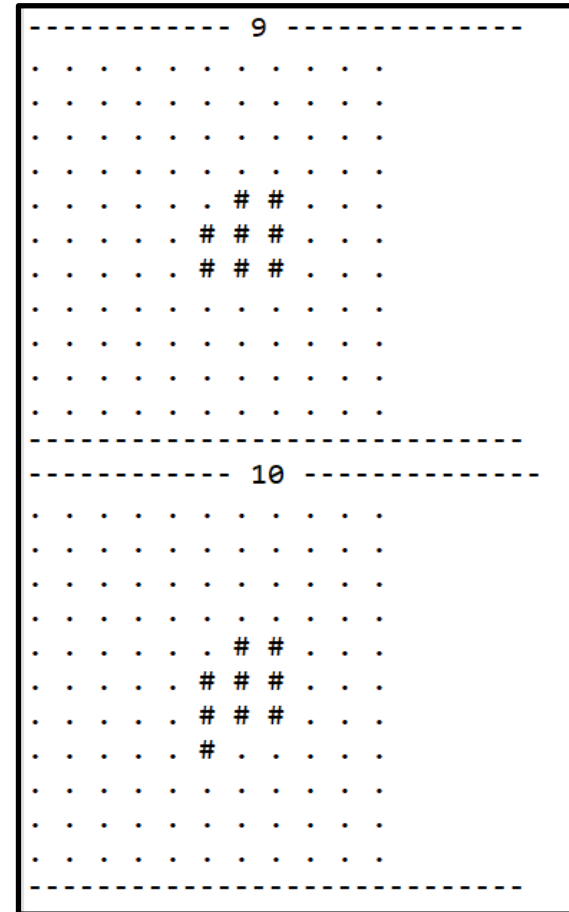
System.out.println("----- " + (steps + 1) + " -----");
for (int i = 0; i < 11; i++) {
    for (int j = 0; j < 11; j++) {
        if (tile[i][j] == true)
            System.out.print("# ");
        else
            System.out.print(". ");
    }
    System.out.println();
}
System.out.println("-----");
}
```

## 2차원 배열

- RandomWalk.java 실행 결과



• • • •



# 연습문제 2

- RandomWalk.java 를 수정해보자
  - 총 10스텝을 움직였으니, 첫 시작점을 포함해서 # 기호는 11개가 나오는게 자연스럽다.
  - 그러나 이전에 방문했던 위치를 다시 방문하는 경우가 발생하므로 # 기호의 개수가 11개보다 적게 나올 수 있다 (아래결과에서는 9개)
  - 한번 이동했던 위치는 다시 이동하지 않도록 하고싶다.
  - 코드를 어떻게 바꾸면 될까??

```
import java.util.Scanner;

public class RandomWalk {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int x = 5, y = 5;
        boolean[][] tile = new boolean[11][11];
        int steps;
        tile[x][y] = true;

        for (steps = 0; steps < 10; steps++) {
            int direction = (int) (Math.random() * 4);
            if (direction == 0 && x > 0) // go left
                x--;
            else if (direction == 1 && x < 10) // go right
                x++;
            else if (direction == 2 && y > 0) // go up
                y--;
            else if (y < 10) // go down
                y++;
            tile[y][x] = true;
        }
    }
}
```

```
System.out.print("Enter.");
input.nextLine();
```

```
System.out.println("-----" + " + ");
for (int i = 0; i < 11; i++) {
    for (int j = 0; j < 11; j++) {
        if (tile[i][j] == true)
            System.out.print("# ");
        else
            System.out.print(". ");
    }
    System.out.println();
}
System.out.println("-----");
```

```
----- 9 -----
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
----- 10 -----
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
-----
```

# String

---

- 문자열을 저장하기 위해 String 참조 자료형 사용
- String 객체 생성 방법 2가지
  - new 키워드를 사용하여 객체를 생성함
  - new 키워드 없이 큰따옴표와 문자열 리터럴만으로 객체를 생성함

# String

- new 키워드를 사용하여 String 객체 생성

## String 클래스의 객체 생성 방법 ①

String 참조 변수명 = new String("문자열")

String 클래스의 생성자

저장할 문자열

예 String str = new String("안녕");

# String

- new 키워드 없이 String 객체 생성

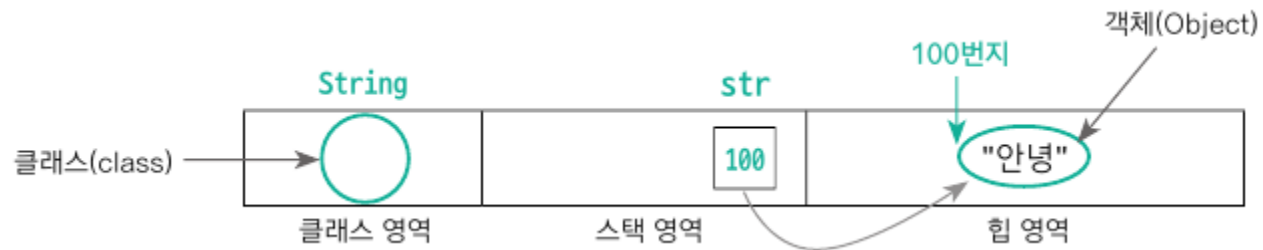
String 클래스의 객체 생성 방법 ②

```
String 참조 변수명 = "문자열"
```

예 `String str = "안녕";`

# String

- String 객체와 메모리 구조





# String

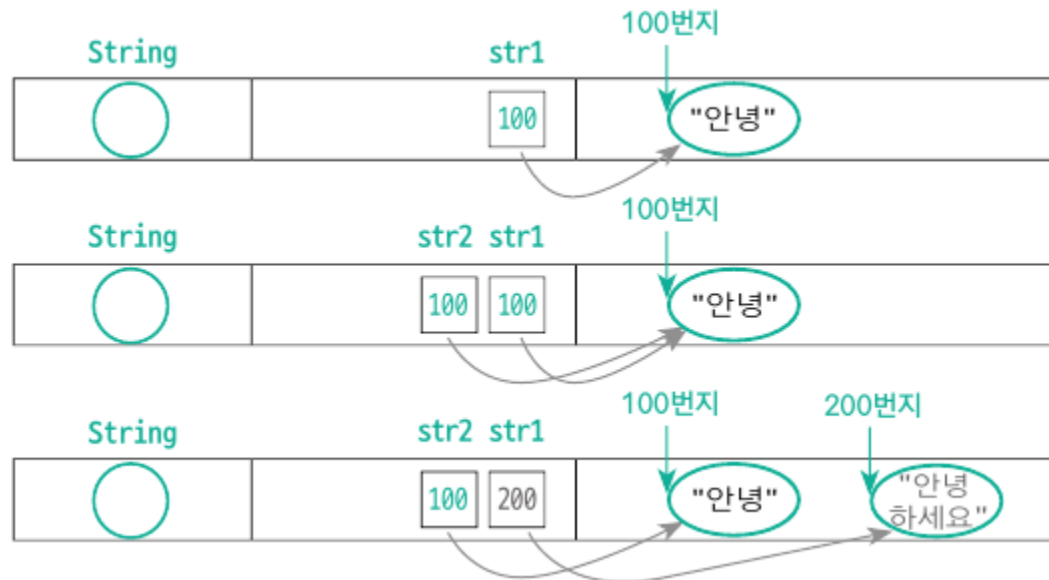
---

- String 의 2가지 특징
- 한번 정의한 문자열을 변경할 수 없음
  - 문자열 변경을 시도하는 경우 JVM은 기존의 문자열을 수정하지 않음
  - 새로운 문자열을 포함하는 객체 생성 후, 더 이상 (쓰지 않는) 객체는 폐기함
- 리터럴을 바로 입력한 데이터는 문자열이 같을 때 객체를 공유함
  - 메모리의 효율적인 사용을 위함

# String

- String 객체 값을 변경 시 -> 새로운 객체를 생성

```
String str1 = new String("안녕");  
String str2 = str1;  
str1 = "안녕하세요";  
System.out.println(str1);    // 안녕하세요  
System.out.println(str2);    // 안녕
```



# String

- 리터럴을 바로 입력한 데이터는 문자열이 같을 때 하나의 객체를 공유
  - 메모리 효율성을 증가시키기 위함

```
String str1 = new String("안녕");  
String str2 = "안녕";  
String str3 = "안녕";  
String str4 = new String("안녕");
```



# String

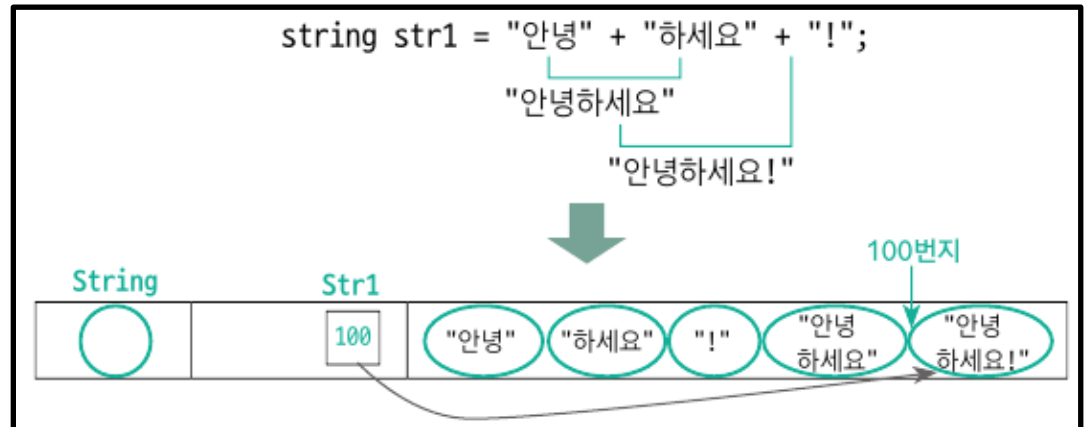
---

- String 객체의 '+' 연산
  - 문자열 + 문자열 => 문자열
  - 문자열 + 기본 자료형 => 기본 자료형이 문자열로 변환 => 문자열
  - 기본 자료형 + 문자열 => 기본 자료형이 문자열로 변환 => 문자열

# String

- 문자열 + 문자열
- 아래의 코드를 실행하는 경우 str1, str2를 위한 객체가 각각 5개씩 생성됨
  - 객체 3개 생성: "안녕", "하세요", "!"
  - 객체 1개 생성: "안녕" (폐기), "하세요" (폐기) -> "안녕하세요"
  - 객체 1개 생성: "안녕하세요" (폐기), "!" (폐기) -> "안녕하세요!"

```
String str1 = "안녕" + "하세요" + "!";  
System.out.println(str1); // 안녕하세요!  
  
String str2 = "반갑";  
str2 += "습니다";  
str2 += "!";  
System.out.println(str2); // 반갑습니다!  
System.out.println();
```



# String

- 문자열 + 기본 자료형 / 기본 자료형 + 문자열
- 1 + "안녕"
  - 1 인식 -> 1 문자열로 변환 -> "1안녕"
- 1 + " 안녕" + 2
  - 1 인식 -> 1 문자열 변환 -> "1안녕" -> 2 문자열 변환 -> "1안녕2"
- "안녕" + 1 + 2
  - "안녕" 인식 -> 1 문자열 변환 -> "안녕1" -> 2 문자열 변환 -> "안녕12"
- 1 + 2 + "안녕"
  - 1 + 2 = 3 수행 -> 3 문자열 변환 -> "3안녕"

```
System.out.println(1 + "안녕");      // 1안녕
System.out.println(1 + "안녕" + 2);  // 1안녕2
System.out.println("안녕" + 1 + 2);  // 안녕12
System.out.println(1 + 2 + "안녕");  // 3안녕
```

# String

- String 클래스의 주요 메서드


구분	리턴 타입	메서드	설명
문자열 길이	int	length()	문자열의 길이
문자열 검색	char	charAt(int index)	인덱스 위치에서의 문자
	int	indexOf(int ch) indexOf(int ch, int fromIndex) indexOf(String str) indexOf(String str, int fromIndex)	문자열에 포함된 문자 또는 문자열의 위치를 앞에서부터 검색했을 때 일치하는 인덱스 값 (fromIndex는 검색 시작 위치)
	int	lastIndexOf(int ch) lastIndexOf(int ch, int fromIndex) lastIndexOf(String str) lastIndexOf(String str, int fromIndex)	문자열에 포함된 문자 또는 문자열의 위치를 뒤에서부터 검색했을 때 일치하는 인덱스값 (fromIndex는 검색 시작 위치)
문자열 변환 및 검색	boolean	String.valueOf(boolean b)	boolean, char, int, long, float, double 값을 문자열로 변환하기 위한 정적 메서드
	char	String.valueOf(char c)	
문자열 변환 및 검색	int	String.valueOf(int i)	boolean, char, int, long, float, double 값을 문자열로 변환하기 위한 정적 메서드
	long	String.valueOf(long l)	
	float	String.valueOf(float f)	
	double	String.valueOf(double d)	
	double	concat(String str)	
문자열 배열 변환	byte[]	getBytes() getBytes(Charset charset)	문자열을 byte[]로 변환(변환할 때 문자 셋 (charset) 지정 가능)
	char[]	toCharArray()	문자열을 char[]로 변환

# String

- 문자열 -> 숫자 변환
- 래퍼 클래스 (Wrapper Class) 사용
  - 예를 들어 문자열 "123"을 정수형 변수로 바꾸려면 Integer 클래스 사용

기초 자료형	래퍼 클래스
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean
void	Void

래퍼 클래스는 기초 자료형을 클래스로 만들고 싶은 경우에 사용하면 됩니다. 문자열을 수치값으로 변환해주는 메소드도 가지고 있습니다.





# String

- 문자열 -> 숫자 변환
  - 문자열을 기초 자료형으로 변환하기 위해 각 래퍼 클래스의 `parseXXX()` 메서드를 활용

```
int i = Integer.parseInt("123");  
double d = Double.parseDouble("3.141592");
```

# String

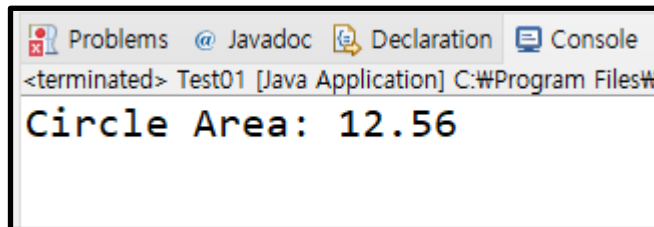
- GetCircleArea.java
  - 원의 반지름을 입력받아서 원의 넓이를 구하는 프로그램 (파이값은 3.14로 가정)
  - 입력값을 실수로 변환한 다음 계산을 수행

```
import java.util.Scanner;

public class GetCircleArea {
    public static void main(String[] args) {
        String radiusStr = "2";

        double radiusDbl = Double.parseDouble(radiusStr);
        double circleArea = radiusDbl * radiusDbl * 3.14;

        System.out.println("Circle Area: " + circleArea);
    }
}
```



## 연습문제 3

- GetCircleArea.java 를 수정해보자
  - 코드로 값을 고정하지 말고 프로그램을 실행한 후에 반지름 값을 입력하게 하자
  - 코드를 수정해보자

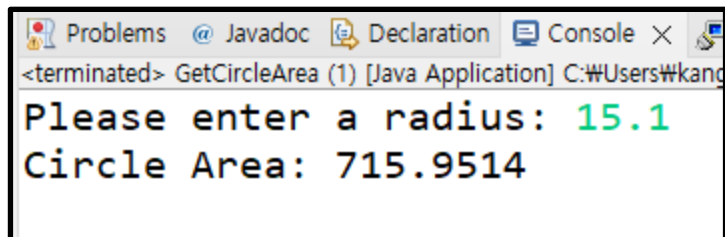
```
import java.util.Scanner;

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

        ??????

        double circleArea = radiusDb1 * radiusDb1 * 3.14;

        System.out.println("Circle Area: " + circleArea);
    }
}
```



```
<terminated> GetCircleArea (1) [Java Application] C:\Users\kang
Please enter a radius: 15.1
Circle Area: 715.9514
```

**감사합니다! XD**

