Java의 정석

Chapter 01~02

자바에 대하여

자바(Java)란?

- 썬 마이크로 시스템즈가 개발하고
- 2010년 오라클이 인수한 프로그래밍 언어

<특징>
- 객체지향 언어이다.(ch12 때 발표)
- 운영체제에 독립적이다.
- 가비지 컬렉터가 메모리를 관리해준다.(ch11 때 발표)
- 멀티쓰레드를 지원한다.(ch13 때 발표)

OOP(객체지향프로그래밍) 4가지 특징

1. 캡슐화
2. 추상화
3. 상속/일반화 관계
4. 다형성

해당 내용들은 클래스/객체, 상속 등의 개념을 알아야 하므로
추후에 해당 단원들을 공부한 후 구체적으로 설명하도록 하겠음.

https://gmlwjd9405.github.io/2018/07/05/oop-features.html

https://velog.io/@ygh7687/OOP%EC%9D%98-5%EC%9B%90%EC%B9%99%EA%B3%BC-4%EA%B0%80%EC%A7%80-%ED%8A%B9%EC%84%B1

운영체제 독립성(JVM)

그림 1-2 C++의 프로그래밍 방식

운영체제 독립성(JVM)

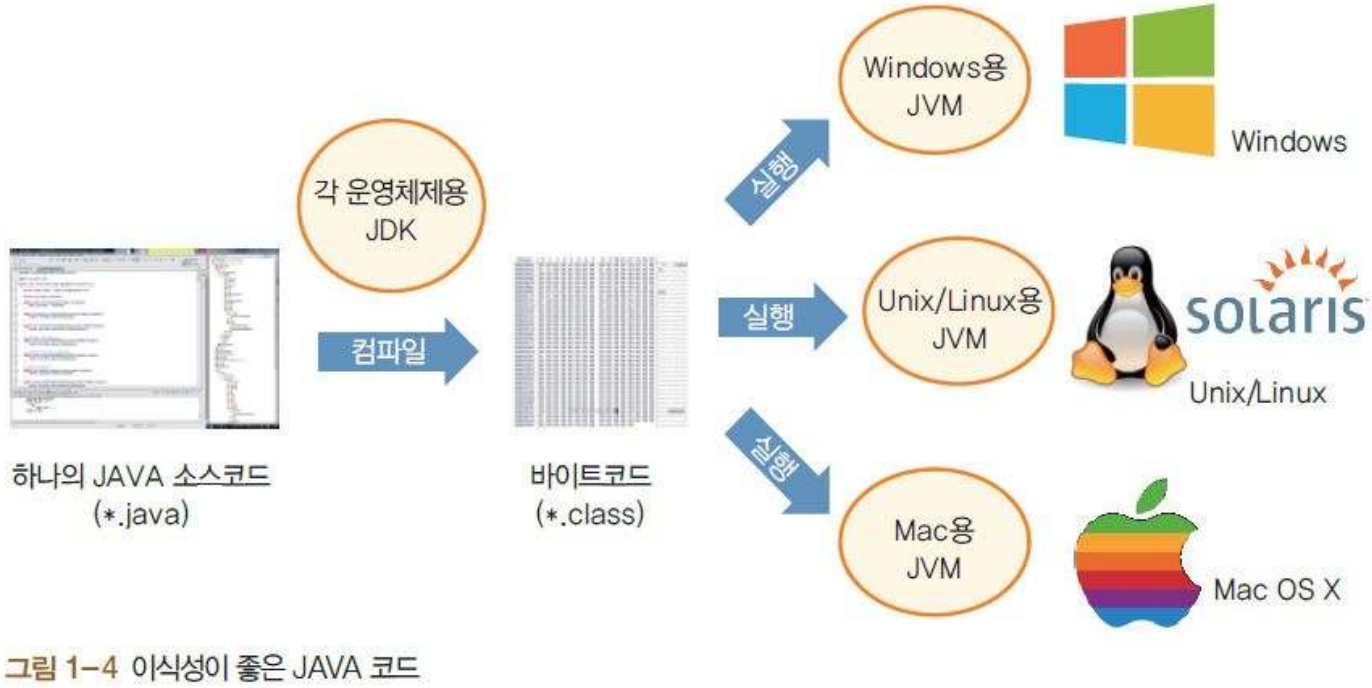


그림 1-4 이식성이 좋은 JAVA 코드

- 메모장이나 이클립스(Eclipse) 등의 텍스트 에디터로 작성, 작성된 소스코드를 **javac.exe**로 컴파일하 면 바이트코드가 생성(바이트코드는 확장명이 *.class인데 사람은 이 파일의 내용을 읽을 수 없음)
- 바이트코드는 모든 운영체제에서 실행이 가능하지만, 단 운영체제에 JAVA 가상 머신인 **JVM(Java Virtual Machine)**이 미리 설치되어 있어야 한다. JVM은 오라클에서 무료로 배포하고 있다.

운영체제 독립성(JVM)

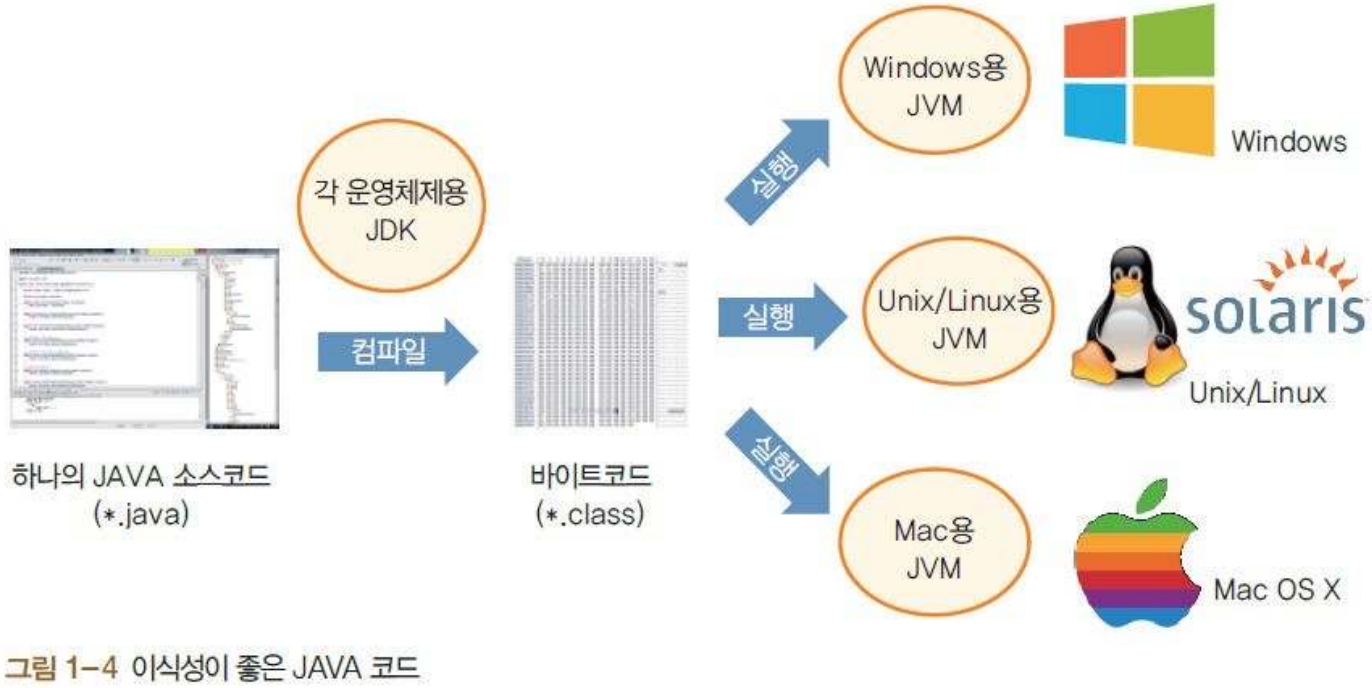


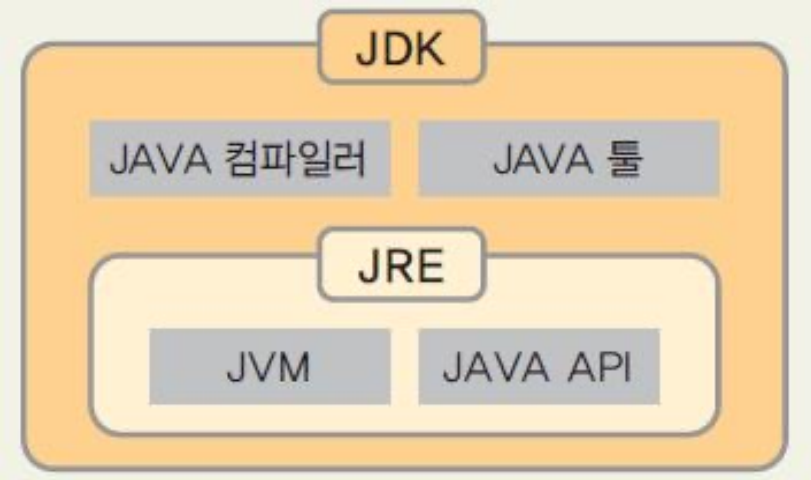
그림 1-4 이식성이 좋은 JAVA 코드

• 실습을 위해 패키지(최상위 폴더)를 만들어보면 밑에 src라는 폴더가 생긴다. 또한, src폴더 내에 자바 파일을 만들고 실행시키면 bin 폴더 안에 *.class 파일이 생긴다.
• 즉, 자바 파일(소스코드)는 **src(=source)** 폴더에 만들어야 하고, 이를 실행하면 **bin(=binary)** 폴더에 클래스 파일(바이트코드)이 만들어진다

운영체제 독립성(JVM)

■ JDK와 JRE

- JAVA 개발 도구인 JDK와 JAVA 실행 환경인 JRE를 명확한 구분
- JDK : JAVA 프로그램을 작성하고 컴파일하는데 필요
- JRE : JAVA를 실행하는 데 필요하다.
- JDK에는 JRE가 포함(JAVA 개발 없이 컴파일 결과(*.class)만 실행하려면 JRE만 설치)



컴파일러 vs 인터프리터

구분	컴파일러	인터프리터
작동 방식	소스코드를 기계어로 먼저 번역하고, 해당 플랫폼에 최적화되어 프로그램을 실행함	별도의 번역 과정 없이 소스코드를 실행 시점에 해석하여 컴퓨터가 처리할 수 있도록 함
장점	실행 속도가 빠름	간단히 작성, 메모리가 적게 필요
단점	한 번에 많은 기억 장소가 필요함	실행 속도가 느림
주요 언어	C, 자바(Java), C++, C#	파이썬, 스칼라

변수(Variable)

❑ 변수란, 단 하나의 값을 저장할 수 있는 **메모리 공간**이다.

- 대소문자 구분되고, 길이제한이 없다
- 예약어를 사용하면 안 된다.(예약어를 작성하면 글씨색깔과 두께가 바뀌어 알 수 있음)
- 숫자로 시작하면 안 된다
- 특수문자는 '_'를 가장 많이 사용한다.
- 클래스 이름의 경우, 첫 글자는 대문자로 한다. (ex. class Car)
- 상수의 이름은 모두 대문자로 한다 (ex. MAX_NUMBER)

❑ 변수의 초기화란, 변수를 사용하기 전에 **처음으로 값을 저장**하는 것

- 특히나 for문을 사용할 때 등의 경우 변수를 초기화 해주는 것이 중요하다.

변수의 타입

❑ 변수 타입에는 기본형과 참조형이 있다.

❑ 기본형

- 논리형: boolean
- 문자형: char
- 정수형: byte, short, int, long
- 실수형: float, double

메모리의 개념 또한 추후 다루게 될 "=="와 ".equals()"의 차이에서 자세히 알게 될 것이다.

https://namocom.tistory.com/869

https://gaeko-security-hack.tistory.com/160

❑ 참조형

- 기본형과 달리, 실제 값이 아닌 메모리 주소로 저장한다.
- new를 붙여서 만드는 애들을 참조형이라고 쉽게 생각하자
- Car car = new Car();
- Null의 경우 어떤 객체 주소도 저장되지 않음을 뜻한다. (나중에 다룰 가비지컬렉터와 연관/메모리관리)

래퍼 클래스(Wrapper Class)

❑ 기본형(원시 type)과 비슷한 형태로 래퍼 클래스라는 것이 있다.(Chapter 8에서 다뤄짐)

- 래퍼클래스는 간단히 설명하면, 기본 타입들을 객체화 한 것이다.

❑ 그렇다면 래퍼 클래스를 왜 쓰는가?

- 가장 큰 이유는 객체로써의 특징을 활용하기 위함이다.
- 예를 들어, 기본형인 int의 경우 무조건 정수형이 들어가야 하지만, 래퍼의 경우 빈 값으로 null을 줄 수 있다.(null은 어떤 객체의 주소도 저장하지 않음을 의미함)
- 또한, 향후 배울 제네릭의 특성을 쓸 수 있으며 Object 클래스(타입)을 받았을 때 다룰기에도 용이하다(Chapter 8에서 자세히 다룰 예정)

기본 자료형 심화(1): 문자형

❑ 문자형 'char'는 문자를 내부적으로 정수(유니코드)로 저장하기에 int와 연산이 가능하다.

- 단, 이 경우 유니코드 테이블에서 읽은 대응되는 숫자로 연산된다.

```
16
17          char a = '5';
18          int b = 3;
19          System.out.println(a+b);
20
21
22
23      }
24
25  }
```

Problems @ Javadoc Declaration Console

<terminated> ex_0628 [Java Application] /Library/Java/JavaVirtualMachines/jd

56

❑ 문자 '5'는 유니코드에 대응되는 정수 상으로 53에 해당한다.

❑ 따라서 결과는 53+3 =56이 나온다

기본 자료형 심화(1): 문자형 - 인코딩

❑ **인코딩:** 문자를 어떻게 출력할지에 대한 약속

- ASCII -> 유니코드 -> UTF-8 순서로 발전

❑ **ASCII:** 128개 문자 조합을 제공하는 7비트 부호

- 각 국의 언어를 표현하기에 부족하다는 단점으로 유니코드 개발됨.

❑ **유니코드:** 각 나라별 언어를 모두 표현하기 위해 나온 코드 체계

- 16비트(2바이트)를 표현하므로 최대 65,536자 표현 가능

❑ **UTF-8:** 유니코드를 사용하는 **인코딩** 방식 중 하나

- 가변길이 인코딩
- 1~4바이트 사이로 인코딩이 가능하여 유니코드보다 효율적
- Java는 UTF-16을 사용

기본 자료형 심화(2): 실수형

- ❑ 부동소수점과 고정소수점 차이에 대한 이해 (뒤에서 추가로 설명할 것)
- ❑ 자세한 내용은 링크 참고 https://blog.naver.com/passionisall/222139065323

Ⅲ. 금융업에서 부동소수점을 기피하는 이유

앞서 설명한 내용들과 투자시장의 자산들의 특징을 비교해보면 **금융업에서 부동소수점을 사용하는 것은 큰 문제가 발생**할 수 있다는 점을 유추해볼 수 있습니다. 이러한 문제는 자리수가 엄청 크거나 무한대로 순환하는 실수에서 발생합니다. 부동소수점의 연산 방법은 소수점의 위치를 고정하지 않고 근사값으로 표현하므로 오차가 발생하게 됩니다. (오차는 10진수를 2진 실수로 정확히 변환할 수 없기 때문에 발생함)

사례1: 0.1을 100번 더해도 정확히 10이 나오지 않는다.

사례2: 부동소수점 방식으로 Token Decimal이 18자리인 EOS 0.3개를 전송할 경우 0.000012207031249988 9 EOS가 증발하는 일이 발생한다.

코인시장의 경우 1개 이하의 단위로도 코인을 거래할 수 있습니다. 한편, 이러한 가격 parameter 등을 Double 혹은 Float로 받을 경우 **사례2**와 같은 문제가 발생할 수 있습니다. 국내의 일반적인 주식의 경우 원화의 거래 단위를 고려해볼 때 소수점을 다루는 일은 거의 없겠지만, 선물·옵션과 같이 지수로 표시되는 시장 혹은 미국 시장(cent는 소수점)의 경우에도 부동소수점을 사용할 경우 문제가 발생할 수 있습니다.

Ⅳ. 부동소수점 문제 해결방안

1. 자바

자바의 경우 **java.math.BigDecimal클래스**의 **BigDecimal**을 통해 이러한 문제를 해결할 수 있습니다. 실수형 중에서 가장 큰 double형의 정밀도가 15자리 밖에 안 되는 반면 BigDecimal의 경우 32bit의 소수점의 크기를 갖습니다. 또한, BigDecimal은 [정수*1-^(-scale)] 형태로 값을 10진수로 받는다는 특징이 있습니다.

기본 자료형 심화(2): 실수형

```
float a = 10.000;
float b = 3.000;

a+b;
결과: 13.0000000199999...

a-b;
결과: 6.9999999999....
```

기본 자료형 심화(2): 실수형

```
BigDecimal a = new BigDecimal("10");
BigDecimal a = new BigDecimal("3");

a.add(b);
= 13

a.subtract(b)
= 7

a.multiply(b);
= 30

a.divide(b);
= 3.3333...
```

기본 자료형 심화(3): 크기

- ❑ 리터럴 타입(우변)은 저장될 변수의 타입과 일치하는 것이 일반적이다
- ❑ 하지만, 넓은 타입에 작은 타입 값을 넣을 수 있다.
- ❑ Int에 'A'등의 문자를 넣으면 유니코드 값으로 들어간다.

- ❑ **일반적:** float a = 3.14f;
- ❑ **허용:** double a = 3.14f;
- ❑ int i = 'A'; (유니코드 값 65 저장)

기본 자료형 심화(3): 크기

종류	데이터형	크기(byte / bit)	표현 범위
논리형	boolean	1 / 8	true 또는 false
문자형	char	2 / 16	'\u0000' ~ 'uFFFF' (16비트 유니코드 문자 데이터)
정수형	byte	1 / 8	-128 ~ 127
	short	2 / 16	-32768 ~ 32767
	int	4 / 32	-2147483648 ~ 2147483647
	long	8 / 64	-9223372036854775808 ~ 9223372036854775807
실수형	float	4 / 32	1.4E-45 ~ 3.4028235E38
	double	8 / 64	4.9E-324 ~ 1.7976931348623157E308

10진법 변환하기

❑ **N진법 수를 10진법으로 변환하기**

- 각 자리수에 N^k를 곱해나간다.

Ex) $1011.101_{(2)}$

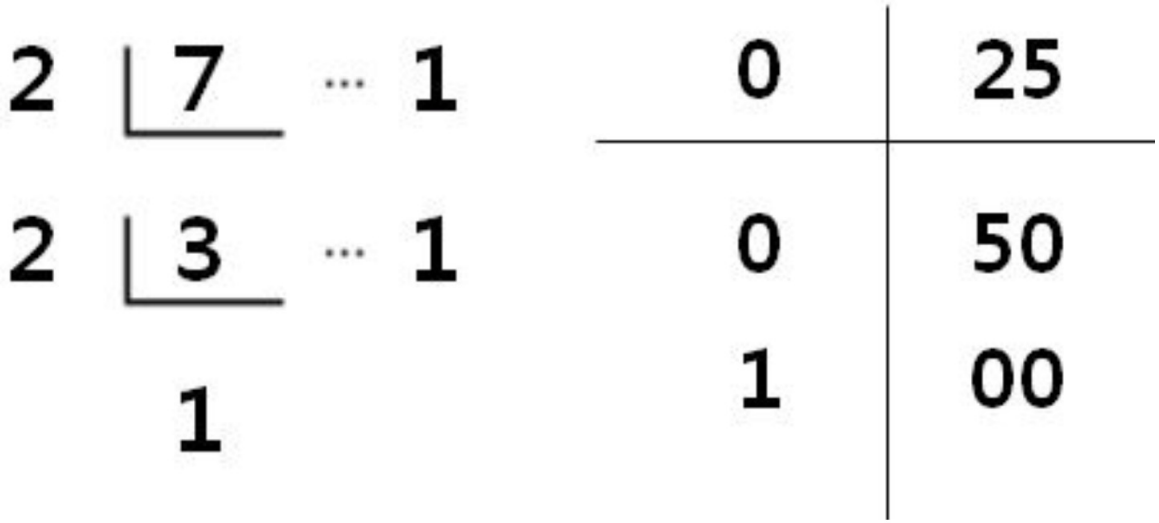
$$1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 \ + \ 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3}$$

N진법 변환하기

❑ **10진법 수를 N진법으로 변환하기**

- 10진법 수를 N으로 나누며 정리한다.

초등학교 때 배웠듯이 정수부분에 대한 2진법 계산은 숫자를 2로 계속 나누며 결과를 얻을 수 있습니다. 반대로 소수부분의 경우 2로 계속 곱하며 결과를 얻을 수 있습니다. 7.25는 위처럼 2진수로 계산하면 111.01(2)로 표현이 가능합니다.



왼쪽: 정수 계산 / 오른쪽: 가수 계산

진법변환 연습문제

문제) 십진수 33.25를 이진수로 올바르게 표현한 것은?

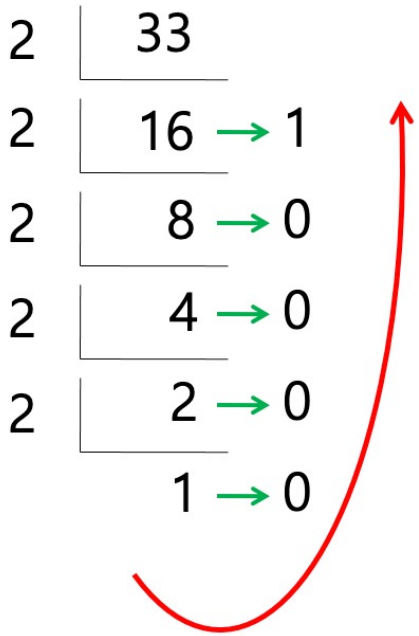
① 100001.01

② 100010.1

③ 100101.01

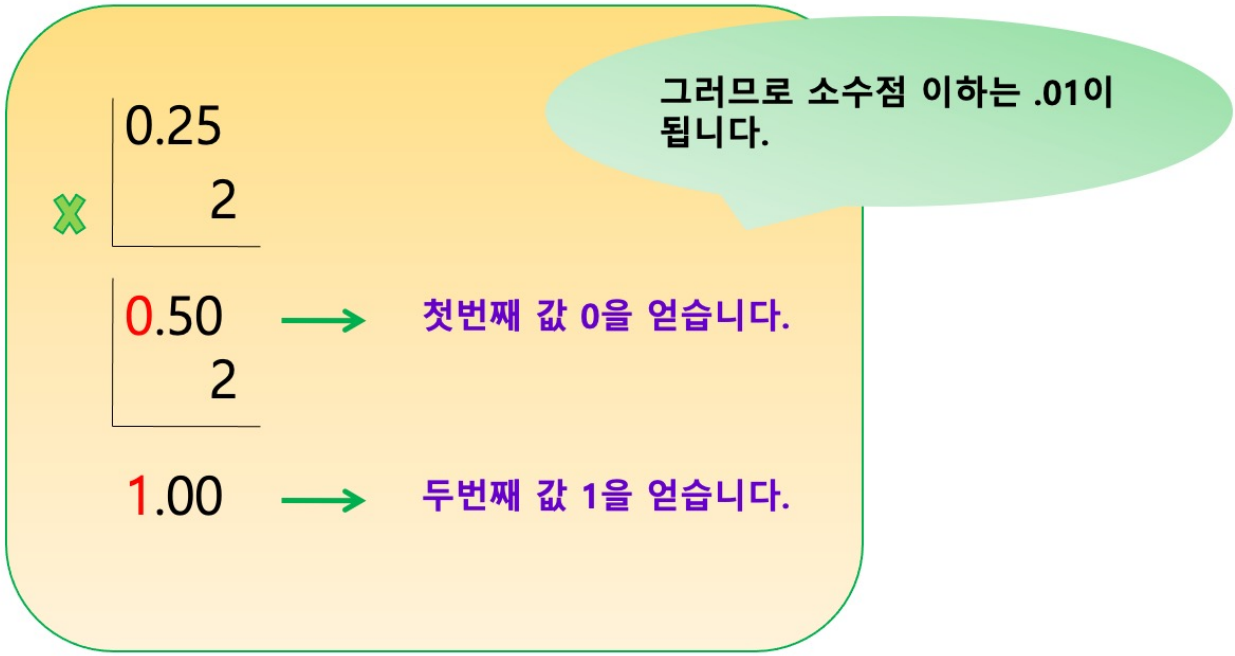
④ 11001.1

진법변환 연습문제



따라서 정수부분의 비트는 100001이됩니다.
이제는 소수점아래의 변환을 보겠습니다.

진법변환 연습문제



종합하면 33.25는 이진수로 100001.01가 됩니다.

보수법

❑ **N의 보수: 더했을 때 n0이 되는 수**

- 7의 ‘10의 보수’는 3이다.
- 2의 보수는 더해서 2가 되는 수이다.
- 10진수 2는 2진수 ’10’이고 자리수가 바뀌게 된다.
- 즉, 2의 보수를 찾으라는 것은 주어진 비트 내에서 합이 0이 되는 것을 찾는 것이라 할 수 있다.

보수법

❑ **2진수의 1의 보수 구하는 법**

- 주어진 2진수의 0은 1로, 1은 0으로 바꾼다.

❑ **2진수의 2의 보수 구하는 법**

- 1의 보수를 구한 후 1을 더해준다.

❑ **음수의 2진 표현을 구하는 법**

- 음수의 절대값을 2진수로 변환한다.
- 앞에서 구한 2진수의 1의 보수를 구한 후 1을 더한다

Cf) 1의 보수와 2의 보수 모두 첫번째 비트값(MBS)은 부호를 나타낸다

보수법 연습문제

1) -3을 2의 보수로 표현해 보세요.

보수법 연습문제

-3의 2's Complement

3를 이진수로 표현한다

0	0	0	0	0	0	1	1

1의 보수로 만든다

1	1	1	1	1	1	0	0

더하기 1을 하면 완성된다.

1	1	1	1	1	1	0	1

기본형 특징

❑ 자바가 데이터를 다루는 최소 단위는 byte이다.(bit x)

❑ 문자형(char)은 문자가 아닌 문자의 유니코드가 저장된다.

- char는 2byte이다(unicode 수)
- 따라서 인코딩과 디코딩 방식이 있다.

부동소수점 vs 고정소수점: 부동소수점

1. 부동소수점(Floating Point)

부동소수점이란 2진수로 변환한 실수를 정규화과정을 거쳐 저장하는 방식을 의미합니다. 부동소수점은 실수를 [부호 | 가수부 | 지수부] 형태로 나누어 해석합니다. 대부분의 시스템에서는 일반적으로 부동소수점의 방식으로 실수를 표현합니다. IEEE(전기전자 기술자 협회) 표준에 따르면 부동소수점의 방식으로 실수를 저장하는 때 32비트나 64비트를 사용합니다.

cf) 정규화: 2진수로 변환한 값을 (1.xxxxx)*2^n 형태로 바꾼 것.

<출처> 네이버 blog, 컴퓨터에서의 실수 표현: 고정소수점 vs 부동소수점

32비트의 경우로 예를 들면 부호 1비트, 지수부 8비트, 가수부 23비트가 할당이 됩니다.

부동소수점 vs 고정소수점: 부동소수점

32비트 작성방법:

① 실수 → 2진수 변환

② 정규화

③ 가수의 소수점(.xxx….)부분을 가수부에 작성. 남은 칸 0채우기

④ (n+127)을 2진수로 변환한 후 지수부에 삽입

⑤ 양수면 0, 음수면 1을 부호비트에 작성

이때 127은 bias라고 하며 지수부에 8자리를 할당했을 때 n이 음수일 경우를 처리하기 위함입니다. 64비트의 경우 bias는 1023입니다.

❑ 정규화 방식

- **10진수 123.45는 $1.2345*10^2$로 표현이 가능하다**
- **2진수 111.101은 $1.1101*2^2$로 표현이 가능하다**

부동소수점 vs 고정소수점: 부동소수점

Type	저장 가능한 값의 범위(양수)	정밀도	크기	
			bit	byte
float	1.4*10^(-45)~3.4*10^38	7자리	32	4
double	4.9*10^(-324)~1.8*10^308	15자리	64	8

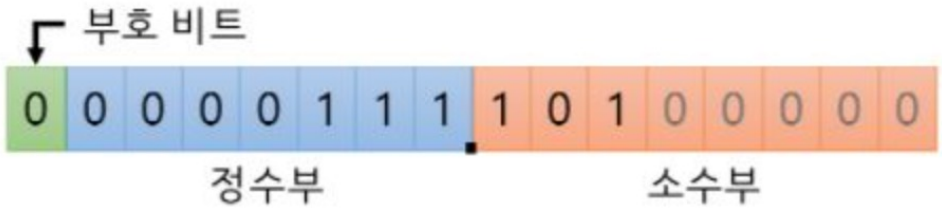
<출처> 남궁성, 「Java의 정석」, 도우출판, p.68

부동소수점을 나타내는 대표적인 type은 float와 double입니다. 표에서 말하는 정밀도는 위에 설명한 가수부를 통해 표현할 수 있는 10진수의 최대 자리수를 의미합니다. 정밀도는 뒤에 설명할 BigDecimal과 관련이 있습니다.

부동소수점 vs 고정소수점: 고정소수점

2. 고정소수점(Fixed Point)

고정소수점의 경우 [부호 | 정수부 | 소수부]로 표현이 됩니다. 고정소수점의 경우 소수부의 자릿수를 미리 정한 후 2진수로 변환한 값을 그대로 대입하는 방식입니다.



<출처> 네이버 blog, 컴퓨터에서의 실수 표현: 고정소수점 vs 부동소수점

고정소수점의 경우 정수부와 소수부의 지릿수가 크지 않으므로 표현할 수 있는 범위가 작다는 단점이 있습니다.

오버플로우 / 언더플로우

```java
class OverflowEx {
	public static void main(String[] args) {
		short sMin = -32768;
		short sMax = 32767;
		char  cMin = 0;
		char  cMax = 65535;

		System.out.println("sMin   = " + sMin);
		System.out.println("sMin-1= " + (short)(sMin-1));
		System.out.println("sMax   = " + sMax);
		System.out.println("sMax+1= " + (short)(sMax+1));
		System.out.println("cMin   = " + (int)cMin);
		System.out.println("cMin-1= " + (int)--cMin);
		System.out.println("cMax   = " + (int)cMax);
		System.out.println("cMax+1= " + (int)++cMax);
	}
}

```

Problems Javadoc Declaration Console

```
<terminated> OverflowEx [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.11.jdk/Contents/Home
sMin   = -32768
sMin-1= 32767
sMax   = 32767
sMax+1= -32768
cMin   = 0
cMin-1= 65535
cMax   = 65535
cMax+1= 0
```

- Underflow의 경우 0이 된다.

형변환(casting)

- **형변환**이란 변수 또는 상수의 타입을 다른 타입으로 변환하는 것을 의미한다.
- 일반적으로 큰 타입에 작은 타입을 넣는다.
- 작은 타입에 큰 타입을 넣을 경우 값 손실이 발생할 수 있다.

```
int a = 10;
float b = 10.3f;
short c = 5;

short d1 = a+c; // a가 int라(더 커서) 불가
short d2 = (short) a; // a의 값이 short 범위보다 클 경우 데이터 유실 가능
int e = a+c; // 가능
float f = a+b; // 가능
short g = (short) a;
```