JAVA 변수 자료형

프로그램은 사람이 이해하는 코드를 작성.

느려도 꾸준하면 경기에서 이긴다.

작성자 : 홍효상
이메일 : hyomee@naver.com
소스 : https://github.com/hyomee/JAVA_EDU

Content

2. 자바 변수 자료형

1. 변수
2. 기본 자료형 (Primitive Type)
3. 참조 자료형 (Reference Type)
4. 변수 구분
5. JVM 상태
6. 형 변환
7. 변수 범위

1. 변수

"메모리 공간에 부여하는 이름은 변수, 크기와 목적은 자료형"

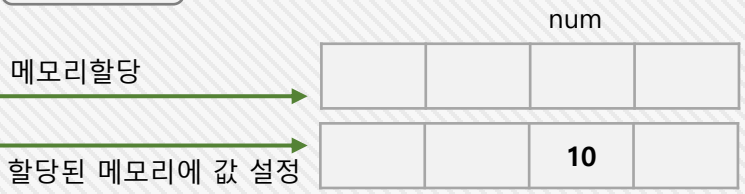
변수

- 메모리 공간에 데이터를 저장 하고 읽어 오기 위해 부여한 이름
- JVM Runtime Data Area중에 Stack Area에 저장
- 변하는 수를 언제나 변경 가능 함

01. 변수(자료형) 선언

자료형 변수명

```
int num;
num = 10;
```



➢ 변수 이름 짓기 규칙

- 영문 대소 문자, 한글 사용 가능
- 특수문자는 밑줄(_), 달러($) 표기만 사용 가능
- 아라비아 숫자 가능. 단 첫 문자는 숫자 불가
- 자바에서 사용하는 예약어 불가

* 일반적으로 Camel Case(카멜 표기법) 사용
* 상수는 대문자 사용 (final)

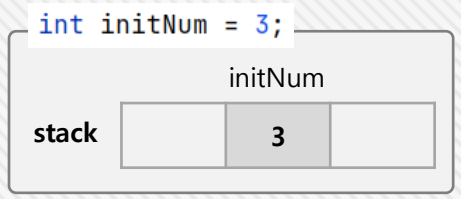
자료형 (Data Type)

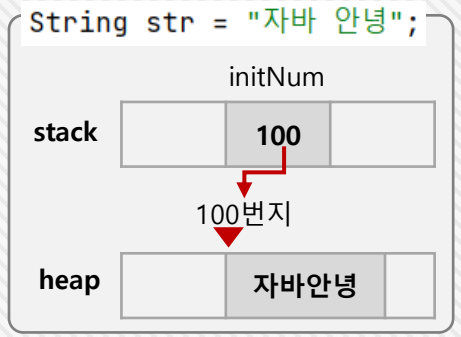
- 데이터를 메모리 공간에 저장 하는 목적에 따라 크기와 특징을 구분 해 애 하는데 이것이 자료형

02. 자료형 종류

- 기본 자료형 (Primitive Type)
  - 부울대수 (Boolean Type)
  - 숫자 (Numeric Type)
    - 안전한 (Integral Type)
      - 정수형 (Integer Type)
        - byte, short, int, long
      - 실수형 (Floating Type)
        - float, double
    - 문자 (Character Type)
      - char
- 참조 자료형 (Reference Type)
  - 배열 (Array Type)
  - Enum Type
  - Class Type
    - String Class
    - Wrapper Class

...





1. 변수

03. 변수(자료형) 사용 하기

```java
public class VariableDeclared {

    public static void main(String[] args) {
        // 1. 변수 선언과 값 대입 분리
        int num;        // 정수형 변수 num 선언
        num = 3;        // 정수형 변수 num에 3 저장

        // 2. 변수 선언과 값 대입 동시
        int initNum = 3;
        System.out.println(String.format("정수 num=%s, initNum=%s", num, initNum));

        double doubleNum = 5;
        doubleNum = 7.2;
        System.out.println(String.format("실수 doubleNum=%f", doubleNum));

        String str = "자바 안녕";
        // str = 10;   // 오류 :: String 형에는 문자만 가능
        System.out.println(String.format("문자열 str=%s", str));
    }
}
```

```
정수 num=3, initNum=3
실수 doubleNum=7.200000
문자열 str=자바 안녕
```

1. 기본 자료형 (Primitive Type)

기본 자료형 (Primitive Type)

- 자바에서 기본 자료형은 반드시 사용하기 전에 선언(Declared)되어야 함
- OS에 따라 자료형의 길이가 변하지 않음
- 비 객체 타입입니다. 따라서 null 값을 가질 수 없음

Type				Byte	Range of Values
부울대수 (Boolean Type)			boolean	1bit)	true, false
숫자 (Numeric Type)	안전한 (Integral Type)	정수형 (Integer Type)	byte	1 Byte (8 bit)	-2^7 ~ 2^7-1 (-128 ~ 127)
			short	2 Byte (16 bit)	-2^15 ~ 2^15-1 (-32768 ~ 32767)
			int	4 Byte (32 bit)	-2^31 ~ 2^31-1 (-2147483648 ~ 2147483647)
			long	8 Byte (64 bit)	-2^63 ~ 2^63-1 (-9223372036854775808 ~ 9223372036854775807)
		실수형 (Floating Type)	float	4 Byte (32 bit)	0x0.000002P-126f ~ 0x1.fffffeP+127f
			double	8 Byte (64 bit)	0x0.0000000000001P-1022 ~ 0x1.fffffffffffffP+1023
문자 (Character Type)			char	2 Byte (16 bit)	₩u0000 ~ ₩uffff (0 ~ 2^15-1)

- 문자 (Character Type)는 자바에서 unsigned로 동작하는 자료형
- BigInteger : 연산자에는 사용 하지 않음

2. 기본 자료형 사용하기

01. 부울 대수 / 정수형 사용하기

- 정수 리터럴이 선언한 변수 범위에 있으면 선언한 형으로 인식
- 범위 밖에 있으면 기본 int형으로 인식 하여 오류 발생 -> 형 변환 필요

- long : 정수리터럴 + L (or l)

```
public class PrimitiveTypeBooleanNumeric {
    public static void main(String[] args) {
        boolean isVar = true;
        if (isVar) {
            isVar = false;
        }
        // 정수 리터럴이 선언한 변수 범위에 있으면 선언한 형으로 인식 하지만
        // 범위 밖에 있으면 기본 int형으로 인식 하여 오류 발생 -> 형 변환 필요
        byte valueByte = 10;
        short valueShort = 10;
        int valueInt = 10;
        long valueLong = 10;
        long valueLongL = 10L;
        // 범위 밖에 값이면 오류 발생 ( 32768는 int로 인식 ) -> 형변환 해야 암
        //short valueByteOver = 32768;
        short valueByteOver = (short) 32768;

        System.out.println(String.format("boolean boolean = %s", isVar));
        System.out.println(String.format("byte    byte = %s", valueByte));
        System.out.println(String.format("short   short = %s", valueByte));
        System.out.println(String.format("int     int = %s", valueInt));
        System.out.println(String.format("long    long = %s", valueLong));
        System.out.println(String.format("long    long L = %s", valueLongL));
        System.out.println(String.format("short   범위밖 = %s", valueByteOver));
    }
}
```

```
boolean boolean = false
byte    byte = 10
short   short = 10
int     int = 10
long    long = 10
long    long L = 10
short   범위밖 = -32768
```

2. 기본 자료형 사용하기

02. 실수형 사용하기

- 실수형 자료형에서 실수 리터럴은 **기본이 double 형**
- 소수점 (실수 리터럴)은 float형에 넣으면 오류 발생 -> 형 반환 필요
- **정수형 리터럴은 자동 변환 됨**

- **float : 실수리터럴 + F (or f)**

```java
public class PrimitiveFloat {
    public static void main(String[] args) {
        // 실수형 자료형에서 실수 리터럴은 기본이 double 형
        // 소수점 (실수 리터럴)은 float형에 넣으면 오류 발생 -> 형반환 필요
        // 정수형 리터럴은 자동 변환 됨

        float numericLiteralToFloat = 10;
        float floatingLiteralToFloat = 10.25F;

        // 오류 : 형변환 필요
        // float valueFloat = 10.25;
        float floatingLiteralToCastingFloat = (float) 10.25;

        double floatingLiteralTeDouble = 10.25;
        double numericLiteralToDouble = 10;

        System.out.println(String.format("float numericLiteralToFloat = %f", numericLiteralToFloat));
        System.out.println(String.format("float floatingLiteralToFloat = %f", floatingLiteralToFloat));
        System.out.println(String.format("float floatingLiteralToCastingFloat = %f", floatingLiteralToCastingFloat));
        System.out.println(String.format("double floatingLiteralTeDouble = %f", floatingLiteralTeDouble));
        System.out.println(String.format("double numericLiteralToDouble = %f", numericLiteralToDouble));
    }
}
```

```
float numericLiteralToFloat = 10.000000
float floatingLiteralToFloat = 10.250000
float floatingLiteralToCastingFloat = 10.250000
double floatingLiteralTeDouble = 10.250000
double numericLiteralToDouble = 10.000000
```

2. 기본 자료형 사용하기

03. 문자형 사용 하기

- **문자형은 유니코드로 변환 해서 메모리에 저장 함**
  - 유니코드 : 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이다. 유니코드는 유니코드 협회(Unicode Consortium)가 제정 함
- **문자형은 한 문자를 의미하며 문자 ('A'), 10진수(65), 2진수 (0b1000001), 8진수(00101), 16진수(0x0041), 유니코드('₩u0041')를 저장**

```java
public class PrimitiveChar {
    public static void main(String[] arg) {
        char charliteralToChar = 'A';
        char charIntegerliteralToChar = '3';
        // '10'은 문자 2개로 char 범위 밖으로 오류
        // char charIntegerliteralToChar = '10';
        char integerliteralToChar = 65;
        char bianeyliteralToChar = 0b1000001;
        char octalliteralToChar = 00101;
        char hexadecimalToChar = 0x0041;
        char unicodeToChar = '\u0041';

        System.out.println(String.format("문자       = %s", charliteralToChar));
        System.out.println(String.format("숫자문자    = %s", charIntegerliteralToChar));
        System.out.println(String.format("10진수     = %s", integerliteralToChar));
        System.out.println(String.format("2진수      = %s", bianeyliteralToChar));
        System.out.println(String.format("8진수      = %s", octalliteralToChar));
        System.out.println(String.format("16진수     = %s", hexadecimalToChar));
        System.out.println(String.format("유니코드    = %s", unicodeToChar));
    }
}
```

```
문자       = A
숫자문자    = 3
10진수     = A
2진수      = A
8진수      = A
16진수     = A
유니코드    = A
```

2. 기본 자료형 사용하기

03. 문자형 사용 하기

- 진법 변환은 십진수ToN진수 (Integer.toXXXXString()), N진수To 10진수 (Integer.parserXXX())사용

```java
public class DecimalConversion {
    public static void main(String[] args) {
        int decomalNum = 12;
        String binaryNum = "1100";
        String octalNum = "14";
        String hexNum = "c";

        System.out.println("2진수 변환 = " + Integer.toBinaryString(decomalNum));
        System.out.println("8진수 변환 = " + Integer.toOctalString(decomalNum));
        System.out.println("16진수 변환 = " + Integer.toHexString(decomalNum));

        System.out.println("2진수->10진수 변환 = " + Integer.parseInt(binaryNum, radix: 2));
        System.out.println("8진수->10진수 변환 = " + Integer.parseInt(octalNum, radix: 8));
        System.out.println("16진수->10진수 변환 = " + Integer.parseInt(hexNum, radix: 16));
    }
}
```

```
2진수 변환 = 1100
8진수 변환 = 14
16진수 변환 = c
2진수->10진수 변환 = 12
8진수->10진수 변환 = 12
16진수->10진수 변환 = 12
```

1. 참조 자료형 (Reference Type)

참조 자료형 (Reference Type)

- 기본적으로 java.lang.Object를 상속 받으면 참조형

Type		설명
배열 (Array Type)		• 기본형으로도 만들 수 있고 참조형으로 만들 수 있음
Enum Type	byte	• 열거형, • String 클래스와 마찬가지로 **불변의 객체**, • 상수의 집합을 만들거나 특정 객체의 상태를 모아서 열거형을 만듦
Class Type	String Class	• 참조형에 속하지만 기본적인 사용은 기본형처럼 사용 • 불변하는immutable 객체 • String 클래스에는 값을 변경해주는 메소드들이 존재하지만 해당 메소드를 통해 데이터를 바꾼다 해도 새로운 String 클래스 객체를 만들어내는 것 • String 객체간의 비교는 .equals() 메소드를 사용
	Wrapper Class	• 기본형에 null을 넣고 싶다면 래퍼 클래스Wrapper Class를 활용 기본형 대응 래퍼 클래스 byte Byte short Short int Integer long Long float Float double Double char Char boolean Boolean

1. 변수 구분

지역 변수 (로컬변수)

- 메소드 내부에서 정의 되어 사용 하는 변수
- 자동으로 초기화 되지 않음
- 매개변수도 지역 변수
  : 메소드이 인자로 사용되는 변수

인스턴스 변수

- static 으로 선언 되어 있지 않는 모든 멤버 변수
- 객체(클래스의 인스턴스)는 자신만의 복사본을 Heap에 저장 함
  : new로 생성시 마다 Heap에 할당
  : 인스턴스 변수의 값은 각각이 객체와 구분 됨

클래스 변수

- 객체(클래스의 인스턴스)가 아니라 정의된 클래스와 연관되므로 Runtime Data Area의 Method Area에 한 개 존재
  : 객체를 많이 생성 해도 하나만 존재 함
  : 초기화가 한번만 실행
- static 한정자
  - 생성시점 : 최초 new하는 경우 , Class가 최초로 참조 되는 경우
  - 일반적으로 상수로 사용
    static final double PI=3.14;
    Class.클래스변수로 접근 : ClassName.PI

변수 자동 초기화

- 클래스, 인스턴스 변수는 자동 초기화 됨
  : boolean -> false
  : char -> '₩u0000'
  : Byte : short : int : long -> 0
  : Float -> 0.0f
  : Double -> 0.0d
  : Object type -> null

- 자동으로 초기화 되지 않음
  : 지역변수, 매개변수
  : 매개변수도 지역 변수

2. 변수 구분 예제

01. 변수 초기화 예제

```java
public class VariableBase {
    public static void main(String[] args) {
        InitVariable initVariable = new InitVariable();
        initVariable.printInitVariable();
    }
}
```

```
Field mvBoolean  :: false
Field mvChar :: 
Field mvByte :: 0
Field mvShort :: 0
Field mvInt :: 0
Field mvLong :: 0
Field mvFloat :: 0.0
Field mvDouble :: 0.0
Field mvObject :: null
```

```java
public class InitVariable {
    private boolean mvBoolean;
    private char mvChar;
    private byte mvByte;
    private short mvShort;
    private int mvInt;
    private long mvLong;
    private float mvFloat;
    private double mvDouble;
    private Object mvObject;

    public InitVariable() {}

    public void printInitVariable() {
        int localNum; // 초기화 되지 않음 오류
        // System.out.println(String.format("localNum  :: %s", localNum));
        System.out.println(String.format("Field mvBoolean  :: %s", mvBoolean));
        System.out.println(String.format("Field mvChar :: %s", mvChar));
        System.out.println(String.format("Field mvByte :: %s", mvByte));
        System.out.println(String.format("Field mvShort :: %s", mvShort));
        System.out.println(String.format("Field mvInt :: %s", mvInt));
        System.out.println(String.format("Field mvLong :: %s", mvLong));
        System.out.println(String.format("Field mvFloat :: %s", mvFloat));
        System.out.println(String.format("Field mvDouble :: %s", mvDouble));
        System.out.println(String.format("Field mvObject :: %s", mvObject));
    }
}
```

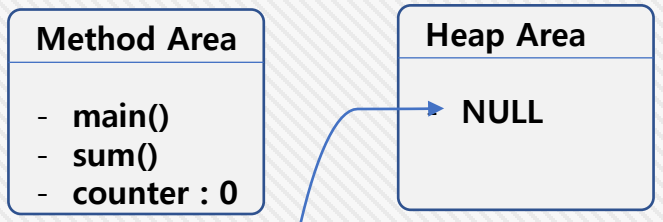
1. JVM 상태

01. JVM 상태

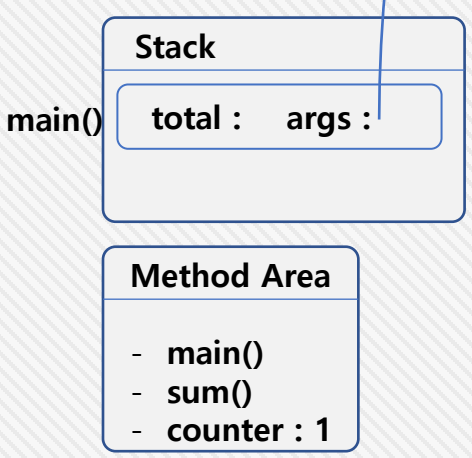
```java
public class JvmVariableCycle {
    static int counter;
    public static void main(String[] args) {
        int total = sum( i: 10,  j: 30);
        System.out.println("합계 = " + total);
    }

    static int sum(int i, int j) {
        int sum = i + j;
        counter = counter + 1;
        return sum;
    }
}
```

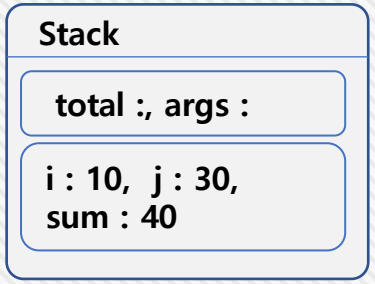
1. JvmVariableCycle Class가 시작 할 때 할당 됨



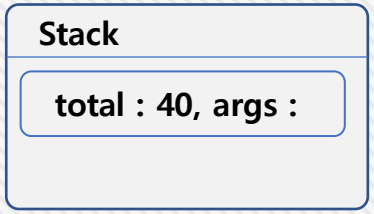
2. 메인 실행



3. SUM 실행



3. SUM 종료



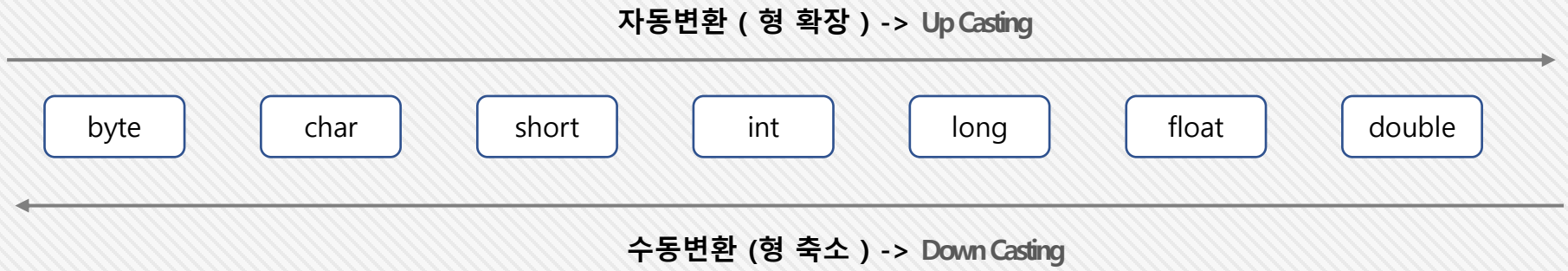
* MAIN 종료 후 모두 사라짐

1. 형 변환

" 시스템에서 자동으로 하는 자동 변환(Up Casting), 개발자에 의해 강제로 하는 수동 변환 (Down Casting) "

자료형 변환

- 자료형 크기가 큰 쪽, 범위가 넓은 쪽으로 자동 변환 되는 것 -> 형 확장 -> **자동 변환** -> Up Casting
  - 범위 안에 있는 경우 자동 Casting 됨
- 자료형 크기가 작은 쪽, 범위가 좁은 쪽으로 자동 변환 되는 것 -> 형 축소 -> **수동변환** -> Down Casting




```java
// 자동 변환
long valueLong = 10;        // int -> long ( Up )
float valueFloat = 10;      // int -> float ( Up )
double valueDouble = 10;    // int -> double ( Up )
byte valueByte = 10;        // int -> byte
short valueShort = 10;      // int -> short
```

```java
// 수동 변환
byte valueByteCasting = (byte) 100;       // int -> byte ( Down )
int valueIntCasting = (int) 3.5;          // double -> int ( Down )
float valueFloatCasting = (float) 3.5;    // double -> float ( Down )
```

1. 형 변환


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

        // 자동 변환
        long valueLong = 10;        // int -> long ( Up )
        float valueFloat = 10;      // int -> float ( Up )
        double valueDouble = 10;    // int -> double ( Up )
        byte valueByte = 10;        // int -> byte
        short valueShort = 10;      // int -> short

        System.out.println("int -> long ( Up ) = " + valueLong);
        System.out.println("int -> float ( Up ) = " + valueFloat);
        System.out.println("int -> double ( Up ) = " + valueDouble);
        System.out.println("int -> byte = " + valueByte);
        System.out.println("int -> short = " + valueShort);

        // 수동 변환
        byte valueByteCasting = (byte) 100;        // int -> byte ( Down )
        int valueIntCasting = (int) 3.5;           // double -> int ( Down )
        float valueFloatCasting = (float) 3.5;     // double -> float ( Down )

        System.out.println("int -> byte ( Down ) = " + valueByteCasting);
        System.out.println("double -> int ( Down ) = " + valueIntCasting);
        System.out.println("double -> float ( Down ) = " + valueFloatCasting);

        // 정수형은 작은 범위의 자료형으로 down Castring 하면 서쿨러 구조 임
        byte valueByte128 = (byte) 128;
        byte valueByte129 = (byte) 129;
        byte valueByte_129 = (byte) -129;
        byte valueByte_130 = (byte) -130;
        System.out.println("(byte) 128 = " + valueByte128);
        System.out.println("(byte) 129 = " + valueByte129);
        System.out.println("(byte) -129 = " + valueByte_129);
        System.out.println("(byte) -130 = " + valueByte_130);
    }
}
```

```
int -> long ( Up ) = 10
int -> float ( Up ) = 10.0
int -> double ( Up ) = 10.0
int -> byte = 10
int -> short = 10
int -> byte ( Down ) = 100
double -> int ( Down ) = 3
double -> float ( Down ) = 3.5
(byte) 128 = -128
(byte) 129 = -127
(byte) -129 = 127
(byte) -130 = 126
```

1. 변수 범위

" 위에서 아래는 사용 가능 "

변수 범위

- 변수의 선언 위치에 따라서 변수의 수명이 지역 범위를 가지는 지역 변수와 프로그램 전체에 영향을 주는 전역 변수.
  외부에서 참조 시 public
- 변수 선언 시 static, const keyword를 사용 하면 의미가 변경 됨