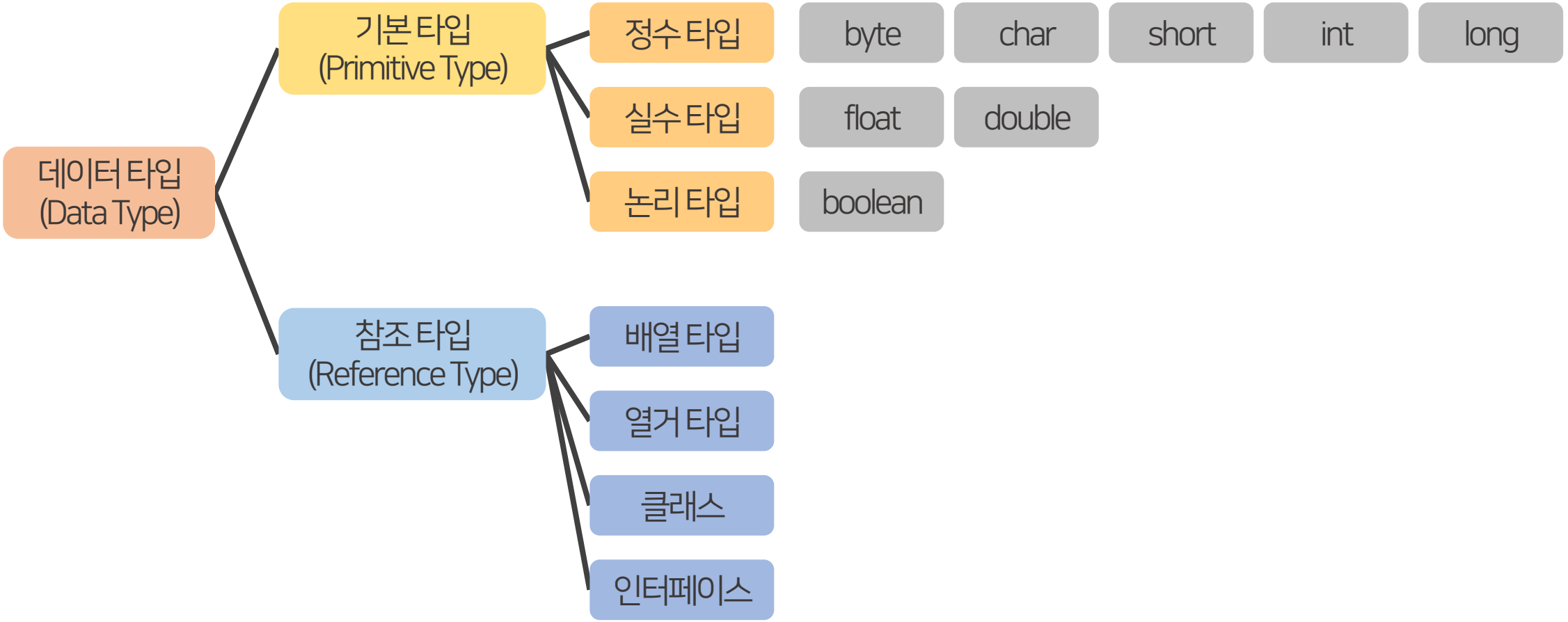
Java

참조 타입

- Java의 데이터 타입은 크게 기본 타입(Primitive Type)과 참조 타입(Reference Type)으로 분류된다.



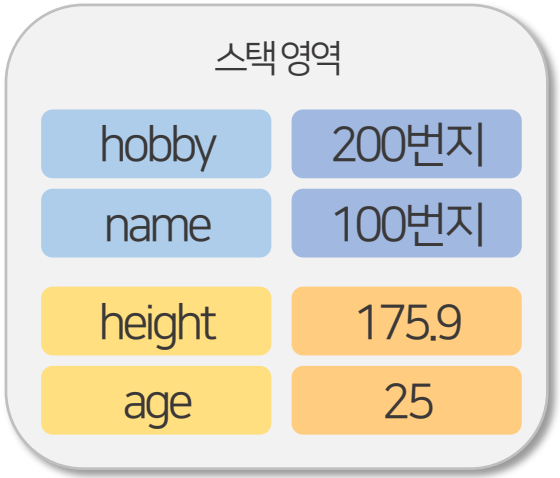
참조 타입

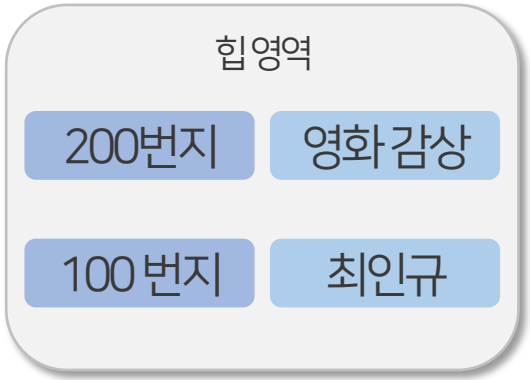
- 참조 타입이란 객체의 메모리 주소를 참조하는 타입으로 배열, 열거, 클래스, 인터페이스 타입이 있다.
- 기본 타입으로 선언된 변수는 값 자체를 저장하고 있지만,
  참조 타입으로 선언된 변수는 객체가 생성된 메모리 주소를 저장한다.

- 변수들은 모두 스택(Stack)이라는 메모리 영역에 생성된다.
- 기본 타입 변수는 스택 영역에 직접 값을 저장하는 반면, 참조 타입 변수는 스택 영역에 힙 메모리 영역의 주소를 저장한다.

```
int age = 25;
double height = 175.9;

String name = "최인규";
String hobby = "영화 감상";
```





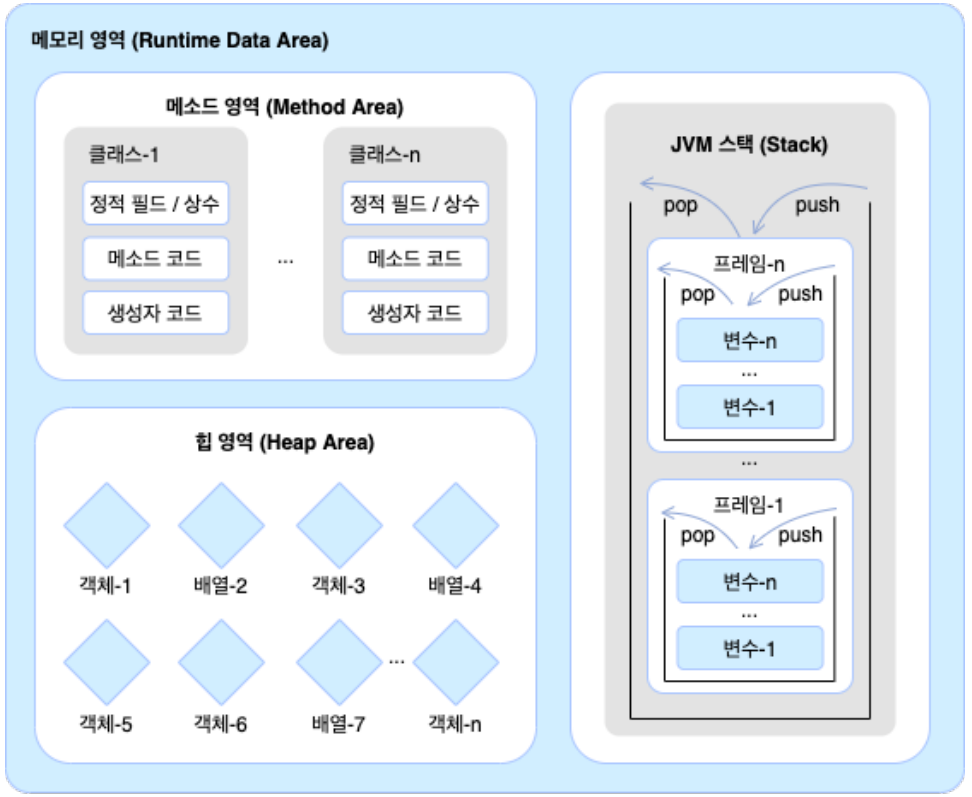
참조 타입

- Java에서 JVM이 구동되면 JVM은 운영체제에서 할당받은 메모리 영역을 다음과 같이 구분해서 사용한다.

  - 메소드 영역 :
    바이트코드 파일 내용이 저장되는 영역으로
    클래스 별로 상수, 정적 필드, 메소드 코드, 생성자 코드 등이 저장된다.

  - 힙 영역 :
    객체가 생성되는 영역으로
    객체의 메모리 주소는 메소드 영역과 스택 영역에서 참조된다.

  - 스택 영역 :
    메소드를 호출할 때마다 생성되는 프레임이 저장되는 영역으로,
    프레임 내부의 변수 스택에 변수가 생성되고 제거된다.



참조 타입 변수의 동등 비교

- 일반적으로 기본 타입 변수의 경우에는 == 또는 != 연산자로 비교를 진행한다.
- 하지만 참조 타입 변수의 ==, != 연산자는 값을 비교하는 것이 아니라 메모리 주소를 비교하는 것이 된다.
- 번지가 같다면 동일한 객체를 참조하는 것이고, 다르다면 다른 객체를 참조하는 것이다.

```
int[] arr1;
int[] arr2;
int[] arr3;
int[] arr4 = {1, 2, 3};

arr1 = new int[] {1, 2, 3};
arr2 = new int[] {1, 2, 3};
arr3 = arr2;

System.out.println(arr1);
System.out.println(arr2);
System.out.println(arr3);
System.out.println(arr4);
```

TIP

배열은 나중에 자세히...

null과 NullPointerException

- 참조 타입 변수는 아직 메모리 주소를 저장하고 있지 않다는 의미로 null(널) 값을 가질 수 있다.
- null 값으로 초기화된 변수는 스택 영역에 생성되기 때문에 ==, != 연산자로 null 값 비교가 가능하다.

```
String var1 = "자바";
String var2 = null;

System.out.println(var1 == null);
System.out.println(var2 == null);
```

- 프로그램 실행 도중에 발생하는 오류를 예외(Exception)라고 부른다.
- 참조 타입 변수 사용 시, 가장 많이 발생하는 예외 중 하나는 NullPointerException이다.
- NullPointerException은 변수가 null인 상태에서 객체의 데이터나 메소드를 사용하려 할 때 발생한다.

```
System.out.println(var2.length());
```

null과 NullPointerException [실습]

```
int[] arr1 = {1, 2, 3};

for (int i = 0; i < 4; i++) {
    int j = arr1[i];
    System.out.println(j);
}
```

null과 NullPointerException

- 경우에 따라서는 참조 타입 변수에 일부러 null을 대입하기도 한다.
- 변수에 null을 대입하면 메모리 주소를 잃게 되므로 더 이상 객체를 사용할 수 없게 된다.

- 즉, 힙(heap) 메모리에는 있지만, 위치 정보를 알 수 없게 되었기 때문에 사용이 불가능해진다.
- Java는 이런 객체를 쓰레기(Garbage)로 취급하고, 가비지 콜렉터(Garbage Collector)를 실행시켜 자동으로 제거한다.

```
String name = "최인규";
name = null;

String hobby = "영화 감상";
hobby = "등산";
```

문자열(String) 타입

- Java의 문자열은 String 객체로 생성된다.
- Java는 문자열이 동일하다면 String 객체를 서로 공유하도록 설계되어 있다.

- String 변수에 문자열 리터럴을 대입하는 것이 일반적이지만, new 연산자로 직접 String 객체를 생성하고 대입할 수 있다.
- 이 경우에는 서로 다른 메모리 주소를 가지게 된다.
- 따라서 내부 문자열만을 비교하기 위해서는 equals() 메소드를 사용해야 하는 것이 바람직하다.

```
String name1 = "최인규";
String name2 = "최인규";
String name3 = new String("최인규");
String name4 = new String("최인규");

System.out.println(name1 == name2);
System.out.println(name1 == name3);
System.out.println(name1 == name4);
System.out.println(name2 == name3);
System.out.println(name2 == name4);
System.out.println(name3 == name4);
```

문자열(String) 타입 [실습]

```
String name1 = "최인규";
String name2 = "최인규";

if(name1 == name2) {
System.out.println("name1과 name2는 참조가 같음");
} else {
System.out.println("name1과 name2는 참조가 다름");
}

String name3 = new String("최인규");
String name4 = new String("최인규");

if(name3 == name4) {
System.out.println("name3과 name4는 참조가 같음");
} else {
System.out.println("name3과 name4는 참조가 다름");
}

if(name1.equals(name4)) {
System.out.println("name1과 name4는 문자열이 같음");
}
```

문자열(String) 타입 [실습]

- String 변수에 빈 문자열("")을 대입할 수도 있다.
- 빈 문자열도 String 객체로 생성되기 때문에 equals() 메소드를 사용해야 한다.

```
String name = "";

if(name.equals("")) {
    System.out.println("문자열 변수 hobby는 빈 문자열이다.");
}
```

문자열(String) 타입

- 문자열에서 특정 위치의 문자를 얻고자 한다면, charAt() 메소드를 이용할 수 있다.
- charAt() 메소드는 매개 변수로 주어진 인덱스의 문자를 반환한다.
- 여기서 인덱스란 0번째부터 문자열 길이 -1까지의 문자열 내부 위치 번호를 의미한다.

```
String subject = "자바 프로그래밍";
char var = subject.charAt(3);

System.out.println(var);
```

문자열(String) 타입 [실습]

- 주민등록번호에서 성별에 해당하는 7번째 문자를 읽고, 남자 또는 여자인지 출력하기

```
String idNumber = "991231-1001234";
char genderNumber = idNumber.charAt(7);

if (genderNumber % 2 == 0) {
System.out.println("여자입니다.");
} else {
System.out.println("남자입니다.");
}
```

문자열(String) 타입

- 문자열에서 문자의 개수를 알고 싶다면, length() 메소드를 사용한다.

```
String subject = "자바 프로그래밍";
int length = subject.length();

System.out.println(length);



String idNumber = "991231-1001234";
int idNumberLength = idNumber.length();

if (idNumberLength == 14) {
    System.out.println("주민등록번호 자릿수가 맞습니다");
} else {
    System.out.println("주민등록번호 자릿수가 틀립니다");
}
```

문자열(String) 타입 [실습]

```
Scanner sc = new Scanner(System.in);
System.out.println("입력한 메시지를 한 글자씩 출력합니다.");
System.out.print("메시지를 입력하세요 : ");
String msg = sc.nextLine();

for (int i = 0; i < msg.length(); i++) {
    char var = msg.charAt(i);
    System.out.println(var);
}
```

문자열(String) 타입

- 문자열에서 특정 문자열을 다른 문자열로 대체하고 싶다면 replace() 메소드를 사용한다.
- replace() 메소드는 기존 문자열은 그대로 두고, 대체할 새로운 문자열을 반환한다.

```
String oldStr = "자바 프로그래밍";
String newStr = oldStr.replace("자바", "Java");
System.out.println(oldStr);
System.out.println(newStr);
```

TIP

String 문자열은 변경이 불가능한 특성을 갖고 있다.

문자열(String) 타입

- 문자열에서 특정 위치의 문자열을 잘라내어 가져오고 싶다면, substring() 메소드를 사용한다.

```
String idNumber = "991231-1001234";

// 매개변수 두 개 [첫 번째 매개변수 시작 인덱스, 두 번째 매개변수 끝 인덱스]
String firstNumber = idNumber.substring(0, 6);

// 매개변수 한 개 [시작 인덱스]
String secondNumber = idNumber.substring(7);

System.out.println(firstNumber);
System.out.println(secondNumber);
```

문자열(String) 타입

- 문자열에서 특정 문자열의 위치를 찾고자 할 때는 indexOf() 메소드를 사용한다.
- indexOf() 메소드는 주어진 문자열이 시작되는 인덱스를 반환한다.

```
String lyrics = "떳다 떳다 비행기 날아라 날아라";

int flyIndex = lyrics.indexOf("날아라");

System.out.println(flyIndex + "번째 글자: " + lyrics.charAt(flyIndex));

int highIndex = lyrics.indexOf("높이");

if (highIndex == -1) {
    System.out.println("없는 글자입니다.");
}
```

문자열(String) 타입

- 문자열 내부에 특정 문자열의 위치가 아닌, 단순히 포함되어 있는지 여부를 확인하기 위해서는 contains() 메소드를 사용한다.
- contains() 메소드는 원하는 문자열이 포함되어 있으면 true, 그렇지 않으면 false를 반환한다.

```
Scanner sc = new Scanner(System.in);
System.out.print("책 제목을 입력하세요: ");
String title = sc.nextLine();

if (title.contains("자바") || title.contains("java") ||title.contains("Java")) {
    System.out.println("Java와 관련된 책이군요.");
} else {
    System.out.println("Java와 관련없는 책이군요.");
}
```

문자열(String) 타입

- 문자열을 특정 문자로 분리하기 위해서는 split() 메소드를 사용한다.
- split() 메소드는 문자열로 구성된 배열을 반환한다.

TIP

배열은 나중에 자세히...

```
String rainbow = "빨,주,노,초,파,남,보";
String[] rainbowArr = rainbow.split(",");

System.out.println(rainbowArr[0]);
System.out.println(rainbowArr[1]);
System.out.println(rainbowArr[2]);
System.out.println(rainbowArr[3]);
System.out.println(rainbowArr[4]);
System.out.println(rainbowArr[5]);
System.out.println(rainbowArr[6]);

for (int i = 0; i < rainbowArr.length; i++) {
    String color = rainbowArr[i];
    System.out.println(color);
}
```

배열 타입

- 변수는 하나의 값만 저장할 수 있기 때문에 저장할 값이 많아지면 그만큼 많은 변수가 필요하다.
- 예를 들어, 학생 30명의 성적을 변수에 담아 저장하고 평균값을 구한다고 가정해보자.

```
int score1 = 83;
int score2 = 90;
int score3 = 87;
...
int score30 = 75;
int sum = score1 + score2 + score3 + ... + score30;
double avg = sum / score1;
```

- 위와 같은 방식은 상당히 비효율적일 것이다.

배열 타입

- 따라서 많은 양의 값을 다루는 좀 더 효율적인 방법이 필요한데, 그래서 나온 것이 배열이라는 개념이다.
- 배열은 연속된 공간에 값을 나열시키고, 각 값에 인덱스(index)라는 위치번호를 부여해 놓은 자료구조이다.

- 인덱스는 대괄호([ ])와 함께 사용하여 각 항목의 값을 읽거나 저장하는데 사용된다.
- 예를 들어, score[0]은 배열의 가장 첫 번째 값을, score[1]는 배열의 첫 번째 값을 갖는다.

TIP

컴퓨터는 숫자를 0부터 센다는 점을 기억하자.

- 이렇게 성적을 배열에 저장하면 평균값은 배열의 인덱스를 이용해 쉽게 for 문으로 구할 수 있다.

```
int[] score = {83, 90, 87, ..., 75};
int sum = 0;

for (int i = 0; i < score.length; i++) {
     sum += score[i];
}

System.out.println(sum / score.length);
```

TIP

배열변수의 length 필드는 배열에 저장할 수 있는 항목 개수를 반환한다.

배열 타입

- 배열은 아래와 같은 특징을 갖는다.
  - 배열은 같은 타입의 값만 관리한다.
  - 배열의 길이는 늘리거나 줄일 수 없다.
  - 배열의 인덱스는 0부터 시작한다.

- int 배열은 int 타입의 값만 관리하고, String 배열은 문자열만 관리한다.
- 배열은 생성과 동시에 길이가 결정되기 때문에 한 번 결정된 배열의 길이는 늘리거나 줄일 수 없다.

배열 타입

- 배열 변수의 선언은 두 가지 형태로 가능하다.

```
int[] intArr1;
String[] strArr1;

int intArr2[];
String strArr2[];
```

- 일반적으로는 타입과 변수명 사이에 대괄호를 넣는 첫 번째 방식을 주로 사용한다.
- 배열 변수 또한 참조변수이기 때문에 힙 영역에 생성되고, 해당 메모리 주소를 저장한다.
  따라서 참조할 배열이 없다면 배열 변수도 null로 초기화 할 수 있다.

배열 타입

- 아래와 같이 중괄호({}) 내부에 값을 나열하면 배열 변수에 선언과 동시에 값을 할당할 수 있다.

```
int[] intArr1 = {1, 2, 3};
String[] strArr1 = {"로재", "리사"};

int intArr2[] = {4, 5, 6};
String strArr2[] = {"제니", "지수"};
```

- 이렇게 생성된 배열에서 값을 변경하고 싶다면, 인덱스와 대입 연산자를 사용하면 된다.

```
strArr1[0] = "로제";
```

배열 타입 [실습]

- 봄, 여름, 가을, 겨울 값을 갖는 문자열 배열 season을 생성하고 출력한 뒤,
- Spring, Summer, Autumn, Winter로 변경하여 다시 출력해보자.

배열 타입 [실습]

- 봄, 여름, 가을, 겨울 값을 갖는 문자열 배열 season을 생성하고 출력한 뒤,
- Spring, Summer, Autumn, Winter로 변경하여 다시 출력해보자.

```
String[] season = {"봄", "여름", "가을", "겨울"};

System.out.println("season[0]:" + season[0]);
System.out.println("season[1]:" + season[1]);
System.out.println("season[2]:" + season[2]);
System.out.println("season[3]:" + season[3]);

season[0] = "Spring";
season[1] = "Summer";
season[2] = "Autumn";
season[3] = "Winter";

System.out.println("season[0]:" + season[0]);
System.out.println("season[1]:" + season[1]);
System.out.println("season[2]:" + season[2]);
System.out.println("season[3]:" + season[3]);
```

배열 타입

- 중괄호({ })로 감싼 값의 목록을 배열 변수에 대입할 때, 주의할 사항이 하나 있다.
- 배열 변수를 미리 선언한 후에는 값 목록을 변수에 대입할 수 없다는 것이다.

```
String[] season;
// season = {"봄", "여름", "가을", "겨울"}; 컴파일 에러
```

- 이렇게 배열 변수를 선언한 시점과 값 목록을 대입하는 시점이 다른 경우에는 아래와 같이 new 타입[ ]을 중괄호 앞에 붙여주면 된다.

```
season = new String[] {"봄", "여름", "가을", "겨울"};
```

배열 타입

- 따라서 위의 실습 예제는 아래와 같이 변경 가능하다.

```
String[] season = {"봄", "여름", "가을", "겨울"};

System.out.println("season[0]:" + season[0]);
System.out.println("season[1]:" + season[1]);
System.out.println("season[2]:" + season[2]);
System.out.println("season[3]:" + season[3]);

season = new String[] {"Spring", "Summer", "Autumn", "Winter"};

System.out.println("season[0]:" + season[0]);
System.out.println("season[1]:" + season[1]);
System.out.println("season[2]:" + season[2]);
System.out.println("season[3]:" + season[3]);
```

배열 타입

- 메소드의 매개변수가 배열 타입인 경우에도
  매개값으로 중괄호로 감싼 값 목록만을 주면 안되고, new 타입[ ]을 중괄호 앞에 붙여서 전달해야 한다.

```
public static void main(String[] args) {
    String[] season = {"봄", "여름", "가을", "겨울"};
    printItem("season", season);

    season = new String[] {"Spring", "Summer", "Autumn", "Winter"};
    printItem("season", season);

    // printItem({"로제", "리사", "제니", "지수"});
    printItem("blackPink", new String[] {"로제", "리사", "제니", "지수"});
}

public static void printItem(String name, String[] strArr) {
    for(int i = 0; i < strArr.length; i++) {
        System.out.println(name + "[" + i + "]:" + strArr[i]);
    }
}
```

TIP

메소드는 나중에 자세히...

배열 타입

- 값의 목록은 존재하지 않지만, 나중에 값들을 저장할 목적으로 특정 길이의 배열을 미리 생성할 수도 있다.

```
타입[] 변수명 = new 타입[길이];
```

```
int[] numArr = new int[10];
boolean[] boolArr = new boolean[3];
String[] nameArr = new String[5];
```

- 이렇게 new 연산자로 특정 길이의 배열을 처음 생성하면, 배열 항목은 기본값으로 초기화된다.

타입		초기값
기본 타입	byte[], short[], int[]	0
	char[]	'/u00000'
	long[]	0L
	float[]	0.0F
	double[]	0.0
	boolean[]	false
참조 타입	클래스[], 인터페이스[]	null

배열 타입 [실습]

- 아래와 같은 형태로 다양한 데이터 타입의 초기값과 변경 값을 출력해보자.

```
int[] arr1 = new int[3];
for(int i=0; i<arr1.length; i++) {
    System.out.print(arr1[i] + ",");
}
System.out.println();

arr1[0] = 1;
arr1[1] += 2;
arr1[2] -= 3;

for(int i=0; i<arr1.length; i++) {
    System.out.print(arr1[i] + ",");
}
System.out.println("\n");
```

TIP

만약 배열이 가진 최대 인덱스보다 큰 인덱스에 접근하려고 한다면 ArrayIndexOutOfBoundsException이 발생한다.

배열 타입 [실습]

- 5개의 점수를 입력받아, 합계와 평균을 반환하는 코드를 작성해보자.

```
Scanner sc = new Scanner(System.in);

int[] score = new int[5];

for(int i = 0; i < score.length; i++) {
    System.out.println(i+1 + "번째 점수를 입력하세요");
    score[i] = Integer.parseInt(sc.next());
}

int sum = 0;

for(int i = 0; i < score.length; i++) {
    sum += score[i];
}

double avg = sum / score.length;

System.out.println("합계 : " + sum + ", 평균 : " + avg);
```

배열 타입

- 배열 안에는 또 다른 배열이 대입될 수 있는데, 이러한 배열을 다차원 배열(N차원 배열)이라고 부른다.

```
int[][] intMultiArr = { {1, 2, 3}, {4, 5} };
```

- 위의 예시는 2차원 배열이며, [ ]의 개수가 차원의 수만큼 붙는다.
  값 목록도 차원의 수만큼 중괄호가 중첩된다.

```
int[] firstArr = intMultiArr[0];
int firstArrLength = intMultiArr[0].length;
int SecondArrFirstScore = intMultiArr[1][0];
```

- new 연산자를 통해서도 다차원 배열 생성이 가능하다.

```
String[][] strMultiArr1 = new String[2][3];
String[][] strMultiArr2 = new String[2][];
strMultiArr2[0] = new String[2];
strMultiArr2[1] = new String[3];
```

배열 타입

- 기본 타입 배열은 각 항목에 값을 직접 저장하지만, 참조 타입 배열은 각 항목에 객체의 번지를 저장한다.

```
int[] numbers = new int[3];
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
```

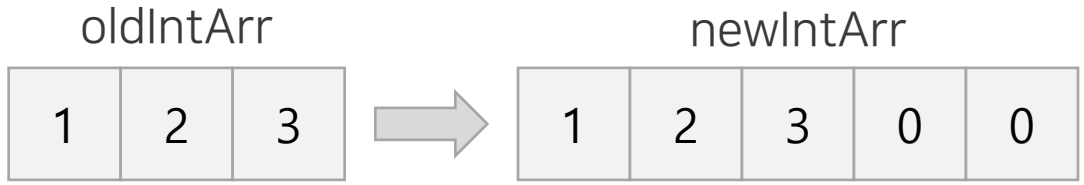
- 위의 배열 numbers는 메모리 상의 연속된 공간을 차지하며, 각 인덱스 위치에 직접 정수값이 저장된다.

```
String[] words = new String[3];
words[0] = "Hello";
words[1] = "World";
words[2] = "Java";
```

- 반면에 배열 words는 각 인덱스 위치에 문자열 객체의 메모리 주소를 저장한다.

배열 타입

- 배열은 한 번 생성하면 길이를 변경할 수 없다.
- 따라서 더 많은 저장공간이 필요하다면 더 큰 길이의 배열을 새로 만들고 이전 배열로부터 항목들을 복사해야 한다.



- 가장 기본적인 복사 방법은 for문을 이용해 항목을 하나씩 읽고, 새로운 배열에 저장하는 것이다.

```
int[] oldIntArr = new int[] {1,2,3};
int[] newIntArr = new int[5];

for(int i=0; i<oldIntArr.length; i++) {
    newIntArr[i] = oldIntArr[i];
}
```

배열 타입

- 배열 복사를 위해 보다 간단한 방법은 System 객체의 arraycopy() 메소드를 활용하는 것이다.

```
System.arraycopy(원본배열, 복사를 시작할 인덱스, 새 배열, 붙여넣기를 시작할 인덱스, 복사할 요소 개수);
```

```
int[] oldIntArr = new int[] {1,2,3};
int[] newIntArr = new int[5];

System.arraycopy(oldIntArr, 0, newIntArr, 0, oldIntArr.length);
```

배열 타입

- Java는 배열 및 컬렉션을 좀 더 쉽게 처리할 목적으로 향상된 for문을 제공한다..
- 향상된 for문을 이용하면, 카운터 변수와 증감식을 사용하지 않고도 항목의 개수만큼 반복한 후 자동으로 for문을 빠져나갑니다.

```
for (타입 변수 : 배열) {
    실행문;
}
```

TIP

컬렌션이란 여러 개의 객체를 모아 하나로 관리할 수 있도록 하는 인터페이스와 클래스들의 집합을 의미합니다.

```
int[] scores = { 95, 71, 84, 93, 87 };
int sum = 0;

for (int score : scores) {
    sum += score;
}

System.out.println("점수 총합:" + sum);
double avg = (double) sum / scores.length;
System.out.println("점수 평균:" + avg);
```

배열 타입

- main() 메서드의 매개변수에는 문자열 배열 형태의 매개변수 args가 있는 것을 볼 수 있다.

- String[] args는 main() 메서드가 호출될 때 전달되는 값으로
  윈도우의 명령 프롬프트나 맥OS의 터미널에서 프로그램을 실행하게 되는 경우 요구하는 값이 필요한 경우에 사용된다.

```java
package mypackage;

public class Sum {
	public static void main(String[] args) {
		int x = Integer.parseInt(args[0]);
		int y = Integer.parseInt(args[1]);
		System.out.println(x + y);
	}
}
```

```
Administrator@DESKTOP-A2ITP0C MINGW64 ~/src/mypackage
$ java Sum.java 10 20
30
```

- 공백으로 구분된 값은 문자열로 취급되며 String[] 배열의 요소 값으로 구성되어 main() 메서드 호출 시 매개값으로 전달된다.

배열 타입

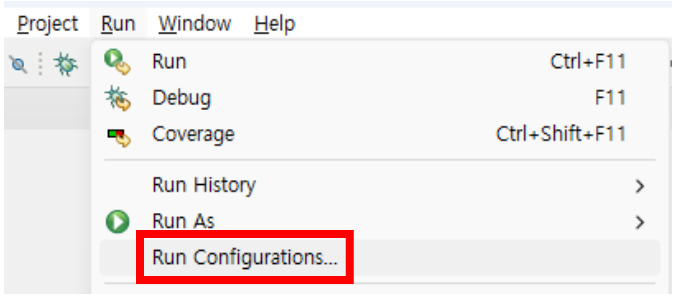
- 이클립스에서 String[] args 를 통해 입력값을 주고 실행해보자.

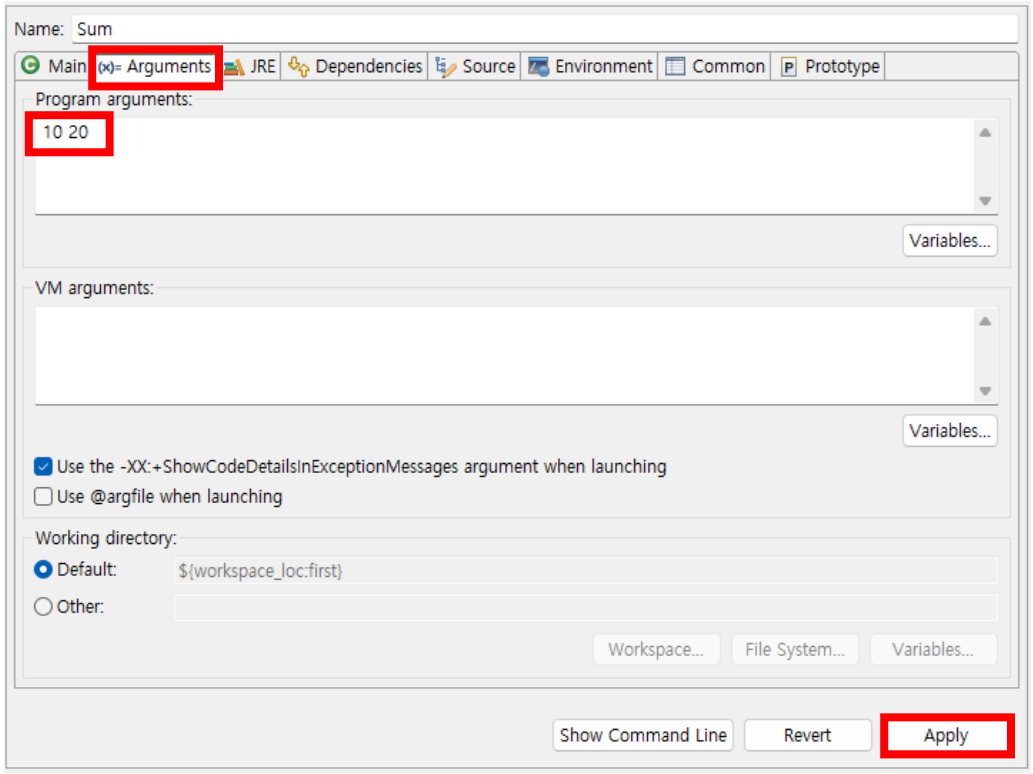
```
package mypackage;

public class Sum {
    public static void main(String[] args) {
        if(args.length != 2) {
            System.out.println("프로그램 입력값 오류");
            System.exit(0);
        }
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        System.out.println(x + y);
    }
}
```

배열 타입

- Run > Run Configuration > Arguments > Program arguments (값 입력) > Apply > Run





배열 타입

- 배열과 관련한 Arrays 클래스를 활용한 유용한 메서드들을 사용해보자. [import java.util.Arrays;]

- Arrays를 활용한 정렬 (배열을 오름차순으로 정렬한다.)

```
int [] arr = {1,44,3,6,8};
Arrays.sort(arr);
for(int i : arr) {
    System.out.print(i + ", ");
}
```

- Arrays를 활용한 배열 출력 (배열을 대괄호로 감싸고, 각 항목을 쉼표로 구분한 문자열로 반환한다.)

```
String [] strs = {"Hello", "Java", "World"};
System.out.println(Arrays.toString(strs));
```

배열 타입 [실습]

- 다음 배열의 짝수 번째 정수의 합을 구하는 프로그램을 작성하시오.
- int[] arr = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

배열 타입 [실습]

- 다음 배열의 내용을 실행 결과와 같이 출력 되도록 프로그램을 작성하시오.
- int[] arr = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

- 출력 결과 : 100 90 80 70 60 50 40 30 20 10

배열 타입 [실습]

- 다음과 같이 선언되어 있는 배열에 1~10까지의 랜덤한 숫자(중복허용)를 넣은 뒤.
  해당 배열에 어떤 값이 세팅 되었는지 출력하고 배열 데이터의 합과 평균을 구하는 프로그램을 작성하시오.
- int[] arr = new int[5]
- random 함수를 이용하고, 합은 정수형으로 평균은 실수형으로 출력한다.

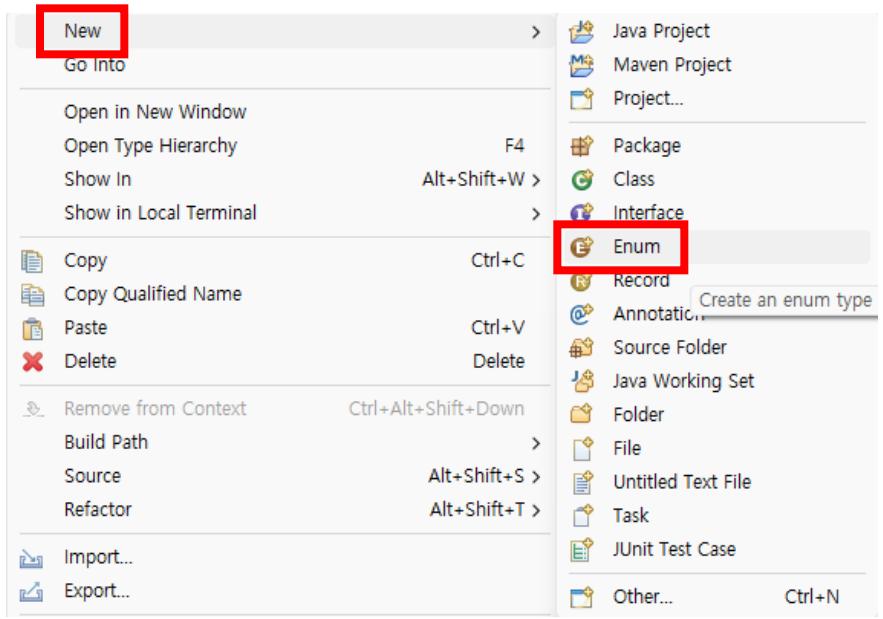
열거 타입

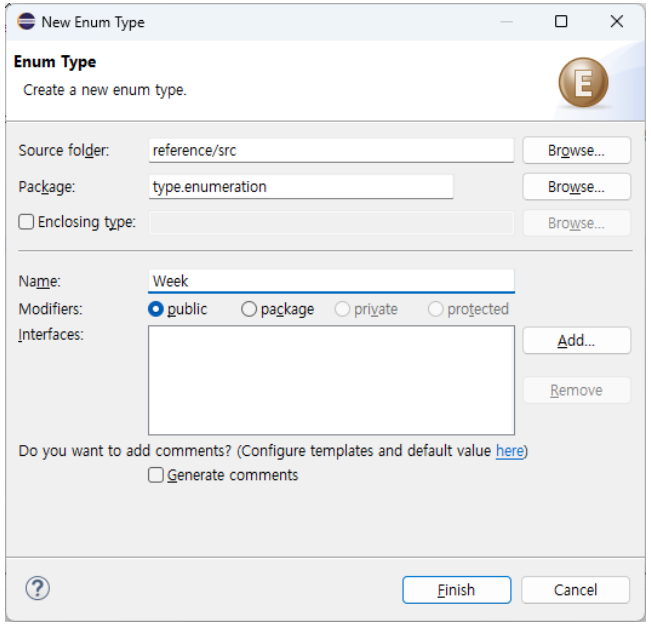
- 데이터 중에는 몇 가지로 한정된 값을 갖는 경우가 있다.
- 요일, 계절, 방향 등과 같이 한정된 값을 갖는 타입을 열거 타입(Enumeration Type)이라 한다.

- 열거 타입(Enum)은 특정한 값들의 집합을 정의하기 때문에 잘못된 값을 사용하지 않도록 강제할 수 있다.
  즉, 허용된 값만을 사용할 수 있게 한다.
- 열거 타입(Enum)을 사용하면 코드의 의미가 명확해지고 가독성이 높아진다.
- 열거 타입(Enum)을 사용하면 값의 변경이나 추가가 필요한 경우, enum의 정의만 수정하면 되므로 유지보수가 용이하다.

열거 타입

- 열거 타입(Enum)을 사용하기 위해서는 열거 타입으로 소스파일(.java)을 생성하고 한정된 값을 코드로 정의해야 한다.
- 열거 타입의 이름은 첫 문자를 대문자로 하는 Pascal Case로 지어주는 것이 관례이다.
  - Week.java, MemberGrade.java, Season.java, Direction.java






```java
package type.enumeration;


public enum Week {
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY
}
```

열거 타입

- 열거 상수는 열거 타입으로 사용할 수 있는 한정된 값을 말한다.

```
package type.enumeration;

public enum 열거타입이름 {
    열거상수목록
}
```

- 대문자 알파벳으로 정의하는 것이 관례이며,
  만약 열거 상수가 여러 단어로 구성될 경우에는 단어 사이에 언더스코어(_)로 연결한다.

```
public enum LoginResult {
    LOGIN_SUCCESS,
    LOGIN_FAILED
}
```

- 열거 타입도 하나의 데이터 타입이므로 변수를 선언하고 사용해야 한다.

```
Week today;
today = Week.MONDAY;

Week reservationDay = null;

Week yesterday = Week.SUNDAY;
```

열거 타입

- 컴퓨터의 날짜 및 요일, 시간을 얻을 때는 Calendar 클래스를 이용한다.

[import java.util.Calendar;]

TIP

Calendar 클래스에 자세한 부분은 뒤에서 알아본다.

```
Calendar now = Calendar.getInstance();
int year = now.get(Calendar.YEAR);              // 연도
int month = now.get(Calendar.MONTH) + 1;        // 월(0~11, 0: 1월)
int day = now.get(Calendar.DAY_OF_MONTH);       // 일
int week = now.get(Calendar.DAY_OF_WEEK);       // 요일(1~7, 1: 일요일)
int hour = now.get(Calendar.HOUR_OF_DAY);       // 시(0~23)
int minute = now.get(Calendar.MINUTE);          // 분
int second = now.get(Calendar.SECOND);          // 초
```

열거 타입

- Calendar 클래스를 이용하여 오늘의 요일을 구해보자.
- 요일이 1~7 사이의 숫자이므로 코드 가독성을 위해 열거 상수로 변환해 Week 변수에 대입하고 사용한다.

```
Week today = null;

Calendar cal = Calendar.getInstance();
int week = cal.get(Calendar.DAY_OF_WEEK);

switch(week) {
     case 1: today = Week.SUNDAY; break;
     case 2: today = Week.MONDAY; break;
     case 3: today = Week.TUESDAY; break;
     case 4: today = Week.WEDNESDAY; break;
     case 5: today = Week.THURSDAY; break;
     case 6: today = Week.FRIDAY; break;
     case 7: today = Week.SATURDAY; break;
}

if(today == Week.SUNDAY) {
     System.out.println("쉬는 날입니다.");
} else {
     System.out.println("열심히 공부하는 날입니다.");
}
```