



Data

경북대학교
소프트웨어융합과
배희호 교수



Data의 의미(意味)



- 모든 Program은 Data를 대상으로 연산(Operation) 행위를 수행
- Data Process(데이터 처리)
 - Data를 기억 장치(Memory)에 저장
 - 장치들 사이에서 이동
 - 계산 등 가공
- 장치들이 처리하기에 적합한 형태로 표현
- 숫자 Data, 문자 Data 등의 다양한 형태가 존재



Data Type



Data 종류		내용
숫자 (digit)	정수 (integer)	소수점이 없는 수
		-345, 0, 123, +1234, 1
	실수(float)	소수점을 갖고 있는 수
		+345.568, -3.45, 0.0, 0.0123 -0.34568x10 ⁻² , 345.68x10 ²³
문자(character)		문자, 숫자나 symbol의 한 글자
		‘A’, ‘a’, ‘G’, ‘+’, ‘&’, ‘1’, ‘가’, ‘ ’
문자열(string)		문자들의 조합
		“A”, “apple”, “Hello World”, “가”, “한글”, “123”, “”

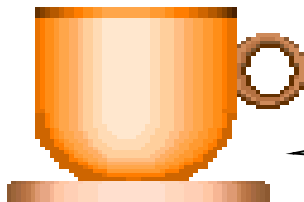


Data Type

- Computer에서 딸기와 물은 Data임
 - 딸기는 과일류를 저장하는 그릇(광주리)이 필요
 - 물은 액체를 저장하는 그릇(컵)이 필요
 - 딸기를 담는 광주리에 물을 담을 수는 없음
- Computer에서는 Data의 형태에 따라 저장(기억) 구조가 다름(Data Type)
- Data Type은 Data의 형태를 정의하는 것



광주리



컵

용기의 종류에
따라 담을 수 있는
내용물도 달라짐



Data Type



- Data의 내부 표현
 - Data 표현 방식의 차이
 - Source Code에 있는 상수와 입력 Data
 - 사람이 읽기에 적합한 형태
 - Compile 후의 Data
 - Computer가 처리하기에 적합한 형태
- Data 표현 방식을 누가 바꾸는가?
 - 상수
 - Compiler가 Compile할 때 변환
 - 입력 Data
 - JDK Library의 메소드를 이용하여 변환 가능



Data Type



■ Memory : 그릇(용기)

- Switch의 접속(ON : 1) 또는 단절(OFF : 0)의 2가지 상태 중의 1가지 상태만을 나타낼 수 있는 전자 회로들의 모임
- 이것을 나타낼 수 있는 단위를 비트(bit)
- 4개의 Bit를 니블(nibble)
- 8개의 Bit를 바이트(byte)
- 2 또는 4개의 Byte를 워드(word)
- 4 또는 8개의 Byte를 더블 워드(double word)



Data Type

- 변수가 가질 수 있는 값의 형태
- 물건을 정리하는 상자도 다양한 Type이 있듯이 **Data를 저장하는 변수도 다양한 종류가 있음**
- JAVA는 다양한 형태의 Data Type을 제공

- JAVA의 Data Type

- Primitive Data Type(기본 데이터 형)
- Reference Data Type(참조 데이터 형)



데이터 형에는 여러 가지 종류가 있음



Data Type

- JAVA Program의 모든 변수와 Data는 Type이 있음

변수에도
모두
Data
Type이
있음

```
public class Simple {  
    public static void main(String args[]) {  
        double result;  
        int a = 1, b = 2;  
        double c = 0.5;  
        result = (a + b) * c;  
        System.out.println("result =" + result);  
    }  
}
```

상수에도 모두
Data Type이
있음

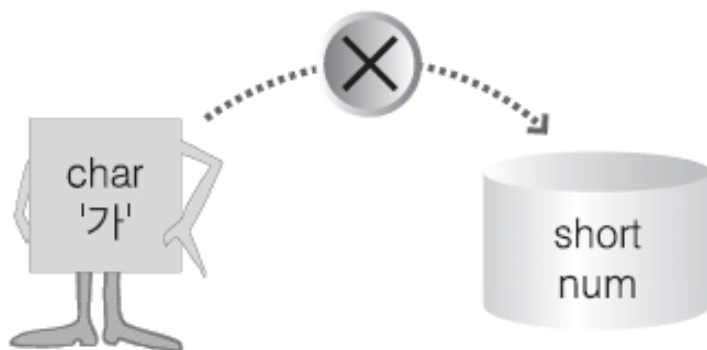
이런 계산의 중간 결과에도
내부적으로 Data Type이 부여 됨



Data Type



■ Data Type으로 인해 발생할 수 있는 문제



변수와 다른 타입의 값을
대입하면 에러가 발생할 수 있습니다.

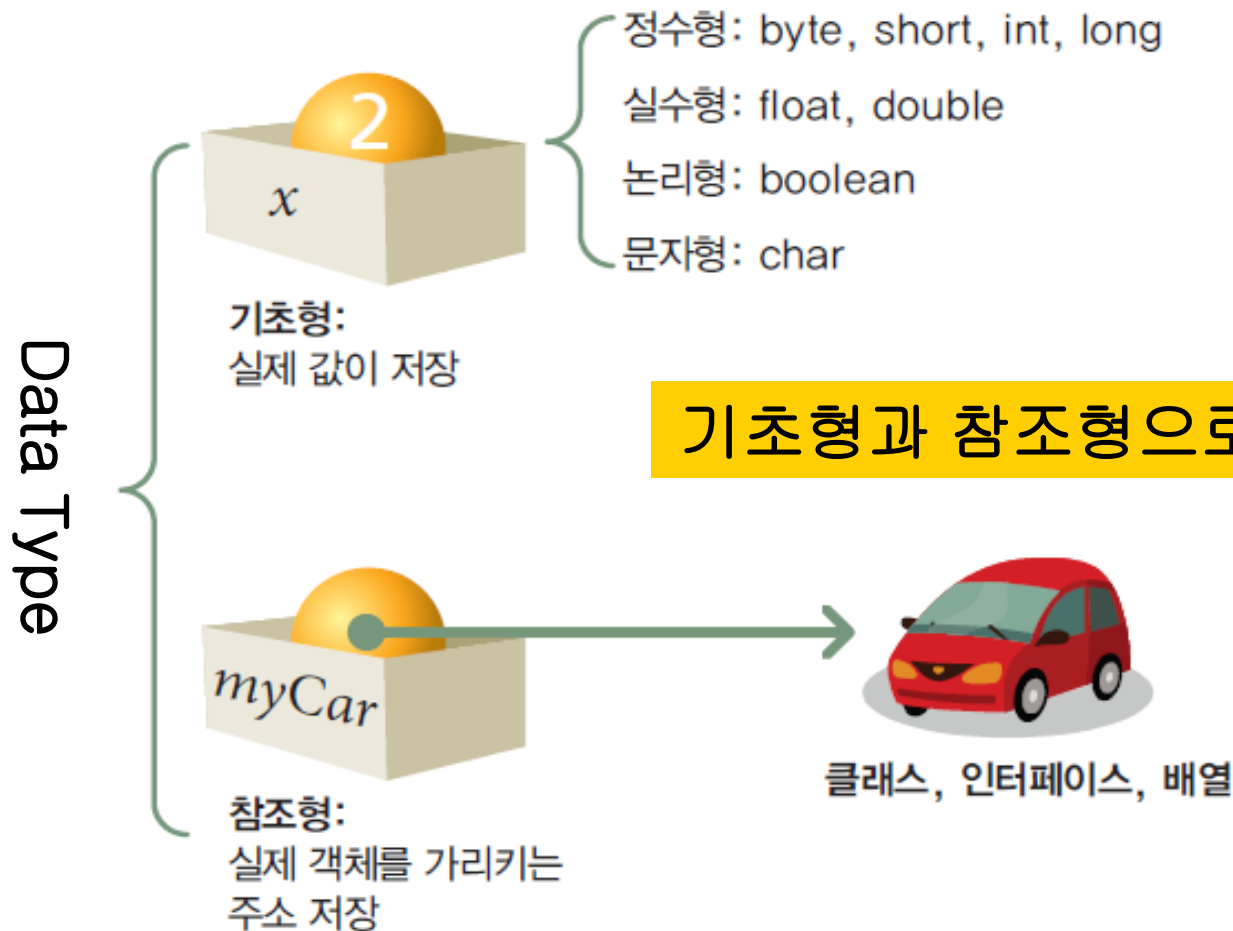


변수와 다른 타입의 값을 담는 도중에
데이터 손실이 일어날 수도 있습니다.



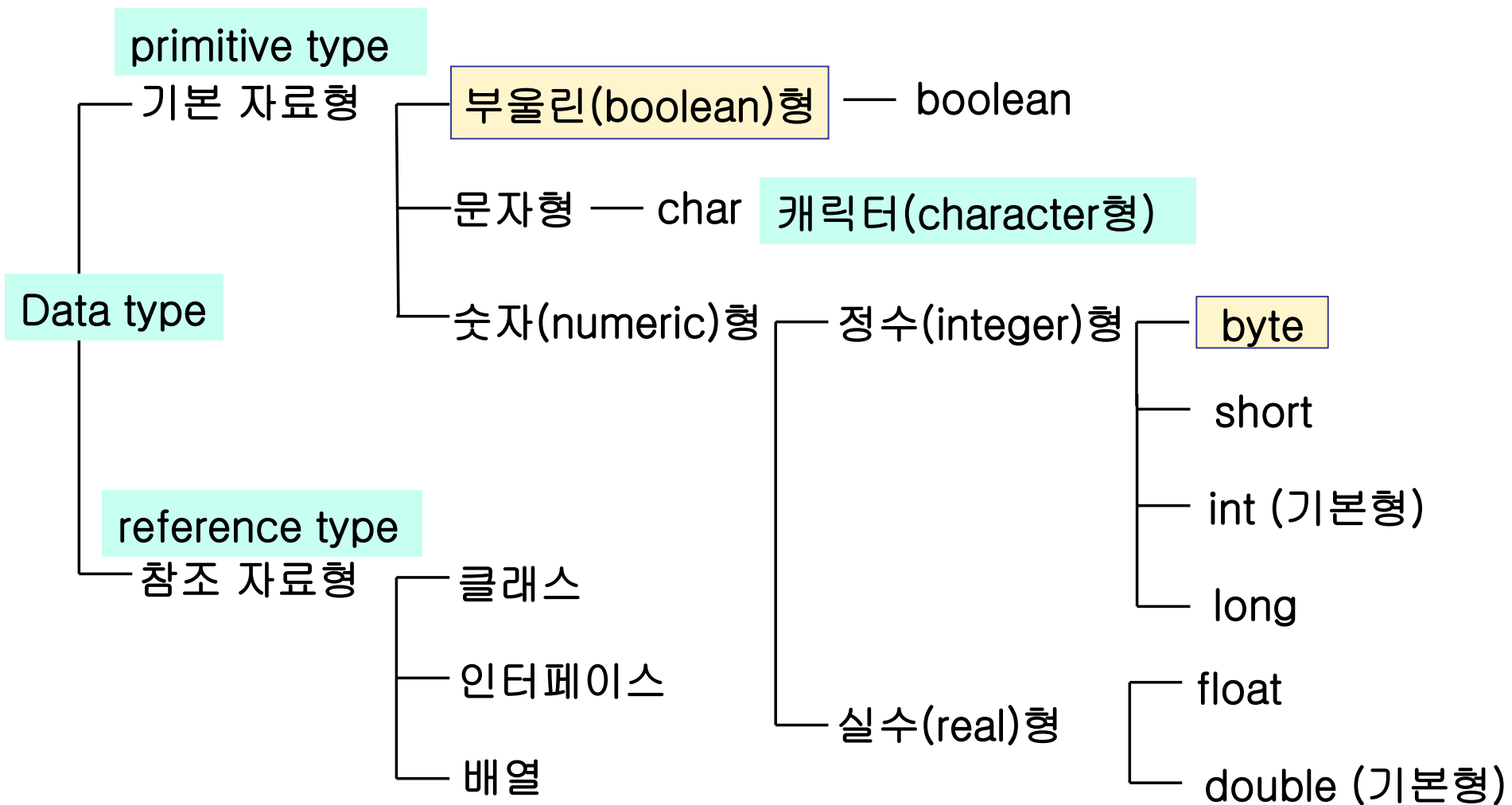
Data Type

Data Type의 분류





Data Type





Data Type



■ Primitive Data Type


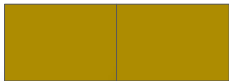
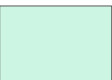
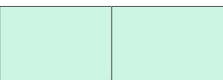
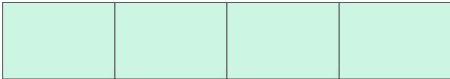
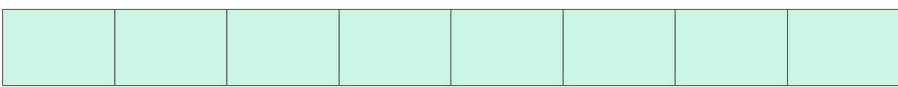
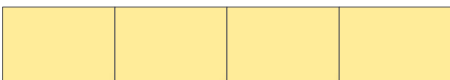
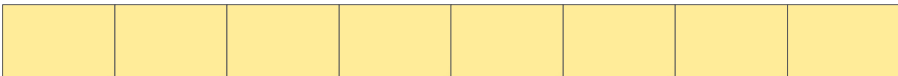
- 일정한 크기와 형식으로 된 하나의 값
- Primitive Data Type의 크기는 정해져 있음
 - Primitive Data Type의 크기는 CPU나 운영체제에 따라 변하지 않음
- 1개의 수, 1개의 문자, 또는 하나의 진리 값
- 정수, 실수, 문자, 논리값 등과 같은 8가지의 Primitive Data Type
- 정수는 32 Bit로 **2의 보수 표기법**으로 표현 수
- 문자는 Unicode로 표기된 16 Bit 자료로 표현
- Primitive Data Type은 대응되는 클래스 존재
 - 예) int 형 -> Integer 클래스
- Primitive Data Type 변수는 값을 가짐
- 매개변수 전달 : call by value



Data Type



■ Primitive Data Type

boolean		<i>true, false</i>
char		<i>유니코드</i>
byte		<i>-128부터 127</i>
short		<i>-32,768부터 32,767</i>
int		<i>약 -21억부터 21억까지</i>
long		<i>-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807</i>
float		<i>유효숫자(정밀도) 7자리</i>
double		<i>유효숫자(정밀도) 15자리</i>



Data Type



■ Reference Data Type

- 배열, 문자열, 클래스, 인터페이스 등과 같은 Data Type
- 객체에 대한 참조 즉 주소(Address)를 가지고 있음
- 매개변수 전달 : call by reference

Reference Data Type	설 명
클래스 참조형	클래스 객체를 참조하기 위한 자료형
인터페이스 참조형	인터페이스 객체를 참조하기 위한 자료형
배열 참조형	배열 객체를 참조하기 위한 자료형

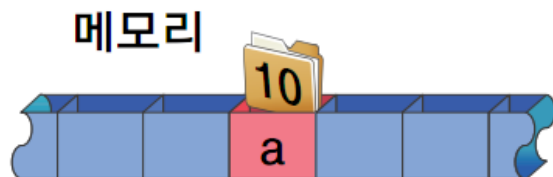


Data Type

Primitive Data Type과 Reference Data Type의 차이

```
int a = 10;
```

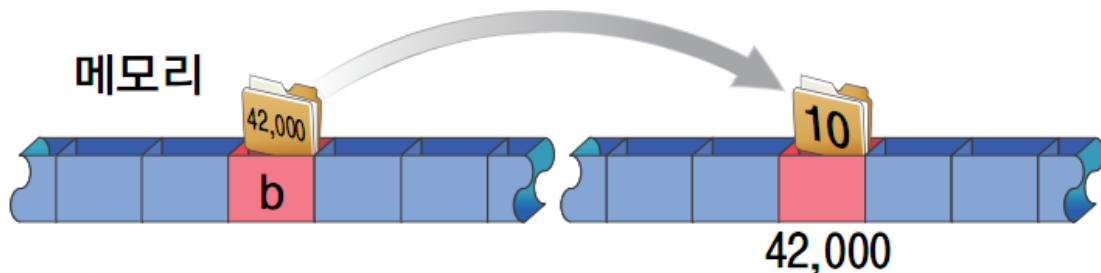
기본 자료형으로서 변수로 지정된 위치에 값이 저장되어 있습니다.



한 번의 접근으로 값을 가져올 수 있습니다.

```
Integer b = new Integer(10);
```

참조 자료형으로서 변수로 지정된 위치에는 실제 값이 있는 곳의 주소가 저장됩니다.



두 번의 접근으로 값을 가져올 수 있으므로 기본 자료형에 비해 효율성이 떨어질 수 있습니다.



Data Type



■ Primitive Data Type

기본 자료형	크기	설명
부울린 형 (boolean)	1 bit	이진형은 참 또는 거짓을 포함하나 다른형으로 전환(casting)할 수 없음
문자형(char)	16 bit, unsigned int	각 문자형은 유니코드
바이트형(byte)	8 bit, 부호있는 2의 보수	$-128(-2^7) \sim +127(+2^7 - 1)$
단축형(short)	16 bit, 부호있는 2의 보수	$-32,768(-2^{15}) \sim +32,767(+2^{15} - 1)$
정수형(int)	32 bit, 부호있는 2의 보수	$-2,147,483,648 \sim +2,147,483,647$
★ long 형	64 bit, 부호있는 2의 보수	$-2^{63} \sim +2^{63}-1$
float 형	32 bit, IEEE 754 단일배정도	$-3.4E38 \sim +3.4E38$
★ double 형	64 bit, IEEE 754 이중배정도	$-1.7E308 \sim +1.7E308$



Data Type



■ Primitive Data Type에 따른 기본 값의 배정

자료형	기본 값	초기 값
boolean 형	false	
char 형	null 문자	'\u0000'
byte 형	zero	(byte) 0
short 형	zero	(short) 0
int 형	zero	0
long 형	zero	0L
float 형	양의 zero	0.0f
double 형	양의 zero	0.0d
reference 형	null	



Boolean



- Boolean(부울린형, 논리형)
 - boolean 예약어 사용
 - 1 bit 크기의 기억 장소 할당
 - 논리 값 true(참) 또는 false(거짓) 중의 한 값을 갖는 Data Type
 - Default 값은 false(거짓) 임
 - 숫자 값 가질 수 없음
 - 다른 Data Type으로 변환되지 않음

boolean condition = **true**;



오류주의

C나 C++ 언어에서는 정수값이 논리형으로 사용된다. 0은 **false**에 해당되고 나머지값은 **true**에 해당된다. 그러나 자바에서는 그렇지 않다. 따라서 정수값을 논리형으로 형변환할 수 없다.



Boolean 실습

초기값을 대입하지
않으면 오류 발생

Filename : **BooleanTest.java**

```
public class BooleanTest1 {  
    public static void main( String[] args) {  
        boolean flag = false;  
        System.out.println(" flag = " + flag);  
        flag = true;  
        System.out.println(" flag = " + flag);  
        flag = (1 > 2);  
        System.out.println(" flag = " + flag);  
    }  
}
```



Boolean 실습



Filename : **BooleanTest1.java**

```
public class BooleanTest1 {  
    public static void main( String[] args) {  
        int num;  
        num = 10 + 20;  
        if (num > 10)  
            System.out.println(" 계산 결과가 10보다 큼니다.");  
    }  
}
```



Boolean 실습



Filename : BooleanTest2.java

```
public class BooleanTest2 {  
    public static void main( String[] args) {  
        while (true)  
            System.out.println(" Hello, Java");  
    }  
}
```



Boolean 실습



Filename : **BooleanTest3.java**

```
class BooleanTest3 {  
    public static void main(String args[]) {  
        int num = 10 + 20;  
        boolean truth;           // boolean 타입의 변수를 선언  
        truth = num > 10;        // boolean 타입의 변수에 조건식의 결과 대입  
        if (truth)               // boolean 타입의 변수를 사용  
            System.out.println("계산 결과가 10보다 큽니다.");  
    }  
}
```



Char



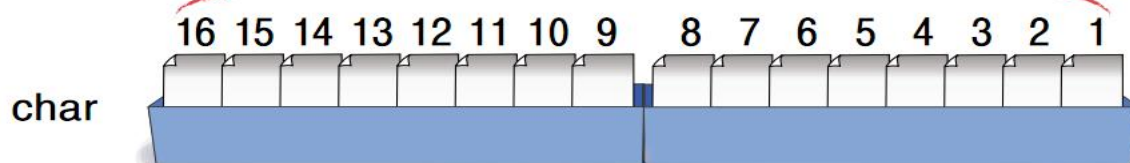
- char 형(문자형)
 - char 예약어 사용
 - 하나의 문자를 나타낼 수 있는 char형을 기본 자료형으로 제공
 - ASCII(8 Bit) 체계가 아닌 Unicode(16 Bit)를 사용
 - Unicode 규격 중에서 UTF-16 규격 사용
 - 세계 다양한 나라들의 모든 언어를 나타낼 수 있음
 - 한글 음절 11,172자
 - 초성 19 × 중성 21 × 종성 28 = 11,172
 - 일본어, 중국어 등……
 - ASCII 코드
 - 최대 65,536 개의 문자를 나타낼 수 있음
 - int형으로 형 변환 가능(0 ~ 65535 범위의 정수)



문자형

■ 문자형과 유니코드(unicode)

2 byte(=16 bit)로 구성



총 2^{16} 개 (65, 536)개의
문자표현 가능

	AC0	AC1	AC2	AC3	AC4	AC5
0	가 AC00	감 AC10	겐 AC20	갠 AC30	갈 AC40	작 AC50
1	각 AC01	갑 AC11	갸 AC21	갹 AC31	갊 AC41	갋 AC51
2	각 AC02	값 AC12	갸 AC22	갹 AC32	갊 AC42	갋 AC52
3	값 AC03	갸 AC13	갹 AC23	갊 AC33	갋 AC43	갌 AC53

가 : AC00
각 : AC01

(a) 한글 유니코드

	304	305	306	307	308
0	ㄱ 3040	가 3050	다 3060	바 3070	마 3080
1	애 3041	케 3051	치 3061	페 3071	메 3081
2	애 3042	게 3052	치 3062	히 3072	모 3082
3	이 3043	코 3053	쓰 3063	비 3073	야 3083

ㄱ : 3050
애 : 3041

(b) 일본어 유니코드

	003	004	005	006	007
0	0 0030	@ 0040	P 0050	` 0060	p 0070
1	1 0031	A 0041	Q 0051	a 0061	q 0071
2	2 0032	B 0042	R 0052	b 0062	r 0072
3	3 0033	C 0043	S 0053	c 0063	s 0073

A : 0041
B : 0042

(c) 영문자 유니코드





문자형

- 문자 자료형 Data를 나타내기 위해서는 하나의 따옴표(' ' : single quotation)를 사용

```
char grade1 = 'A';  
char line = '\n'; // 직접 입력 불가능 문자는 escape s  
char grade2 = '\u0041'; // 'A'와 같은 의미의 유니코드  
char grade2 = '\U0041';  
char years = '2';  
char 한글 = '가'; // 한글도 가능  
char ch2 = '\uac00'; // '가'를 나타낸다
```

참고사항 유니코드

유니코드(unicode)는 전세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이다. 유니코드 협회(unicode consortium)가 제정하며, 현재 최신판은 유니코드 5.0이다. 이 표준에는 문자 집합, 문자 인코딩, 문자 정보 데이터베이스, 문자들을 다루기 위한 알고리즘 등이 포함된다.





문자형



■ JAVA Program에서 사용할 수 있는 escape sequence

Escape Sequence	의미	Unicode
\b	백스페이스 (backspace BS)	0x0008
\t	수평 탭 (horizontal tab HT)	0x0009
\n	줄 바꿈 문자 (line feed LF)	0x000a
\f	새 페이지 문자 (form feed FF)	0x000c
\r	리턴 문자 (carriage return CR)	0x000d
\"	큰따옴표 (double quote “)	0x0022
'	작은따옴표 (single quote ‘)	0x0027
\\	백슬래시 (backslash \)	0x005c
\u8진수	8진수에 해당하는 Unicode 문자. 예) \u8, \u42, \u377	0x0000 ~ 0x00ff



문자형 실습



Filename : CharTypeTest.java

```
public class CharTypeTest {  
    public static void main(String args[]) {  
        char a1 = 'a';           // a 문자 값  
        char a2 = 97;            // a 아스키코드 값  
        char a3 = '\u0061';      // a 유니코드 값  
  
        System.out.println(a1);  
        System.out.println(a2);  
        System.out.println(a3);  
    }  
}
```



문자형 실습



Filename : CharTypeTest1.java

```
public class CharTypeTest1 {  
    public static void main(String args[]) {  
        char ch1 = 'K';  
        char ch2 = '\u004f';  
        char ch3 = 'R';  
        char ch4 = '\u0045';  
        char ch5 = 'A';  
        char 국 = '한', 가 = '국';  
        System.out.println("출력결과 : " + 국 + 가 + " = " +  
            ch1 + ch2 + ch3 + ch4 + ch5);  
    }  
}
```



문자형 실습



Filename : CharTypeTest2.java

```
public class CharTypeTest2 {  
    public static void main(String args[]) {  
        char ch1 = '\\';  
        char 대 = '대', 한 = '한';  
        char ch4 = '\\';  
        char ch5 = '\\t';  
        char 민 = '민', 국 = '국';  
        char ch8 = '\\n';  
        char 만 = '만', 세 = '세';  
        System.out.println("출력결과 :\n" + ch1 + 대 + 한 + ch4 + ch5  
            + 민 + 국 + ch8 + 만 + 세);  
    }  
}
```



문자형 실습



Filename : CharTypeTest3.java

```
public class CharTypeTest3 {  
    public static void main(String args[]) {  
        char test = 25000;  
  
        System.out.println(" 출력결과 : " + test);  
        System.out.printf("출력결과 : %d\n", (short) test);  
        System.out.printf("출력결과 : %c\n", test);  
        System.out.printf("출력결과 : %c\n", (short) test);  
    }  
}
```



Futuristic Innovator

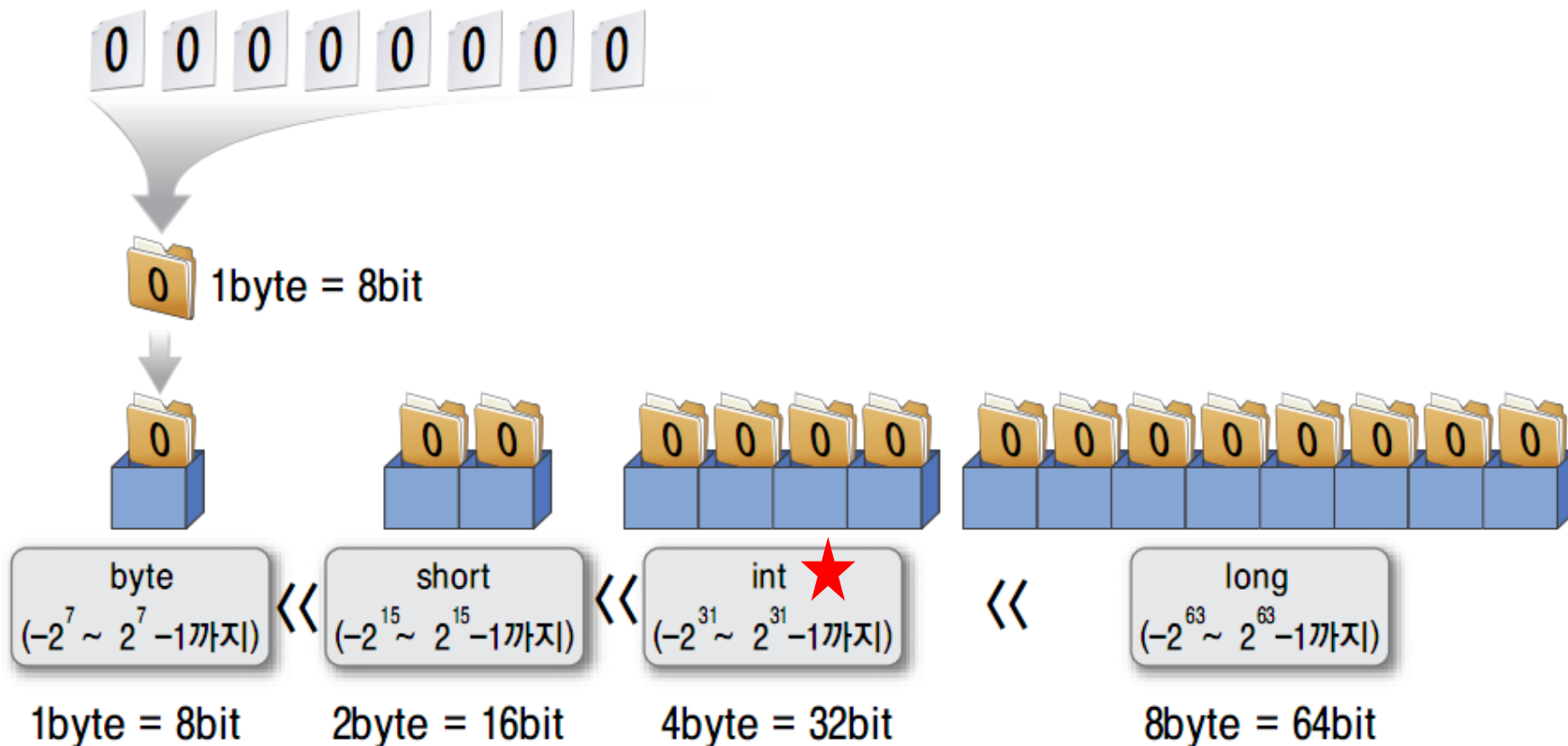
京福大學校
KYUNGBOK UNIVERSITY



integer 형

Integer(정수형)

- 4가지 Data Type을 제공(byte, short, int, long)
- 가장 많이 사용되는 형은 int (묵시적인 정수형은 int형)



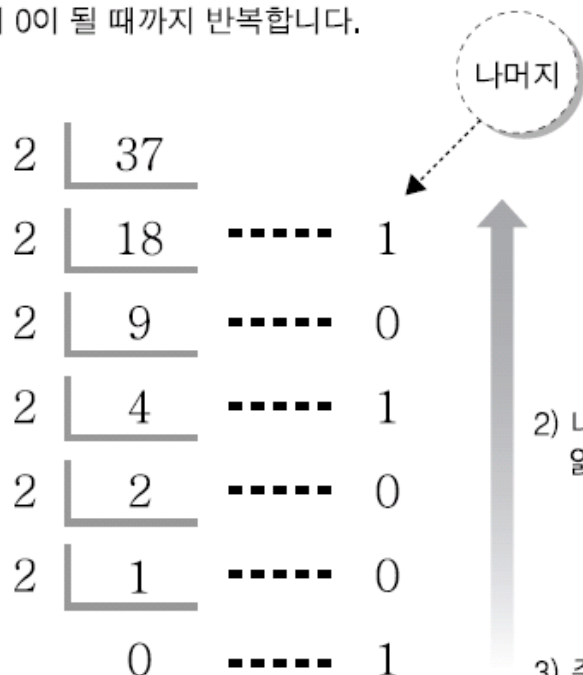


integer 형

■ 정수의 내부 표현

■ 영(零, 0)과 양의 정수는 2진수를 그대로 표현

- 1) 정수를 2로 나눠서 몫은 아래에, 나머지는 오른쪽에 씁니다.
몫이 0이 될 때까지 반복합니다.



- 2) 나머지들을 역순으로
읽으면 구하는 이진수가 됩니다.

100101₂

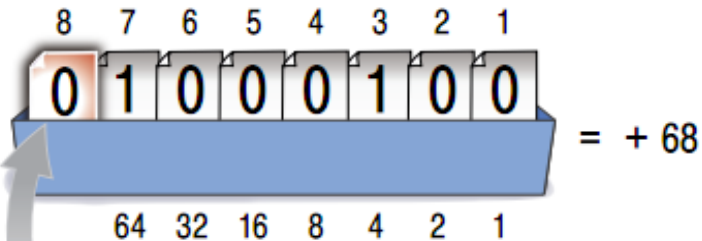
- 3) 주어진 바이트의 마지막 비트부터 이진수로 채우고
남은 비트들은 0으로 채웁니다.

int 타입 00000000 00000000 00000000 00100101

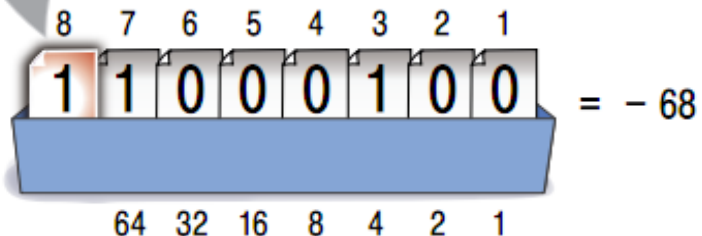


integer 형

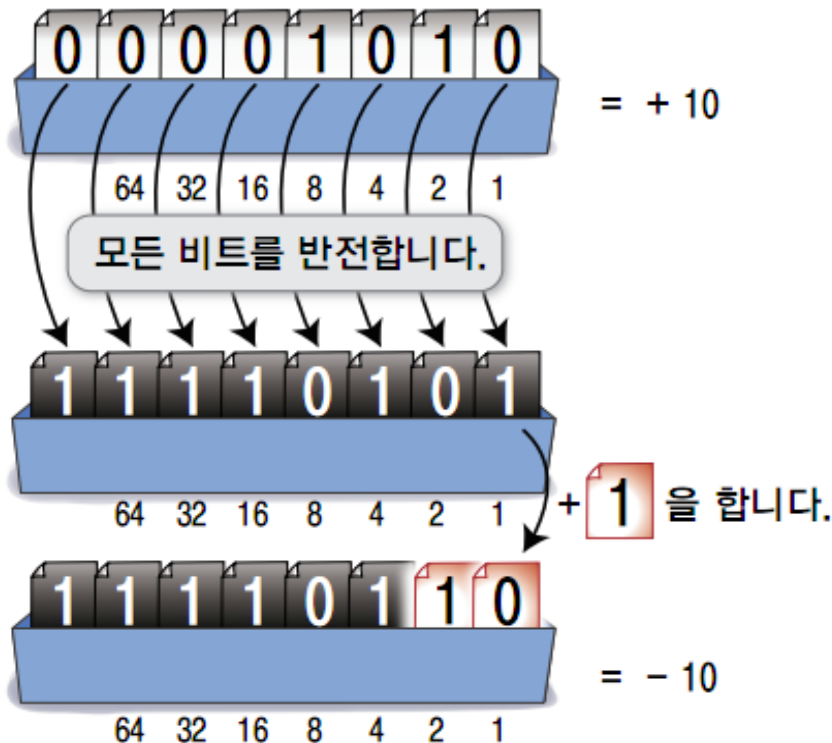
정수의 음수 표현법(2의 보수법)



부호비트가 0이면 양수, 1이면 음수



(a) 부호비트에 의한 음수표기법



(b) 2의 보수를 사용한 음수 표기법



integer 형

■ 2의 보수로 표현된 -5와 7의 덧셈

-5 -----> 11111111111111011

7 -----> **+** 00000000000000111

1000000000000000010 ----->

00000000	00000010
----------	----------

short 타입으로 표현할 수
없는 올림 숫자는 버림



정수(integer) 형

- 정수 데이터를 저장하는 데 사용할 수 있는 변수는 네가지 유형이 있음
- byte(8 bit), short(16 bit), int(32 bit), long(64 bit)
- 부호 없이 양의 정수만을 지원하는 **unsigned 정수형은 지원하지 않음**
- JAVA 언어는 기계 독립성을 높이기 위해서 각 자료형의 크기를 다음과 같이 고정하였음

Type	크기	최소값	최대값
byte	8 bit	-128	127
short	16 bit	-32,768	32,767
int	32 bit	-2^{31}	$2^{31} - 1$
long	64 bit	-2^{63}	$2^{63} - 1$



integer 형



■ byte 형

- byte 예약어 사용
- 1 byte(8 bit) 크기의 기억 장소 할당
- 값의 범위 : $-2^7 \sim 2^7-1$ ($-128 \sim 127$)
- Default 값은 0(zero) 임

//byte의 선언

byte a = -128; //byte의 최저값

byte b = 127; //byte의 최대값



integer 형

■ byte 형

■ 1 Byte를 가지고 표현할 수 있는 정수의 범위

2진수로 표현되는 0과 양의 정수

2의 보수로 표현되는 음의 정수

0	00000000		
1	00000001	----->	11111111 -1
2	00000010	----->	11111110 -2
	⋮		⋮
125	01111101	----->	10000011 -125
126	01111110	----->	10000010 -126
127	01111111	----->	10000001 -127
			10000000 ?



byte 형 실습



Filename : **ByteTypeTest.java**

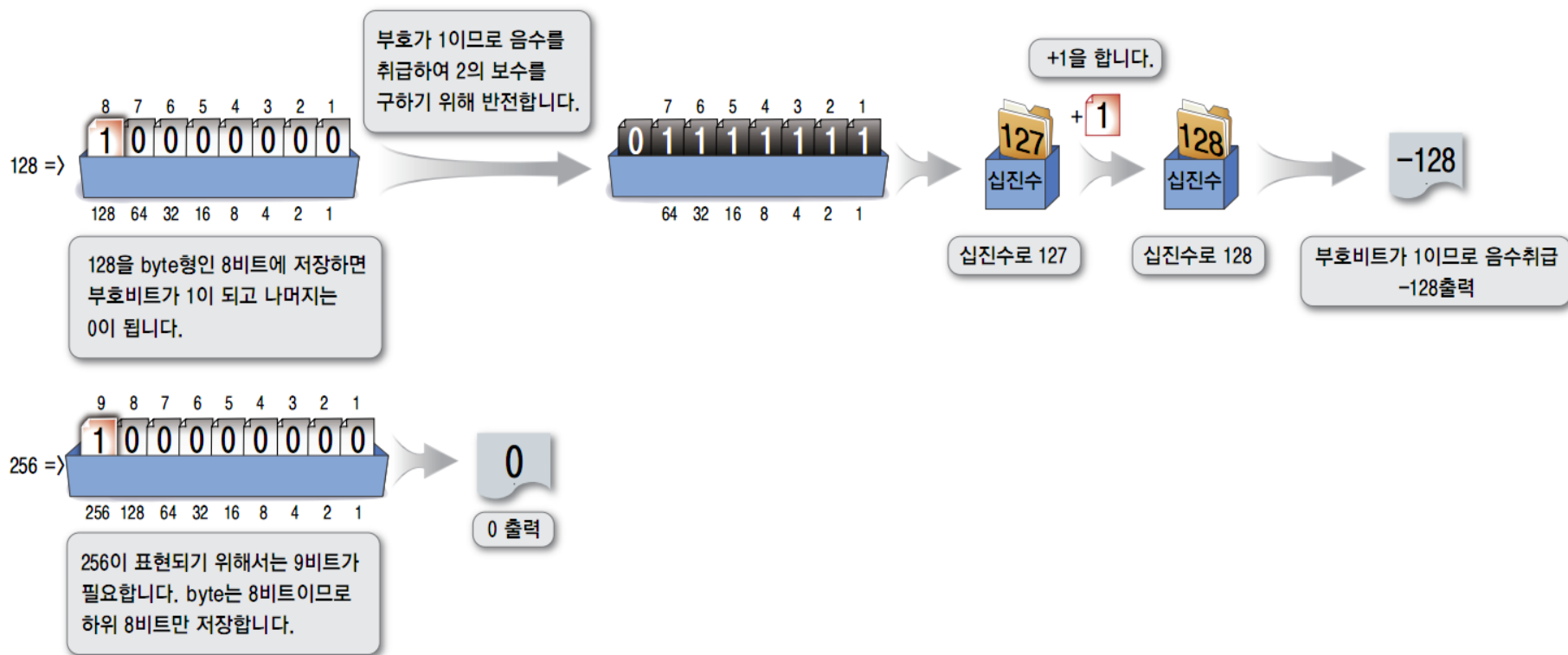
```
public class ByteTypeTest {  
    public static void main(String args[ ]) {  
        byte byte1, byte2, byte3;  
  
        byte1 = 100;  
        byte2 = (byte) 128;  
        byte3 = (byte) 256;  
        System.out.println(" byte1 = " + byte1);  
        System.out.printf(" byte2 = %d\n", byte2);  
        System.out.printf(" byte3 = %d\n", byte3);  
    }  
}
```

127까지 저장 가능한 byte형의
변수에 128을 저장
오류 발생



byte 형 실습

■ byte 변수에 128과 256을 저장한 결과





integer 형

- short 형
 - short 예약어 사용
 - 2 byte(16 bit) 크기의 기억 장소 할당
 - 값의 범위 : $-2^{15} \sim 2^{15} - 1$ ($-32,768 \sim 32,767$)
 - default 값은 0(zero) 임

//short의 선언

short a = -32768; //short의 최저값

short b = 32767; //short의 최대값



short 형 실습



Filename : ByteTypeTest1.java

```
public class ByteTypeTest1 {  
    public static void main(String args[ ]) {  
        short byte1, byte2, byte3;  
  
        byte1 = 100;  
        byte2 = (short) 128;  
        byte3 = (short) 256;  
        System.out.println(" byte1 = " + byte1);  
        System.out.printf(" byte2 = %d\n", byte2);  
        System.out.printf(" byte3 = %d\n", byte3);  
    }  
}
```



integer 형

■ int 형

- int 예약어 사용
- 음수를 포함한 1 word(32 bit) 정수
- 값의 범위 : $-2^{31} \sim 2^{31}-1$ (대략 -21억 ~ 21억)
- Default 값은 0(zero) 임
- 가장 일반적으로 사용되는 정수형은 int
- byte, short, int형을 포함하는 수식을 계산하는 경우 전체식은 먼저 int형으로 전환되어 계산됨
- byte, short형은 연산을 위한 타입이 아니라 저장 공간을 줄이기 위한 Data Type 임

//int의 선언

int a = -2147483648; //int의 최저값

int b = 2147483647; //int의 최대값





integer 형



- 만일 여러분이 사용하려는 값이 100일 경우 반드시 byte 형이나 short 형을 사용하여야 하는가?
- JAVA가 한 번에 처리하는 Data의 기본 단위는 int 형
 - 즉, 기본 처리 단위가 4 byte이기 때문에 byte 형을 사용하거나 int 형을 사용하거나 이득 보는 것은 거의 없음
- JAVA VM은 한 번에 처리할 수 있는 Data의 단위가 32 bit이기 때문에 Program을 최적화하기 위해서는 모든 연산의 기본 단위를 32 bit로 맞추어 주는 것이 좋음



int 형 실습



Filename : IntTypeTest.java

```
public class IntTypeTest {  
    public static void main(String args[]) {  
        int  int1 = 5, int2 = 28;  
        int  int3, int4, int5 ;  
  
        int3 = int2 * int1;  
        int4 = int2 / int1;  
        int5 = 25 / int1;  
  
        System.out.println("28 * 5 = " + int3);  
        System.out.println("28 / 5 = " + int4);  
        System.out.println("25 / 5 = " + int5);  
    }  
}
```



int 형 실습



Filename : IntTypeTest1.java

```
public class IntTypeTest1 {  
    public static void main(String args[]) {  
        int num;  
        num = 10000000000 + 20000000000;    // 30억  
        System.out.println(num);  
    }  
}
```

결과가 왜 30억이 안 될까요 ?



정수형 실습



Filename : NumericTypeTest.java

```
public class NumericTypeTest {  
    public static void main(String args[ ]) {  
        byte a = 127;  
        System.out.println("127을 저장한 byte 값은 " + a);  
        short b = 32767;  
        System.out.println("32767을 저장한 short 값은 : " + b);  
        int c = 2147483647;  
        System.out.println("2147483647을 저장한 int 값은 : " + c);  
        long d = 9223372036854775807L;  
        System.out.println("9223372036854775807을 저장한 long 값은 : " + d);  
    }  
}
```



정수형 실습



Filename : **ByteTestError.java**

```
public class ByteTestError {  
    public static void main(String args[]) {  
        byte value = 128;  
        System.out.println("128을 저장한 byte 값은: " + value);  
    }  
}
```

127까지 저장 가능한 byte형의 변수에 128을 저장
오류 발생



정수형 실습



Filename : CharTypeTest.java

```
public class CharTypeTest {  
    public static void main(String args[]) {  
        char ch1 = 'K';  
        char ch2 = ' \ u004f';  
        char ch3 = 'R';  
        char ch4 = ' \ u0045';  
        char ch5 = 'A';  
        char ch6 = '한', ch7 = '국';  
        System.out.println("출력결과 : "+ch6 + ch7 + "=" + ch1  
            + ch2 + ch3 + ch4 + ch5);  
    }  
}
```



정수형 실습



Filename : CharSpecialType.java

```
public class CharSpecialType {  
    public static void main(String args[]) {  
        char ch1 = '\\';  
        char ch2 = '대', ch3 = '한';  
        char ch4 = '\\';  
        char ch5 = '\\t';  
        char ch6 = '민', ch7 = '국';  
        char ch8 = '\\n';  
        char ch9 = '만', ch10 = '세';  
        System.out.println("출력결과 : \n" + ch1 + ch2 + ch3 +  
            ch4 + ch5 + ch6 + ch7 + ch8 + ch9 + ch10);  
    }  
}
```



integer 형



■ long 형

- $-2^{63} \sim 2^{63} - 1$, 음수를 포함한 64 bit 정수
- long 형은 더 많은 Memory를 필요로 하고 int 형에 비해 속도가 느림
- 보통의 경우에는 int 형을 사용하는 것이 좋음. 즉, 꼭 필요한 경우에만 long을 사용
- 숫자 끝에 **L 또는 l(소문자)**을 붙여주지 않으면 int 형으로 인식해 Compile Error가 발생

long num = 123456789123456789; ---> (X)

long num = 123456789123456789**L**; ---> (O)

long num = 123456789123456789**l**; ---> (O)



long 형 실습



Filename : **LongTest.java**

```
public class LongTest {  
    public static void main(String args[ ]) {  
        long num1 = 123456789123456789;  
        long num2 = 123456789123456789L;  
        long num3 = 123456789123456789l;  
  
        System.out.println(" num1 = " + num1);  
        System.out.println(" num2 = " + num2);  
        System.out.println(" num3 = " + num3);  
    }  
}
```

오류발생



integer 형



- unsigned 형이 없는 JAVA에선 64 bit 이상의 값을 저장하기 위해선 BigInteger를 사용해야 함

```
import java.math.BigInteger;  
BigInteger num = new BigInteger("123456789123456789123456789");
```



실수형



■ Real(실수형)

- 소수점을 갖는 수 표현
- 지수 부분의 유무에 따른 분류
 - 고정 소수점 : 4.567, 0.0045
 - 유동 소수점 : $0.456e03$ (0.456×10^3),
 $2.1e+3f$ (2.1×10^3)
- 정밀도에 따른 분류
 - Single-Precision(단말도)
 - 32 bit이며 **f(F)로 표현** : 3.7f
 - Double-Precision(배밀도)
 - 64 bit이며 **d(D)로 표현(d(D)는 생략 가능)**
 - 3.7d와 3.7은 같은 의미
- 10 진법의 부동소수점이 아니라 2진법의 부동소수점
 - 예) $11_2 \times 2^2$ $1.1_2 \times 2^3$ $0.11_2 \times 2^4$



실수형

■ Real(실수형)

고정소수점 표기 방법

12.375

부동소수점 표기 방법

$$0.012375 \times 10^3$$

$$0.12375 \times 10^2$$

$$1.2375 \times 10^1$$

$$12.375 \times 10^0$$

$$123.75 \times 10^{-1}$$

$$1237.5 \times 10^{-2}$$

$$12375. \times 10^{-3}$$

모두 12.375의 부동
소수점 표기입니다.



실수형

10진 소수를 2진 소수로 바꾸는 방법

12.375

1) 10진수의 정수부에는 정수의 2진수 변환 방법을 그대로 적용합니다.

2) 10진수의 소수부에는 계속 2를 곱하여 소수점 위로 올라온 값을 순서대로 떼어내면 됩니다.

$$\begin{array}{r} 2 \overline{) 12} \\ 2 \overline{) 6} \text{ ----- } 0 \\ 2 \overline{) 3} \text{ ----- } 0 \\ 2 \overline{) 1} \text{ ----- } 1 \\ 2 \overline{) 0} \text{ ----- } 1 \end{array}$$

$$\begin{array}{r} 0.375 \\ \times 2 \\ \hline 0.75 \\ \times 2 \\ \hline 1.5 \\ \times 2 \\ \hline 1.0 \end{array}$$

소수부가 0이 될 때까지 반복합니다.

1100.011₂



실수형

■ 2진수의 고정소수점과 부동소수점

고정소수점 표기 방법

1100.011_2

선택되는 표기

부동소수점 표기 방법

$$0.1100011_2 \times 2^4$$

$$1.100011_2 \times 2^3$$

$$11.00011_2 \times 2^2$$

$$110.0011_2 \times 2^1$$

$$1100.011_2 \times 2^0$$

$$11000.11_2 \times 2^{-1}$$

$$110001.1_2 \times 2^{-2}$$

$$1100011_2 \times 2^{-3}$$

$$11000110_2 \times 2^{-4}$$

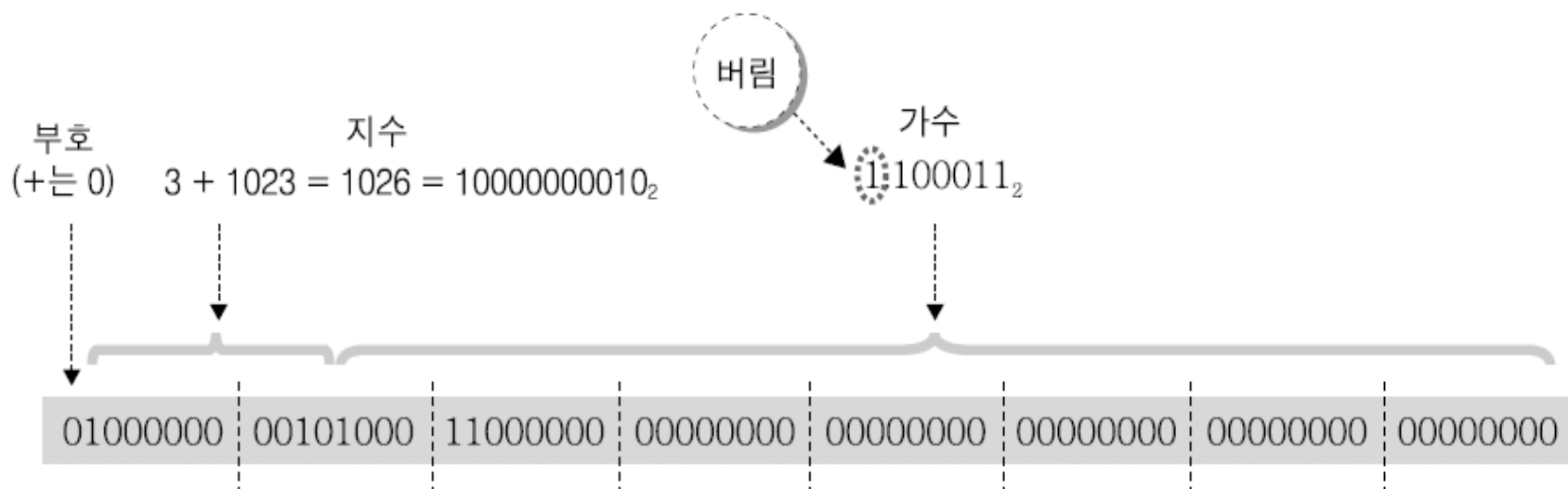
모두 1100.011_2 의
부동소수점 표기입니다.

이 중 어떤 표기를 택해야 할까요?



실수형

■ $1.100011_2 \times 2^3$ 을 double 타입으로 표현한 예



일반 표기법	과학적 표기법	지수 표기법
146.91	1.4691×10^2	1.4691E+2
0.00081	8.1×10^{-4}	8.1E-4
1800000	1.8×10^6	1.8E+6



실수형



■ IEEE 754 표준

- 부호와 크기 표기방식으로 음수를 표시하며,
- 양의 영(+0, 즉 양의 0)과 음의 영(-0, 음의 0)
- 양의 무한대와 음의 무한대
- 특별한 'Not-a-Number' (보통 'NaN'이라 씀)
- JAVA에서 float와 double은 각각 4 bytes, 8 bytes이며, 아래처럼 정보를 나누어 저장

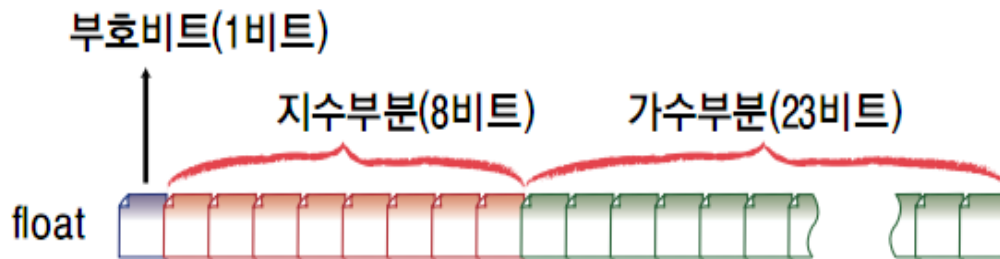
	전체(bits)	부호(bits)	지수(bits)	가수(bits)
float	32(4 bytes)	1	8	23
double	64(8 bytes)	1	11	52



실수형

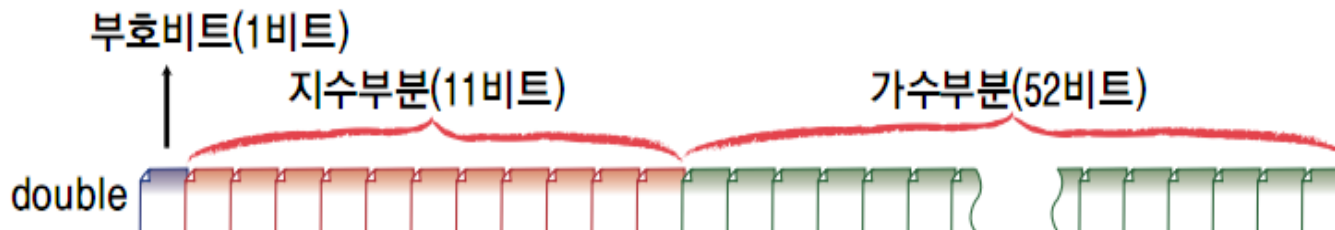
■ Real(실수형)

- 부호와 지수(exponential)부분, 가수(mantissa)부분으로 구성(IEEE 표준 방식)
- 저장할 수 있는 크기에 따라 float 형과 double 형으로 구분
- Default Data Type은 double 형



저장가능범위

1.4E - 45 ~ 3.4028235E38



4.9E - 324 ~

1.7976931348623157E308



실수형



■ float

- 소수점을 갖는 수를 표현하며 32 bit에 표현
- JAVA에서 실수형은 Default가 double이므로 float 형은 값을 대입할 때에는 3.14f 와 같이 'f'나 'F' 접미사를 꼭 붙여 주어야 Compile Error가 발생하지 않음
- float 형의 유효 자리 수가 7자리 임
- 표현 가능한 값의 범위

1.40239846E-45f ~ (표현 가능 양수 범위)
3.40282347E+38f

- Default 값은 0.0f 임



실수형



■ double

- 소수점을 갖는 수를 표현하며 64 bit에 표현
- float형보다 정밀도 2배이고, 'f'가 붙지 않는 실수형
- double 자료는 float 변수에 치환 불가(demotion), 역은 가능(promotion)
- 대부분의 경우에는 double을 사용하는 것이 바람직함
- double 형의 유효 자리 수가 15 ~ 16자리 임
- 표현 가능한 값의 범위

4.94065645841246544E-324

~ (표현 가능 양수 범위)

1.79769313486231570E+308

- Default 값은 0.0 임



실수형



- `float pi = 3.14;`
- `float pi = 3.14f;`
- `double pi = 3.14;`

- 다음 문장이 Error가 나는 이유는?

```
float temperature = 25.6; //25.6 double형이므로 오류
```

```
float temperature = 25.6f; //OK
```

- JDK 7부터 실수형 상수에도 밑줄 기호를 사용할 수 있음

```
Double number = 123_456_789.0; // 밑줄 기호 사용 가능
```



실수형 실습



Filename : FloatDoubleTest.java

```
public class FloatDoubleTest {  
    public static void main(String args[]) {  
        float a = 0.12345678901234567890f;  
        double b = 0.12345678901234567890;  
        System.out.println("float 변수 a의 값은 : " + a);  
        System.out.println("double 변수 b의 값은 : " + b);  
        float c = 1.0f / 3.0f;  
        double d = 1.0 / 3.0;  
        System.out.println("float 변수 c의 값은 : " + c);  
        System.out.println("double 변수 d의 값은 : " + d);  
    }  
}
```




실수형 실습



Filename : FloatDoubleTest1.java

```
public class FloatDoubleTest1 {  
    public static void main(String args[]) {  
        double num;  
        num = 3.14 + 1;  
        System.out.println(num);  
        System.out.printf("%.2f", num);  
    }  
}
```



특수한 실수값



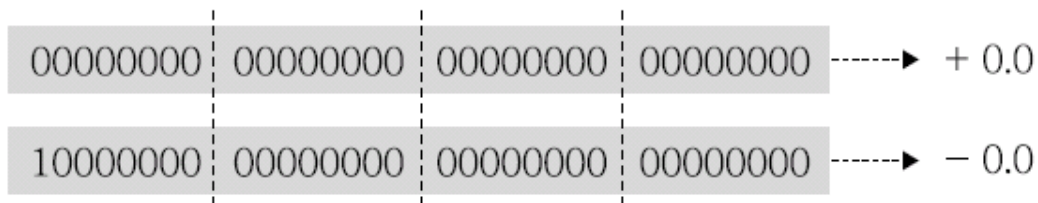
- 양의 무한대(positive infinity) : 오버플로우
- 음의 무한대(negative infinity) : 언더플로우
- NaN(Not a Number) : 유효하지 않은 연산



특수한 실수값

■ 실수형의 +0과 -0의 표현

a) float 타입의 0을 표현하는 값



b) double 타입의 0을 표현하는 값





특수한 실수값

- 실수형 수를 0으로 나누는 프로그램

Filename : **FloatDoubleTest.java**

```
public class TypeExample2 {  
    public static void main(String args[]) {  
        double result1, result2;  
        result1 = 2.0 / 0.0;  
        result2 = 2.0 / -0.0;  
        System.out.println("result1 = " + result1);  
        System.out.println("result2 = " + result2);  
    }  
}
```

```
명령 프롬프트  
E:\work\chap3\3-2-2>java TypeExample2  
result1 = Infinity  
result2 = -Infinity  
E:\work\chap3\3-2-2>
```



특수한 실수값



■ 부동소수점 타입의 NaN(Not a Number) 표현

a) float 타입의 NaN 표현

01111111	11000000	00000000	00000000
----------	----------	----------	----------

b) double 타입의 NaN 표현

01111111	11111000	00000000	00000000	00000000	00000000	00000000	00000000
----------	----------	----------	----------	----------	----------	----------	----------



특수한 실수값



■ JDK Library에 있는 부동소수점 타입 관련 상수들

의미	상수 이름	
	float 타입 상수	double 타입 상수
표현 가능한 최대값, 최소값의 절대치	Float.MAX_VALUE	Double.MAX_VALUE
표현 가능한 가장 미세한 값의 절대치	Float.MIN_VALUE	Double.MIN_VALUE
표현 범위를 넘어서는 양의 값	Float.POSITIVE_INFINITY	Double.POSITIVE_INFINITY
표현 범위를 넘어서는 음의 값	Float.NEGATIVE_INFINITY	Double.NEGATIVE_INFINITY
NaN (Not a Number)	Float.NaN	Double.NaN



참조형



■ 기본형과 참조형

■ byte, short, int, long, float, double, boolean, char

유형	분류	실제자료 유형
기본 자료 유형	정수형	byte, short, int, long
	부동소수형	float, double
	논리형	boolean
	문자형	char
참조 자료 유형	클래스	String, System, Math, Integer, Boolean, Character, ...
	인터페이스	Collection, Enumeration, Set, Iterator, List, Queue, ...
	배열	byte[], short[], int[], long[], ... String[] ...



참조형

■ 기본형과 참조형의 차이

```
int m = 3;
```

기본 유형 변수 : m 선언 유형의 자료 값 : 3

```
String s = "Java";
```

참조 유형 변수 : s 선언 유형의 자료(객체)가 있는 주소 값 : 1024

주소 : 1024

"Java"

[그림 7.1]

기본 유형과 참조 유형



참조형



- 객체를 가리키는 형
 - 클래스형
 - 클래스 이름
 - 객체를 가리키는 참조형
 - String, System, Math, Integer, Boolean, ...
 - 배열
 - 같은 형의 여러 값을 저장하는데 이용하는 자료형
 - C/C++와 달리 객체로 취급
 - Byte[], short[], int[], long[], String[], ..
 - 인터페이스형
 - 인터페이스 이름
 - 인터페이스를 구현한 객체에 대한 참조
 - Collection, Enumeration, Set, Iterator, List, ...
 - 열거형
 - 여러 개의 숫자 상수만을 가진 특별한 형태의 클래스형
 - Enumeration



참조형



- 참조형 변수 선언
 - `int intPri;`
 - `Integer intRef;`
 - 참조형 변수로 클래스형의 Integer형 선언
- 참조형 변수 초기값 부여
 - `Integer intRef = new Integer(7);`
 - 변수 intRef는 클래스 Integer로 부터 하나의 객체를 참조하는 변수로 다음과 같은 문법에 따라 정수를 저장
 - `Integer 참조변수이름 = new Integer(저장할 정수);`
 - 참조형의 변수에 새로운 객체를 하나 만들어 저장하려면 키워드 `new`를 이용
- 참조형 변수 값 출력
 - `System.out.println("intPri = " + intPri);`
 - `System.out.println("intRef = " + intRef);`
 - intRef는 `intRef.toString()` 메소드의 반환 값을 출력



참조형 실습



Filename : **ReferenceType.java**

```
public class ReferenceType {  
    public static void main(String[] args) {  
        int intPri;  
        Integer intRef;  
        intPri = 7;  
        intRef = new Integer(7);  
        System.out.println("intPri = " + intPri);  
        System.out.println("intRef = " + intRef);  
        intPri = intPri + intRef.intValue();  
        System.out.println("intPr + intRef = " + intPri);  
    }  
}
```



참조형



- 기본형과 참조형 정수 더하기
 - `intPri = intPri + intRef.intValue();`
 - `System.out.println("intPr + intRef = " + intPri);`
 - 기본형과 참조형은 자료형이 호환되지 않으므로 두 수를 그대로 더할 수가 없다
 - 참조형에서 내부에 저장된 정수 값을 반환하는 `intValue()`라는 메소드를 호출하여 더할 수 있음



참조형

클래스 Integer 의 메소드와 와 필드

```
1  /*  
2  * @(#)Integer.java 1.90 04/05/11  
3  * Copyright 2004 Sun Microsystems, Inc. All rights reserved.  
4  * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.  
5  */  
6  
7  package java.lang;  
8  
9  /**  
10  * The <code>Integer</code> class wraps a value of the primitive type  
11  * <code>int</code> in an object. An object of type  
12  * <code>Integer</code> contains a single field whose type is  
13  * <code>int</code>.  
14  *  
15  * <p>  
16  * In addition, this class provides several methods for converting an  
17  * <code>int</code> to a <code>String</code> and a <code>String</code>  
18  * to an <code>int</code>, as well as other constants and methods  
19  * useful when dealing with an <code>int</code>.  
20  *  
21  * <p>Implementation note: The implementations of the "bit twiddling"  
22  * methods (such as {@link #highestOneBit(int) highestOneBit} and  
23  * {@link #numberOfTrailingZeros(int) numberOfTrailingZeros}) are  
24  * based on material from Henry S. Warren, Jr.'s <i>Hacker's  
25  * Delight</i>, (Addison Wesley, 2002).  
26  *  
27  * @author Lee Boynton  
28  * @author Arthur van Hoff  
29  * @author Josh Bloch  
30  * @version 1.90, 05/11/04  
31  * @since JDK1.0  
32  */  
33  
34  public final class Integer extends Number implements Comparable  
35  {  
36  /**  
37  * A constant holding the minimum value an <code>int</code> can  
38  * have, -231.  
39  */  
40  public static final int MIN_VALUE = 0x80000000;  
41  
42  /**
```

- Integer
- MIN_VALUE : int
- MAX_VALUE : int
- TYPE : java.lang.Class
- digits : char[]
- DigitTens : char[]
- DigitOnes : char[]
- sizeTable : int[]
- value : int
- SIZE : int
- serialVersionUID : long
- toString(int, int)
- toHexString(int)
- toOctalString(int)
- toBinaryString(int)
- toUnsignedString(int, int)
- toString(int)
- getChars(int, int, char[])
- getString(int)
- parseInt(String, int)
- parseInt(String)
- valueOf(String, int)
- valueOf(String)
- valueOf(int)
- Integer(int)
- Integer(String)
- byteValue()
- shortValue()
- intValue()
- longValue()
- floatValue()
- doubleValue()
- toString()
- hashCode()
- equals(Object)
- getInteger(String)
- getInteger(String, int)
- getInteger(String, Integer)
- decode(String)
- compareTo(Integer)
- highestOneBit(int)
- lowestOneBit(int)
- numberOfLeadingZeros(int)
- numberOfTrailingZeros(int)
- bitCount(int)
- rotateLeft(int, int)
- rotateRight(int, int)
- reverse(int)
- signum(int)
- reverseBytes(int)
- compareTo(Object)
- IntegerCache



String



- JAVA에서 문자열은 기본형이 아니라 참조형이고 이중 따옴표를 이용하여 표현

```
"Happy Java"  
"a" , ""  
"123"
```

- JAVA에서는 문자열을 처리할 수 있는 String 클래스를 제공
- 문자열 변수를 선언/초기화하는 것은 String은 클래스이지만 기본형의 변수를 선언하고 초기화하는 방법과 동일

```
String name;           // 문자열 변수 선언  
name = "홍길동";       // 문자열 초기화
```

```
String title = "선생님"; // 선언과 동시에 초기화  
String title = new String("선생님");
```



String



- 다음과 같이 화면에 출력할 수도 있음

```
System.out.println(name);  
// name 문자열의 내용이 화면에 출력
```

- 두 개의 문자열을 서로 연결하는 데 ‘+’ 연산자를 이용

```
String fullName = name + title;    // 두 문자열 연결  
String sayHi = “안녕하세요.” + name;  
String sayHello = “안녕하세요.” + name + title;  
//세 개 이상도 연결 가능
```



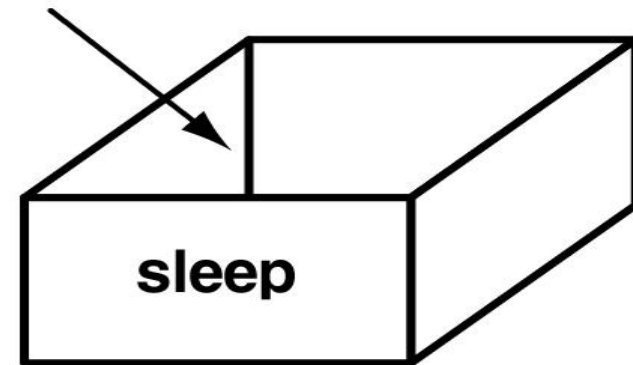
String



■ 문자열 연결(+연산자)

String sleep = “돌돌이” + “는 어젯밤에 ” + 15 + “시간”;

돌돌이 + 는 어젯밤에 + 15 + 시간



String



String



- 다음과 같이 System.out.println() 메소드에 문자열 연결식을 직접 입력할 수도 있음

```
System.out.println("안녕하세요. "+name+"씨.");
```

- 이렇게 하면 불필요한 문자열 변수를 생성하지 않고 직접 연결된 문자열을 화면에 출력할 수 있음
- 어떤 기본형 변수의 값을 출력할 때에도 System.out.println() 메소드를 사용할 수 있음

```
double value = 11.2;  
System.out.println(value);
```

- 메소드 System.out.println()는 문자열을 인자로 주어야 하는데 이경우 변수 값 value는 내부적으로 문자열로 자동으로 변환되어서 System.out.println()에 넘겨줌



String



```
double value = 11.2;
```

```
System.out.println("value 값은 " + value + "입니다.");
```

- 이 경우 연결이 일어나기 전에 value 값은 문자열로 변환되어 세 개의 문자열을 연결한 새로운 문자열이 System.out.println()에 인자로 넘어가는 것



문자열 실습



Filename : Stringtest1.java

```
public class Stringtest1 {  
    public static void main(String[] args) {  
        char a[] = {'C','o','m','p','u','t','e','r'};  
        String s1 = new String(a); //문자배열로부터 문자열 객체 s1 생성  
        //문자 배열로부터 부분 문자열을 추출하여 객체s2생성  
        String s2 = new String(a, 3, 2);  
        //문자열로부터 직접 객체 생성  
        String s3 = new String("String 객체 테스트");  
        System.out.println("문자 배열로부터 생성된 String 객체 s1 : " + s1);  
        System.out.println("문자 배열로부터 추출된 String 객체 s2 : " + s2);  
        System.out.println("문자열로부터 직접 생성된 String 객체 s3 : " + s3);  
        System.out.println("문자열 객체 s1의 길이 : " + s1.length());  
    }  
}
```



문자열 실습



Filename : Stringtest1.java

// byte배열을 이용하여 문자열을 생성하는 예제 프로그램

```
public class Stringtest2 {
    public static void main(String[] args) {
        byte a[] = {74,65,86,65,89,79}; //바이트 배열 a생성
        System.out.println("초기 바이트 배열:{74,65,86,65,89,79}");
        char c[] = new char[6];
        for (int i = 0 ; i < 6 ; i++)
        {
            c[i] = (char)a[i];
            System.out.println("정수"+a[i]+"에 해당하는 아스키코드 문자는"
                + c[i]);
        }
        //바이트 배열 a를 문자열로 변환하여 스트링 객체 a1생성
        String a1 = new String(a,0);
        //부분 문자열 객체 a2생성
        String a2 = new String(a,0,2,4);
        System.out.println("byte형의 배열 a를 문자열로 변환한 String 객체 a1:"+a1);
        System.out.println("두번째 요소(V)에서 부터 3 문자 추출 :"+ a2);
    }
}
```