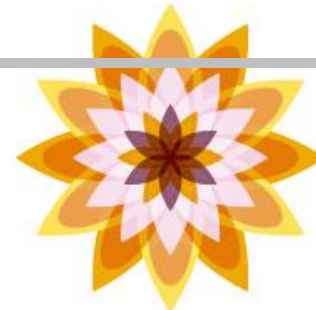
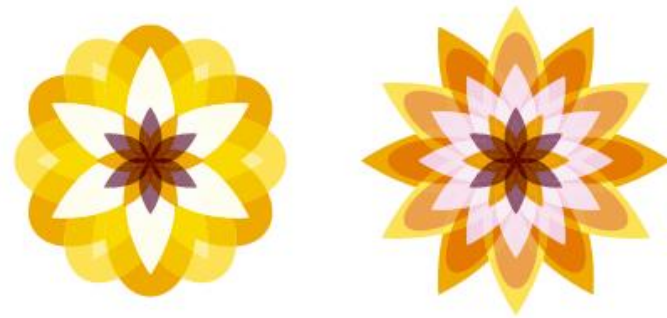


*Chapter 01*

# 오늘의 운세 게임



# 1. 이번에 만들 코드

- 우리가 처음으로 만들 게임은 게임이라고 말하기에는 너무나도 단순한 프로그램입니다.
- 실행을 시키면 단순히 오늘 날짜와 난수로 구한 금전운을 출력할 뿐입니다.
- 그러나 이 프로그램을 만들면서 자바 프로그래밍을 하는데 가장 기본이 되는 데이터형(자료형)을 배우고, 앞으로 복잡한 게임을 만들 때도 반드시 필요한, 날짜를 구하는 방법과 난수를 발생시키는 방법을 배우기 때문에 분명하게 이해하고 넘어가야 합니다.

## 2. 변수와 자료형

- 프로그래밍에서 가장 기본적인 동작은 데이터를 저장하고, 저장된 데이터 값을 읽어오는 것.
- 데이터를 저장하려면 메모리에 값을 저장할 공간을 생성하고 이름을 부여해야...
- 이때 메모리 공간에 부여하는 이름을 '변수'라고 한다.
- 데이터를 저장하기 위해 생성하는 메모리 공간은 목적에 따라 크기와 특징이 다른데, 이를 자료형(data type)이라고 한다.
- 즉, 메모리 공간의 자료형에 따라 저장할 수 있는 값의 종류와 특징이 결정된다.

## 2. 변수와 자료형

### ■ 자료형 선언하기

- C, 자바 같은 컴파일-파" 언어는 변수를 사용하기 전에 반드시 자료형을 선언해야 한다.
- 변수의 자료형을 선언할 때는 변수 이름 앞에 자료형을 표기해야 한다.
- 변수에 자료형이 선언되면 메모리 에는 변수값을 저장할 수 있는 공간이 만들어진다.

```
// 자료형 변수명;  
int a;  
String b;
```

- 이때 2가지 주의할 점이 있다.
  - 첫째, 자료형은 반드시 사용하기 전에 선언해야 한다. 자료형이 선언되지 않은 변수는 사용할 수 없다.
  - 둘째, 자료형은 반드시 한 번만 선언해야 한다. 자료형이 한 번 선언된 변수의 자료형은 바꿀 수 없다.

## 2. 변수와 자료형

### ■ 자료형 선언하기

#### ■ 자료형의 선언 예 1

```
int a = 3;           // (o)  
b = 5;              // (x)  
String c;  
c = "안녕";         // (o)
```

#### ■ 자료형의 선언 예 2

```
int a = 3;           // (o)  
double a = 5;        // (x) -> 변수 a는 int 자료형으로 선언돼 다른 자료형으로 변경 불가능  
String b = "안녕";   // (o)  
b = 10;              // (x) -> String 자료형에는 문자열만 저장 가능
```

## 2. 변수와 자료형

### ■ 변수 사용하기

#### ■ 변수를 사용하려면 두 단계를 거쳐야 한다.

- 첫째, 앞서 살펴본 변수에 자료형을 지정해 선언하는 변수의 선언 과정이다.
- 둘째, 선언된 변수에 값을 대입하는 과정이다.

#### ■ 변수 선언과 함께 값 대입하기

- 첫 번째는 변수의 선언과 값의 대입을 함께 수행해 한꺼번에 처리하는 방법이다.
- 명령 하나로 2개의 수행 내용(선언 및 대입)을 처리하지만, 내부에서는 변수 선언이 먼저 수행돼 메모리에 공간이 생성된 다음 생성된 공간에 값이 대입된다.

```
// 자료형 변수명 = 값;  
int a = 3;
```



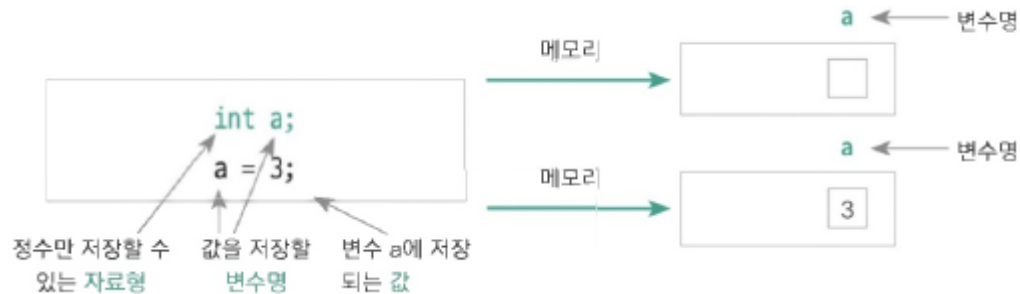
## 2. 변수와 자료형

### ■ 변수 사용하기

#### ■ 변수 선언과 값 대입 분리하기

- 두 번째는 변수의 선언과 값의 대입을 분리해 수행하는 방법 인데, 앞서 설명한 것처럼 변수의 선언이 먼저 이뤄져야 한다.
- 어떤 값을 저장할 수 있는 변수인지를 먼저 정해야 적절한 값을 대입할 수 있기 때문이다.

```
// 자료형 변수명;  
// 변수명 = 값;  
int a;  
a = 3;
```



## 2. 변수와 자료형

### ■ 변수 사용하기

- 변수 선언과 값을 대입하는 2가지 방법

UsageOfDataType.java

```
1 package sec01_datatype.EX01_UsageOfDataType;
2
3 public class UsageOfDataType {
4     public static void main(String[] args) {
5         // 변수 선언과 함께 값 대입
6         int a = 3;
7         // 변수 선언과 값 대입 분리
8         int b;
9         b = 4;
10        System.out.println(a);
11        System.out.println(b);
12    }
13 }
```



### ■ 이름 짓기

- 변수와 상수의 이름을 지을 때는 필수 사항과 권장 사항을 준수해야 한다.
- 필수 사항은 지키지 않으면 문법 오류가 발생해 컴파일 자체가 안 된다.
- 반면 권장 사항은 개발자끼리 약속한 정도로 이해하면 되고, 지키지 않는다 하더라도 문법 오류가 발생하지 않는다.
- 이름을 지을 때 지켜야 하는 필수 사항
  - 변수, 상수, 메서드의 이름을 지을 때 반드시 지켜야 하는 공통 사항
    - 영문 대소 문자와 한글을 사용할 수 있다.
    - 예) abc(O), ABC(O), aBc(O), 가나다(O)
    - 특수 문자는 밑줄(\_)과 달러(\$) 표기만 사용할 수 있다.
    - 예) \$abc(O), \_abc(O), ab\_c(O), \$abc\_(O)
    - 아라비아 숫자를 사용할 수 있다. 단, 첫 번째 글자로는 사용할 수 없다.
    - 예) a3bc(O), ab3c(O), abc3(O), 3abc(X)
    - 자바에서 사용하는 예약어는 사용할 수 없다.
    - 예) int(X), break(X), public(X), static(X)

## 2. 변수와 자료형

### ■ 이름 짓기

#### ■ 이름을 지을 때 지키면 좋은 권장 사항

- 권장 사항의 핵심은 변수, 상수 그리고 메서드를 그 이름이나 구조만으로도 구분할 수 있게 하는 것이다.
- 변수명을 지을 때 권장 사항
  - 영문 소문자로 시작한다.
  - 예) avg, name, value
  - 영문 단어를 2개 이상 결합할 때는 새로운 단어의 첫 글자를 대문자로 한다.
  - 예) myWork, maxVahje, bestPosition
- 변수명을 지을 때의 권장 사항을 정리해 보면, 이름은 소문자로 시작하고 새로운 단어를 결합할 때는 의미를 파악하기 쉽도록 대문자로 시작한다는 것이다.
- 이를 낙타의 혹처럼 생겼다고 해서 '낙타표기법'이라고도 한다.
- 낙타 표기법의 예

```
int ourClassNum;
```

## 2. 변수와 자료형

### ■ 이름 짓기

#### ■ 이름을 지을 때 지키면 좋은 권장 사항

- 상수명을 지을 때 권장 사항
  - 상수는 변수와 구분하기 위해 모두 대문자로 표기한다.
  - 다만 단어가 여러 개 결합하면 가독성이 떨어지므로 각각 밑줄(\_)을 사용해 분리한다.
  - 상수명 짓기 예

```
final int MY_DATA;
```

- 메서드명을 지을 때 권장 사항
  - 메서드명을 지을 때의 권장 사항은 변수명일 때와 같다.
  - 단, 메서드는 이름 뒤에 소괄호()와 중괄호{}가 붙는다.

## 2. 변수와 자료형

### ■ 이름 짓기

#### ■ 실습. 변수와 상수명 짓기

#### NamingVariableAndConstant.java

```
1 package sec01_datatype.EX02_NamingVariableAndConstant;
2
3 /*변수와 상수의 이름 짓기*/
4 public class NamingVariableAndConstant {
5     public static void main(String[] args) {
6         //변수의 이름
7         boolean aBcD;                //가능은 하지만 권장사항 위배
8         byte 가나다;
9         short _abcd;
10        char $ab_cd;
11        //int 3abcd;                //불가능 - 숫자는 제일 앞에 올 수 없다.
12        long abcd3;
13        //float int;                //불가능 - 자바에서 사용하는 키워드
14        double main;                //가능 - 메서드 이름과 변수이름은 동일해도 무관
```

## 2. 변수와 자료형

### ■ 이름 짓기

#### ■ 실습. 변수와 상수명 짓기

NamingVariableAndConstant.java

```
15      //int my Work;           //불가능 스페이스 특수키가 포함
16      String myClassName;
17      int ABC;                 //가능은 하지만 권고사항 위배
18
19      //상수이름
20      final double PI;
21      final int MY_DATA;
22      final float myData;      //가능은 하지만 권고사항 위배
23  }
24 }
```

## 2. 변수와 자료형

### ■ 변수의 생존 기간

- 변수의 생존 기간은 메모리에 변수가 만들어진 이후 사라지기까지의 기간을 의미한다.
- 자바에서는 개발자가 직접 변수를 생성한다.
- 하지만 메모리에서 변수를 삭제하는 작업은 자바 가상 머신이 알아서 한다.
- 변수를 삭제하는 주체가 개발자가 아니다 보니 메모리에서 변수가 사라지는 시점을 이해하는 것은 매우 중요하다.
- 만일 사라진 변수값을 읽거나 값을 대입하려고 하면 문법 오류가 발생하기 때문이다.
- 그렇다면 메모리의 변수는 정확히 언제 생성되고 사라질까?
- 먼저 변수는 선언된 시점에 생성된다.
- 이후 생성된 변수는 자신이 선언된 열린 중괄호({)의 쌍인 닫힌 중괄호(})를 만나면 메모리에서 삭제된다.
- 변수의 생성과 소멸 시점의 예

```
{  
    int a; // 변수 a의 생성 시점  
    {  
        a = 3;  
    }  
} // 변수 a의 소멸 시점
```

## 2. 변수와 자료형

### ■ 변수의 생존 기간

#### ■ 실습. 변수의 생존 기간

RangeOfVariableUse.java

```
1 package sec01_datatype.EX03_RangeOfVariableUse;
2
3 /*변수의 생존기간*/
4 public class RangeOfVariableUse {
5     public static void main(String[] args) {
6         int value1 = 3;
7         {
8             int value2 = 5;
9             System.out.println(value1);    //3
10            System.out.println(value2);    //5
11        }
12
13        System.out.println(value1);        //3
14        //System.out.println(value2);    //오류
```

## 2. 변수와 자료형

### ■ 변수의 생존 기간

- 실습. 변수의 생존 기간

RangeOfVariableUse.java

```
15     }
```

```
16 }
```



### 3. 자료형의 종류

#### ■ 자바에서 사용하는 자료형의 종류

구분	저장값	자료형
기본 자료형	참, 거짓: true, false	boolean
	정수: ..., -1, 0, 1, ...	byte, short, int, long
	실수: -5.4, 1.7, ...	float, double
	문자(정수): 'A', 'b', ...	char
참조 자료형	객체: Object	배열, 클래스, 인터페이스

- 자료형은 크게 '기본 자료형'과 '참조 자료형'으로 나눌 수 있다.
- 자료형을 이렇게 구분해야 하는 이유는 기본 자료형과 참조 자료형의 값 저장 방식이 서로 다르기 때문이다.

### 3. 자료형의 종류

#### ■ 메모리는 목적에 따라 크게 3가지 영역으로 나뉜다.

- 첫 번째 영역은 클래스(class) 영역, 정적(static) 영역, 상수(final) 영역, 메서드(method) 영역이라는 4개의 이름으로 불린다.
- 두 번째 영역은 스택(stack) 영역으로 변수들이 저장되는 공간이다.
- 마지막 영역은 힙(heap) 영역으로 객체들이 저장되는 공간이다.



# 3. 자료형의 종류

## ■ 기본 자료형과 참조 자료형의 차이

### ■ 자료형의 이름 규칙

- 기본 자료형과 참조 자료형의 첫 번째 차이점은 자료형 자체의 이름 규칙에 있다.
- 자바에서 제공하는 기본 자료형 8개의 이름은 모두 소문자(int, long, float, double, ...)로 시작하는 반면, 참조 자료형의 이름은 모두 대문자(String, System, ...)로 시작한다.
- 물론 권장 사항이므로 직접 생성하는 참조 자료형 이름이 소문자로 시작해도 오류는 발생하지 않는다.

### ■ 실제 데이터값의 저장 위치

- 기본 자료형과 참조 자료형의 두 번째 차이점은 실제 데이터값의 저장 위치가 다르다는 것이다.
- 기본 자료형과 참조 자료형 모두 변수의 공간이 스택 메모리에 생성되지만, 그 공간에 저장되는 값의 의미가 서로 다르다.
- 기본자료형은 스택 메모리에 생성된 공간에 실제 변수값을 저장하는 반면, 참조 자료형은 실제 데이터값은 힙 메모리에 저장하고, 스택 메모리의 변수 공간에는 실제 변수값이 저장된 힙 메모리의 위치값을 저장한다.



### 3. 자료형의 종류

#### ■ 기본 자료형의 메모리 크기와 저장할 수 있는 값의 범위

- 기본 자료형에는 참(true)과 거짓(false)을 저장하는 Boolean, 정수를 저장하는 byte, short, int, long, 실수를 저장하는 float과 double 그리고 문자(정수)를 저장하는 char 자료형이 있다.

자료형		자료의 크기	값의 범위
부울대수	boolean	1byte	true, false
정수	byte	1byte	$-2^7 \sim 2^7-1$
	short	2byte	$-2^{15} \sim 2^{15}-1$
	int	4byte	$-2^{31} \sim 2^{31}-1$
	long	8byte	$-2^{63} \sim 2^{63}-1$
실수	float	4byte	$\pm(1.40 \times 10^{-45} \sim 3.40 \times 10^{38})$
	double	8byte	$\pm(4.94 \times 10^{-324} \sim 1.79 \times 10^{308})$
문자(정수)	char	2byte	유니코드 문자( $0 \sim 2^{16}-1$ )

구분	부호 비트	가수 비트	지수 비트
float(32bit)	1	23	8
double(64bit)	1	52	11

### 3. 자료형의 종류

#### ■ 기본 자료형의 메모리 크기와 저장할 수 있는 값의 범위

##### ▣ float과 double의 정밀도 비교

FloatVsDouble.java

```
1 package sec02_primarydatatype.EX01_FloatVsDouble;
2
3 /*float과 double의 정밀도 비교*/
4 public class FloatVsDouble {
5     public static void main(String[] args) {
6         // #1. float의 정밀도 (대략 소수 7자리)
7         float f1 = 1.0000001f;
8         System.out.println(f1);
9         float f2 = 1.00000001f;
10        System.out.println(f2);
11
12        // #2. double의 정밀도 (대략 소수 15자리)
13        double d1 = 1.0000000000000001;
14        System.out.println(d1);
```

### 3. 자료형의 종류

#### ■ 기본 자료형의 메모리 크기와 저장할 수 있는 값의 범위

- float과 double의 정밀도 비교

FloatVsDouble.java

```
15      double d2 = 1.000000000000000001;  
16      System.out.println(d2);  
17  }  
18 }
```

### 3. 자료형의 종류

#### ■ 부울대수 자료형 - boolean

- boolean은 true(참)과 false(거짓)의 값만 저장할수 있는 자료형이다.
- 불리언은 이렇게 2가지 값만 포함할 수 있으므로 실제로는 1bit로도 충분하지만, 자료 처리의 최소 단위가 바이트이므로 1byte가 할당된다.
- 실제로 할당된 1byte(=8bit) 가운데 상위 7bit는 사용하지 않는다.

```
boolean a = true;  
boolean b = false;
```

### 3. 자료형의 종류

#### ■ 정수 자료형 - byte, short, int, long

- 정수를 저장할 수 있는 자료형에는 byte, short, int, long이 있으며, 크기는 각각 1byte, 2byte, 4byte, 8byte다.
- 예를 들어 변수 a에 정수 3을 저장한다면 다음과 같이 4가지 자료형으로 각각 다르게 정의할 수 있다.

```
byte a = 3;  
short a = 3;  
int a = 3;  
long a = 3;
```

- 자바에서는 반드시 지켜야 하는 규칙이 1 개 있는데, 항상 대입 연산자(=)를 중심으로 양쪽의 자료형이 똑같아야 한다는 것이다.
- 만일 양쪽의 자료형이 같지 않으면 문법 오류가 발생한다.
- 그렇다면 코드에서 직접 숫자로 쓴 정수 3은 어떤 자료형일까?
- 코드에 직접 작성한 값을 리터럴(literal)이라고 하는데, 앞서도 설명한 것처럼 3을 저장할 수 있는 자료형이 4개나 있어서 컴파일러는 해당 값의 자료형을 그 중 하나로 정해야 한다.



### 3. 자료형의 종류

#### ■ 정수 자료형 - byte, short, int, long

- 정수 리터럴의 기준은 크게 2가지로 나뉜다.

대입식	정수 리터럴의 자료형 인식
byte a = 3;	byte에 저장할 수 있는 값이 byte 자료형에 대입되면 byte로 인식(byte = byte)
short a = 3;	short에 저장할 수 있는 값이 short 자료형에 대입되면 short로 인식 (short = short)
int a = 3;	int와 같거나 int보다 큰 정수 자료형(long)에 정숫값이 입력되면 크기에 관계없이 int로 인식(int = int)
long a = 3L;	정수 리터럴 뒤에 L(또는 l)이 붙으면 long으로 인식(long = long)
long a = 3;	long 자료형에 대입되는 정숫값은 크기에 상관없이 int로 인식되지만, 자동 타입 변환이 발생해 long으로 변환(long = long)
byte a = 130;	byte에 저장할 수 없는 범위의 정숫값이 입력되면 int로 인식(byte = int) → 오류 발생

- 위에서 아무렇지도 않게 쓴 long a = 3은 틀린 표현일까?
- 정수 리터럴 뒤에 L을 붙이지 않았으므로 int 자료형으로 인식할 것이고, long = int의 형태가 돼 자료형이 불일치하는 것처럼 보인다.
- 하지만 이 코드에서는 오류가 발생하지 않는다.
- 그 이유는 크기가 작은 자료형을 큰 자료형에 대입하면 컴파일러가 자동 타입 변환(type casting)을 수행하기 때문이다.
- 반면에, 큰 자료형에서 작은 자료형으로의 변환은 자동으로 일어나지 않는다.

### 3. 자료형의 종류

#### ■ 실수 자료형 - float, double

- 자바는 실수 리터럴을 double 자료형으로 인식한다.
- 단, float를 나타내는 F(또는 f)를 실수 리터럴 뒤에 붙이면 float 자료형으로 인식한다.

대입식	실수 리터럴의 자료형 인식
float a = 3.5F;	뒤에 F가 붙었으므로 float로 인식(float = float)
double a = 5.8;	실수 리터럴은 기본적으로 double로 인식(double = double)
double a = 5;	int로 인식하지만 자동 타입 변환해 double로 인식(double = double)
float a = 3.5;	실수 리터럴은 기본적으로 double로 인식(float = double) → 오류 발생

### 3. 자료형의 종류

#### ■ 실수 자료형 - float, double

- 실습. 부울대수, 정수, 실수 값의 저장 및 출력

PrimaryDataType\_1.java

```
1 package sec02_primarydatatype.EX02_PrimaryDataType_1;
2
3 /*부울대수/정수/실수 값의 저장 및 출력*/
4 public class PrimaryDataType_1 {
5     public static void main(String[] args) {
6         //#1. boolean : true / false
7         boolean bool1 = true;
8         boolean bool2 = false;
9         System.out.println(bool1);           //true
10        System.out.println(bool2);           //false
11        System.out.println();
12
13        //#2. 정수 (byte, short, int, long) : 음의 정수 / 0 / 양의 정수
14        byte value1 = 10;
```

### 3. 자료형의 종류

#### ■ 실수 자료형 - float, double

- 실습. 부울대수, 정수, 실수 값의 저장 및 출력

PrimaryDataType\_1.java

```
15      short value2 = -10;
16      int value3 = 100;
17      long value4 = -100L;
18      System.out.println(value1);          //10
19      System.out.println(value2);          //-10
20      System.out.println(value3);          //100
21      System.out.println(value4);          //-100
22      System.out.println();
23
24      // #3. 실수 (float, double) : 음의 실수 / 0 / 양의 실수
25      float value5 = 1.2F;
26      double value6 = -1.5;
27      double value7 = 5;
28      System.out.println(value5); //1.2
```

### 3. 자료형의 종류

#### ■ 실수 자료형 - float, double

- 실습. 부울대수, 정수, 실수 값의 저장 및 출력

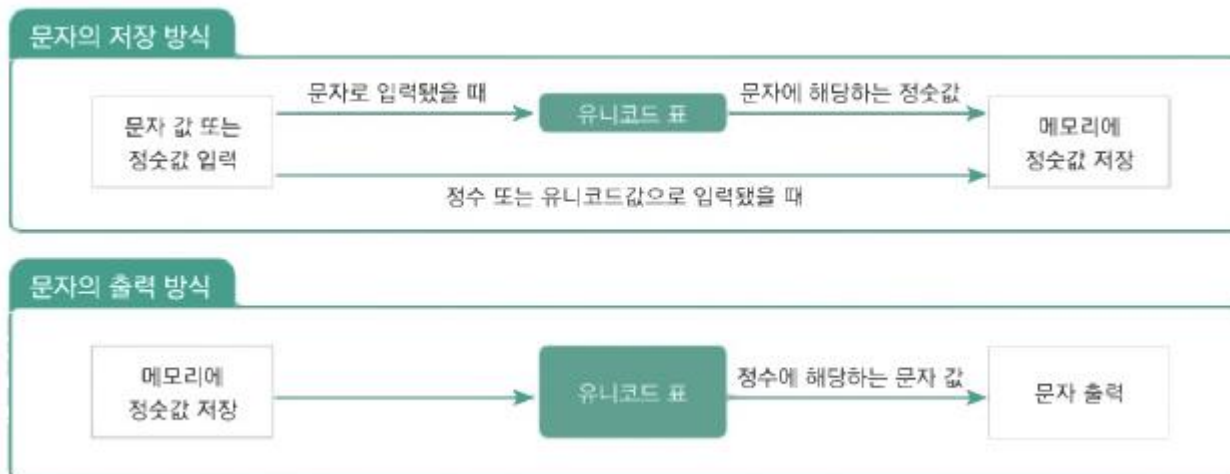
PrimaryDataType\_1.java

```
29      System.out.println(value6); //-1.5
30      System.out.println(value7); //5.0
31  }
32 }
```

### 3. 자료형의 종류

#### ■ 문자 자료형

- char는 문자를 저장하는 자료형으로, 문자를 작은 따옴표(") 안에 표기한다.
- char 자료형은 정수를 저장할 수도 있지만, 앞에서 배운 정수 자료형 4가지와 다소 차이가 있다.
- 'A'라는 문자를 char 자료형에 저장하기 위해 코드를 `char a = 'A'` 와 같이 작성했다면 메모리에는 변수 a의 공간이 만들어지고, 그 안에 문자가 들어가야 할 것이다.
- 하지만 메모리에는 문자를 기록할 수 없다.
- 메모리는 2진수 값만 저장할 수 있는 공간이기 때문이다.
- 그렇다면 문자를 어떻게 저장할까?
- 바로 모든 문자를 특정 정숫값으로 바꿔 저장하는 것이다.



### 3. 자료형의 종류

#### ■ 문자 자료형

- 다음은 char 자료형 변수에 문자 'A'를 다양한 형태로 저장하는 예다.

```
char a = 'A';           // 문자
char b = 65;            // 10진수
char c = 0b1000001;     // 2진수
char d = 00101;         // 8진수
char e = 0x0041;        // 16진수
char f = 'Wu0041';      // 유니코드
```

### 3. 자료형의 종류

#### ■ 문자 자료형

- 문자값의 다양한 저장 방법 및 출력

PrimaryDataType\_2.java

```
1 package sec02_primarydatatype.EX03_PrimaryDataType_2;
2
3 /*문자 값의 다양한 저장방법 및 출력*/
4 public class PrimaryDataType_2 {
5     public static void main(String[] args) {
6         // #4. 문자(char) : 문자, 정수
7         // #4-1. 문자로 저장하는 방법
8         char value1 = 'A';
9         char value2 = '가';
10        char value3 = '3';
11        System.out.println(value1); //A
12        System.out.println(value2); //가
13        System.out.println(value3); //3
14        System.out.println();
```



### 3. 자료형의 종류

#### ■ 문자 자료형

- 문자값의 다양한 저장 방법 및 출력

PrimaryDataType\_2.java

```
15
16      // #4-2. 정수로 저장하는 방법
17      char value4 = 65;
18      char value5 = 0xac00;
19      char value6 = 51;
20      System.out.println(value4); //A
21      System.out.println(value5); //가
22      System.out.println(value6); //3
23      System.out.println();
24
25      // #4-3. 유니코드로 직접 입력
26      char value7 = '\u0041';
27      char value8 = '\uac00';
28      char value9 = '\u0033';
```

### 3. 자료형의 종류

#### ■ 문자 자료형

- 문자값의 다양한 저장 방법 및 출력

PrimaryDataType\_2.java

```
29      System.out.println(value7); //A
30      System.out.println(value8); //가
31      System.out.println(value9); //3
32  }
33 }
```

## 4. 기본 자료형 간의 타입 변환

- boolean 을 제외한 기본 자료형 7개는 자료형을 서로 변환할 수 있는데, 이를 '타입 변환(type casting)'이라고 한다.
- 앞서서도 설명한 것처럼 자바는 항상 대입 연산자 ( = ) 를 중심으로 왼쪽과 오른쪽 자료형을 일치시켜야 하므로 타입 변환을 수행해야 할 때가 있다.
- 타입 변환 방법은 단순히 변환 대상 앞에 (자료형)만 표기하면 된다.
- 정수나 실수 리터럴은 숫자 뒤에 L이나 F를 붙여 각각 long, float로 타입 변환을 수행할 수도 있다.

자료형 A 변수명 = (자료형 A) 값

long 변수명 = 값 + L;

long 변수명 = 값 + l;

float 변수명 = 값 + F;

float 변수명 = 값 + f;

## 4. 기본 자료형 간의 타입 변환

### ■ 실습. 2가지 타입 변환 방법

### TypeCasting\_1.java

```
1 package sec02_primarydatatype.EX04_TypeCasting_1;
2
3 /*두 가지 타입 변환 방법*/
4 public class TypeCasting_1 {
5     public static void main(String[] args) {
6         //#1. 캐스팅 방법
7         //@1-1 캐스팅 방법 1 (데이터타입)
8         int value1 = (int)5.3;
9         long value2 = (long)10;
10        float value3 = (float)5.8;
11        double value4 = (double)16;
12
13        System.out.println(value1); //5
14        System.out.println(value2); //10
15        System.out.println(value3); //5.8
```

## 4. 기본 자료형 간의 타입 변환

### ■ 실습. 2가지 타입 변환 방법

TypeCasting\_1.java

```
16      System.out.println(value4); //16.0
17      System.out.println();
18
19      //@1-2 캐스팅 방법 2 (L, F)
20      long value5 = 10L;
21      long value6 = 10l;
22      float value7 = 5.8F;
23      float value8 = 5.8f;
24
25      System.out.println(value5); //10
26      System.out.println(value6); //10
27      System.out.println(value7); //5.8
28      System.out.println(value8); //5.8
29  }
30 }
```

## 4. 기본 자료형 간의 타입 변환

### ■ 자동 타입 변환과 수동 타입 변환

- 타입 변환에는 컴파일러가 자동으로 수행하는 '자동 타입 변환'과 개발자가 직접 타입 변환을 수행해야 하는 '수동 타입 변환'이 있다.
- 먼저 크기(범위)가 작은 자료형을 큰 자료형에 대입할 때를 살펴보자.
- 이때는 어떠한 데이터 손실도 발생하지 않는다.
- 따라서 작은 자료형을 큰 자료형에 담으면 개발자가 타입 변환 코드를 넣어 주지 않더라도 컴파일러가 자동으로 타입 변환을 실행하는데, 이를 '업캐스팅(up-casting)'이라고 한다.
- 업캐스팅이 아닌데도 자동 타입 변환이 적용되는 때가 있다.
- 사실 모든 정수 리터럴 값은 int 자료형으로 인식된다.
- 하지만 byte 및 short 자료형에 저장할 수 있는 범위 내의 정수 리터럴 값이 대입될 때는 자동 타입 변환이 각각의 자료형으로 수행된다.
- 이것이 바로 앞서 살펴본 것처럼 byte 또는 short 자료형에 저장할 수 있는 리터럴값이 입력될 때 해당 리터럴이 각각의 타입으로 인식되는 이유다.
- 이제 반대 상황을 살펴보자.
- 큰 자료형을 작은 자료형에 대입하는 행위를 '다운캐스팅(down-casting)'이라고 한다.
- 이때는 데이터 손실이 발생할 수 있으므로 컴파일러에 따른 자동 타입 변환은 일어나지 않으며, 개발자가 직접 명시적으로 타입 변환을 수행해야 한다.
- 자료형의 크기는 'byte<short<char<int<long<float<double'의 순서로 커진다.

## 4. 기본 자료형 간의 타입 변환

### ■ 자동 타입 변환과 수동 타입 변환

#### ■ 실습. 자동 타입 변환과 수동 타입 변환

TypeCasting\_2.java

```
1 package sec02_primarydatatype.EX05_TypeCasting_2;
2
3 /*자동타입변환과 수동타입변환*/
4 public class TypeCasting_2 {
5     public static void main(String[] args) {
6         //#1. 자동타입변환 (업캐스팅 + byte/short 자료형 데이터 입력)
7         float value1 = 3;        //int -> float (업캐스팅)
8         long value2 = 5;         //int -> long (업캐스팅)
9         double value3 = 7;       //int -> double (업캐스팅)
10        byte value4 = 9;          //(예외: int -> byte)
11        short value5 = 11;        //(예외: int -> short)
12
13        System.out.println(value1);
14        System.out.println(value2);
```

## 4. 기본 자료형 간의 타입 변환

### ■ 자동 타입 변환과 수동 타입 변환

#### ■ 실습. 자동 타입 변환과 수동 타입 변환

TypeCasting\_2.java

```
15      System.out.println(value3);
16      System.out.println(value4);
17      System.out.println(value5);
18      System.out.println();
19
20      ///#2. 수동 타입변환
21      byte value6 = (byte)128;          //int -> byte (다운캐스팅)
22      int value7 = (int)3.5;    //double -> int (다운캐스팅)
23      float value8 = (float)7.5;        //double -> float (다운캐스팅)
24
25      System.out.println(value6);
26      System.out.println(value7);
27      System.out.println(value8);
28  }
```



## 4. 기본 자료형 간의 타입 변환

### ■ 기본 자료형 간의 연산

- `boolean`을 제외한 나머지 기본 자료형은 서로 연산할 수 있다.
- 이때 모든 연산은 같은 자료형끼리만 가능하며 연산 결과 역시 같은 자료형이 나온다.
- 단, `int`보다 작은 자료형 간의 연산 결과는 `int`가 나온다.
- 예를 들어 `byte + byte`, `short + short`, `byte + short`의 결과 자료형은 `int`이다.
- 이는 CPU에서 연산 최소 단위가 `int`이므로 `int`보다 작은 자료형도 일단 `int`로 읽어 와서 연산을 수행하기 때문이다.
- 즉, CPU에서는 `int + int`가 수행된 셈이므로 결과도 `int`가 나오는 것이다.
- 쉽게 생각해서 `int`보다 작은 자료형 간의 연산은 `int`, `int`보다 크거나 같은 자료형 간의 연산은 해당 자료형이 나온다고 생각하면 된다.

연산	결과
byte 자료형 + byte 자료형	int 자료형
short 자료형 + short 자료형	int 자료형
int 자료형 + int 자료형	int 자료형
long 자료형 + long 자료형	long 자료형
float 자료형 + float 자료형	float 자료형
double 자료형 + double 자료형	double 자료형

## 4. 기본 자료형 간의 타입 변환

### ■ 기본 자료형 간의 연산

- 그렇다면 `int + long`은 계산할 수 없을까?
- 이것이 바로 타입 변환을 배운 이유다.
- 다운캐스팅은 개발자가 직접 해줘야 하지만, 업캐스팅은 자동 타입 변환이므로 `int + long`을 수행하면 컴파일러는 연산을 위해 앞의 `int`를 `long`으로 자동 업캐스팅해 `long + long`으로 계산한다.
- 당연한 이야기겠지만, 결과도 `long`의 값이 나올 것이다.
- 여기서도 역시 `int`보다 작은 자료형 간의 연산 결과는 `int` 자료형이 나올 것이다.

연산	결과
byte 자료형 + short 자료형	int 자료형
byte 자료형 + int 자료형	int 자료형
short 자료형 + long 자료형	long 자료형
int 자료형 + float 자료형	float 자료형
long 자료형 + float 자료형	float 자료형
float 자료형 + double 자료형	double 자료형

## 4. 기본 자료형 간의 타입 변환

### ■ 기본 자료형 간의 연산

- 실습. 같은 자료형 간의 연산과 다른 자료형 간의 연산

OperationBetweenDataType.java

```
1 package sec02_primarydatatype.EX06_OperationBetweenDataType;
2
3 /*같은 자료형간의 연산과 다른 자료형간의 연산*/
4 public class OperationBetweenDataType {
5     public static void main(String[] args) {
6         // #1. 같은 자료형간의 연산
7         int value1 = 3+5;
8         int value2 = 8/5; //1
9         float value3 = 3.0f + 5.0f;
10        double value4 = 8.0/5.0; //1.6
11
12        byte data1 = 3;
13        byte data2 = 5;
14        //byte value5 = data1 + data2; //오류
```

## 4. 기본 자료형 간의 타입 변환

### ■ 기본 자료형 간의 연산

- 실습. 같은 자료형 간의 연산과 다른 자료형 간의 연산

OperationBetweenDataType.java

```
15         int value5 = data1 + data2;
16
17         System.out.println(value1);
18         System.out.println(value2);
19         System.out.println(value3);
20         System.out.println(value4);
21         System.out.println(value5);
22         System.out.println();
23
24         // #2. 다른 자료형 간의 연산
25         // int value6 = 5 + 3.5; // 오류
26         double value6 = 5 + 3.5;
27         int value7 = 5 + (int)3.5;
28
```

## 4. 기본 자료형 간의 타입 변환

### ■ 기본 자료형 간의 연산

- 실습. 같은 자료형 간의 연산과 다른 자료형 간의 연산

OperationBetweenDataType.java

```
29      double value8 = 5/2.0;
30      byte data3 = 3;
31      short data4 = 5;
32      int value9 = data3 + data4;
33      double value10 = data3 + data4;
34
35      System.out.println(value6);
36      System.out.println(value7);
37      System.out.println(value8);
38      System.out.println(value9);
39      System.out.println(value10);
40  }
41 }
```

## 5. Date 클래스

- 게임에서 오늘의 날짜와 현재시간을 알아야하는 경우는 빈번하게 발생한다.
- 예를 들어 게임의 점수를 기록할 때, 게임을 한 날도 함께 기록할 수 있다.
- 게임에 따라서는 플레이한 시간에 따라 점수를 계산해야 하는 경우도 있다.
- 퍼즐게임 등은 제한시간 내에 풀어야한다는 규칙이 있는 경우가 많이 있다.
- 따라서 여기서 배우는 오늘의 날짜와 현재시간을 알아내는 방법은 꼭 알아두어야 한다.
- 자바에서는 날짜와 시간을 알아내기 위해 Date라는 클래스를 미리 만들어서 제공하고 있다.
- 우리는 아직 클래스에 대해 배우지 않았지만, 현재 시점에서는 전체 프로그램을 이루는 작은 프로그램정도로만 이해하도록 하자.
- 엄격히 말하면 우리가 지금까지 만든 모든 프로그램도 클래스이다.

## 5. Date 클래스

- `Byte myByte;`라고 선언하면 우리도 모르는 사이에 자바가 메모리에 1바이트 크기의 공간을 잡아서 `myByte`라고 이름표를 붙여준다.
- 그래야만 `myByte = 5;`와 같은 명령을 내렸을 때, 5를 메모리에 저장할 수 있다.
- 마찬가지로 클래스도 메모리에 공간을 할당해줘야 값을 저장할 수 있는데, 이렇게 클래스에 메모리 공간을 할당하는 작업을 '생성한다'고 한다.
- 클래스가 앞에서 배운 기본 데이터형들과 다른 점은 자동으로 공간이 잡히지 않기 때문에 사용 전에 우리가 직접 공간을 잡아줘야만 한다는 점이다.
- 따라서 클래스를 사용하기 위해선, 사용 전에 반드시 생성해야 한다.
- 클래스를 생성하는 방법은 다음과 같다.

```
ClassX    myClass = new ClassX();
```

↑                    ↑                    ↑

클래스이름        변수이름            클래스이름

## 5. Date 클래스

- 우리가 사용할 Date 클래스도 우리가 만들지는 않았지만, 미리 만들어둔 클래스이기 때문에 사용법은 똑같다.

```
Date today= new Date( );
```

- Date 클래스는 편리하게도 그냥 생성하기만 하면 오늘 날짜와 현재시간을 알아내서 우리가 정한 변수(여기서는 today)에 저장하기 때문에, 바로 사용해도 된다.
- 주의할 점은 Date 클래스는 java.util 이라는 패키지 안에 들어 있기 때문에 프로그램 맨 처음 부분에서 import 선언을 해줘야만 한다는 점이다.
- 패키지라는 것은 클래스들의 모임이다.



## 5. Date 클래스

- 자바가 기본적으로 제공하는 클래스는 상당히 많기 때문에 용도에 따라서 몇 개의 패키지로 분류되어 있다.
- 그 중 하나가 java.util 패키지이고, java.util 패키지 내에 있는 Date 클래스를 쓰겠다는 import 선언은 다음처럼 하면 된다.

```
import java.util.Date;
```

- import 명령은 항상 프로그램의 가장 앞에 나와야만 한다.
- Import 명령 앞에 올 수 있는 명령은 import 명령뿐이기 때문에, 프로그램을 작성할 때는 먼저 필요한 패키지를 모두 import하고 시작해야 한다.
- 이 점은 변수를 아무 곳에서나 선언할 수 있는 것과는 다른 점이다.

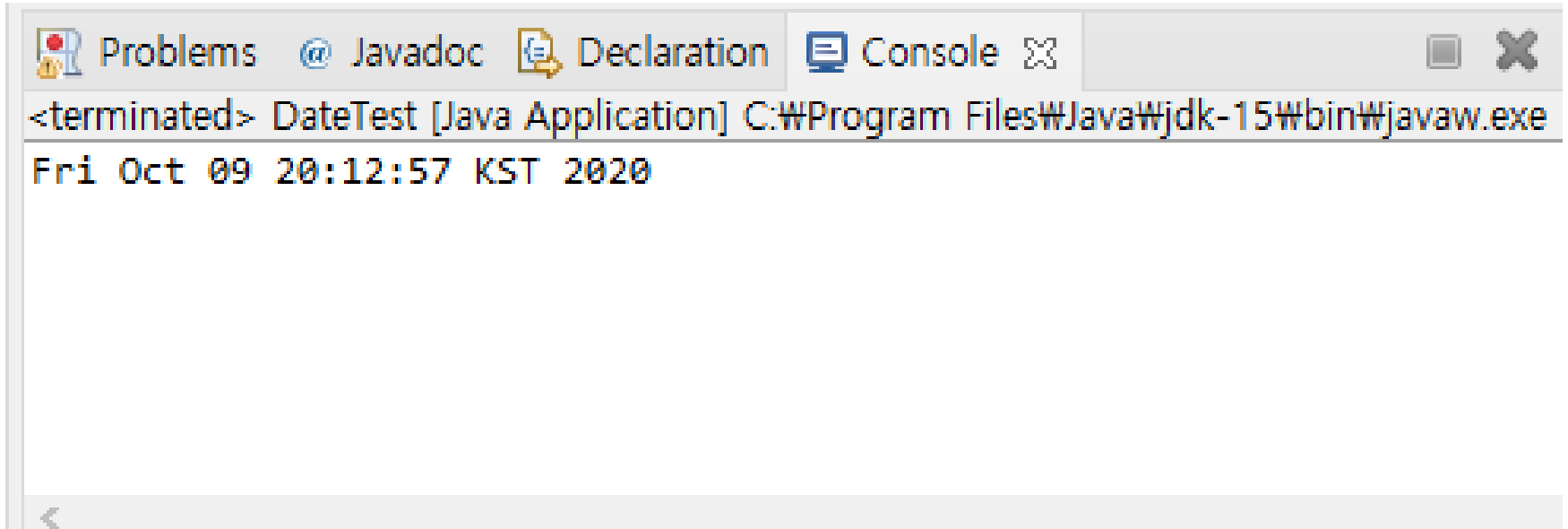
## 5. Date 클래스

- 다음은 오늘 날짜와 현재 시간을 출력하는 예제 프로그램이다.

```
1  import java.util.Date;
2
3  public class DateTest{
4      public static void main(String[] args) {
5          Date today= new Date();
6
7          System.out.println(today);
8      }
9  }
```

## 5. Date 클래스

- 7행에서와 같이 Date 클래스를 생성한 객체인 today를 바로 출력하면 미리 정해진 형식대로 오늘 날짜와 현재 시간을 화면에 출력한다.



The screenshot shows an IDE's console window with the following tabs: Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of a Java application named DateTest. The output consists of two lines: the first line is the path to the Java runtime executable, and the second line is the current date and time in KST.

```
<terminated> DateTest [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe  
Fri Oct 09 20:12:57 KST 2020
```

## 6. SimpleDateFormat 클래스

- 앞의 DateTest 예제 프로그램에서도 알 수 있지만, Date를 생성할 때 얻는 날짜와 현재시간을 사용하기에 너무 불편하다.
- 예를 들어 위 프로그램에서 나온 결과를 “2007년 04월 15일” 식으로 나타내려면 today가 돌려주는 문자열을 하나하나 세어서 년도, 월, 일 등을 얻는 프로그램을 따로 만들어야 한다.
- 요일, 시, 분, 초도 마찬가지이다.
- 자바 구버전(1.0)에서는 Date 클래스 내에 getYear() 메서드 등이 들어 있었지만, 현재 버전에서는 사용이 금지되었다.
- 따라서 자바 최신 버전에서는 원하는 대로 날짜와 시간의 출력을 조정할 수 있는 SimpleDateFormat이라는 클래스를 따로 제공한다.
- SimpleDateFormat 클래스를 생성할 때, 우리가 원하는 출력형식을 미리 정해두고, 생성된 변수에 Date가 돌려준 값을 넣으면 우리가 원하는 모양으로 출력된다.

## 6. SimpleDateFormat 클래스

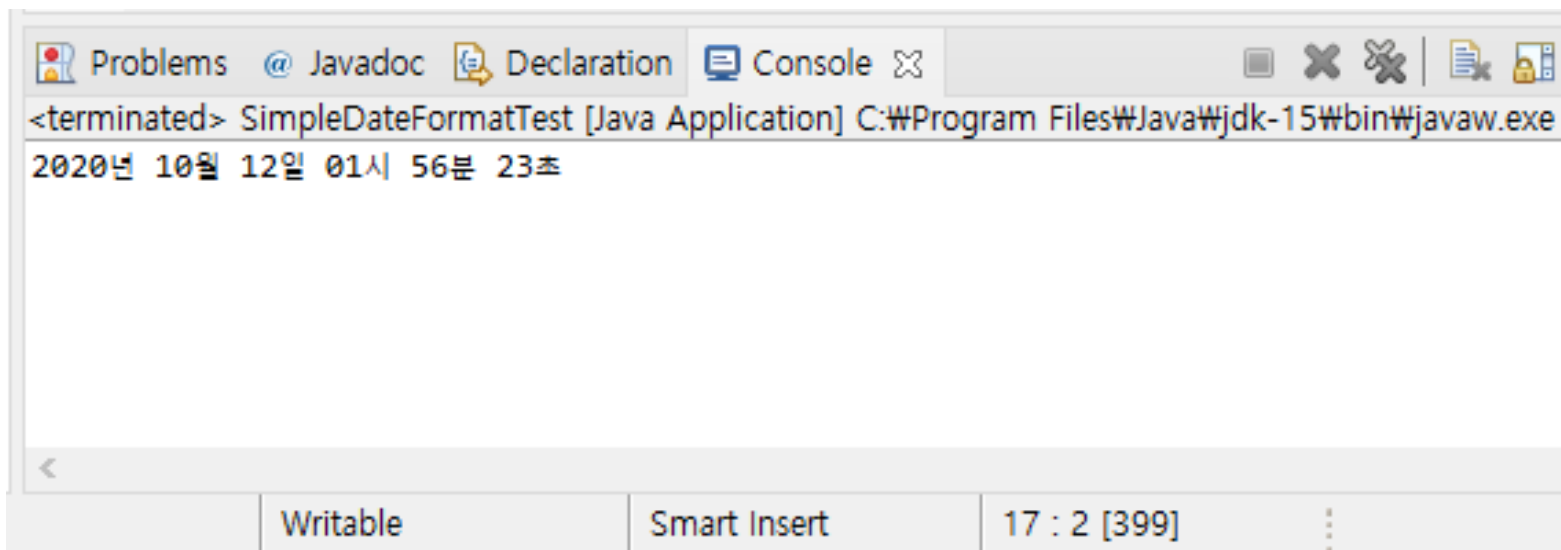
- 예를 들어 '2007년 04월 15일 03시 58분 09초' 형식으로 출력하고 싶다면 다음 예제 프로그램처럼 하면 된다.
- 주의할 점은 Date 클래스에서와 마찬가지로 SimpleDateFormat 클래스도 패키지 `import`를 선언해야 한다는 점이다.
- SimpleDateFormat 클래스는 `java.text` 패키지에 들어 있다.

## 6. SimpleDateFormat 클래스

```
1  import java.util.Date;
2  import java.text.SimpleDateFormat;
3
4  public class SimpleDateFormatTest {
5      public static void main(String[] args) {
6          Date today = new Date();
7
8          // 출력형식을 지정
9          SimpleDateFormat dateForm = new SimpleDateFormat("yyyy년 MM월 dd일
10             hh시 mm분 ss초");
11
12          // 미리 지정한 출력형식에 맞춰서 화면에 출력
13          System.out.println(dateForm.format(today));
14      }
15  }
```

## 6. SimpleDateFormat 클래스

- 10행에서 SimpleDateFormat 클래스를 생성할 때 준 yyyy는 년도, MM은 월, hh는 시, mm은 분 ss는 초로 바뀌어서 출력된다.



- 앞의 예제 프로그램에서 알 수 있듯이 SimpleDateFormat 클래스를 생성할 때 준 yyyy는 년도, MM은 월, dd는 일, hh는 시, mm은 분, ss는 초로 바뀌었다.

## 6. SimpleDateFormat 클래스

- 이렇게 SimpleDateFormat 클래스에서 년도, 월, 일... 등으로 대체되는 대표적인 심벌은 다음의 표와 같다.

심벌	의미	예
Y	년도(year)	1996
M	월(month in year)	07
d	일(day in month)	10
h	시(hour in am/pm (1~12))	12
H	시(hour in day (0~23))	0
m	분(minute in hour)	30
s	초(second in minute)	55
S	1000분의 1초(milliseconds)	978
E	요일(day in week)	Tuesday
a	오전/오후(am/pm marker)	PM



## 7. Calendar 클래스

- 미리 정해진 형식에 따라 날짜나 시간을 출력하는 데는 SimpleDateFormat 클래스가 확실히 편리하다.
- 그러나 변화되는 시간이나 날짜를 알고 싶은 경우에 SimpleDateFormat 클래스는 좀 문제가 있다.
- 시간은 계속 변하기 때문에 참조할 때마다 시스템으로부터 시간을 얻어야만 하기 때문이다.
- 이럴 때는 Calendar 클래스를 사용하면 간편하게 알아낼 수 있다.
- Calendar 클래스는 시스템에서 시간 정보를 얻어야 하기 때문에 getInstance()라는 메서드를 이용해서 생성한다.

```
Calendar now= Calendar.getInstance( );
```

## 7. Calendar 클래스

- Calendar가 일단 생성되면 현재 년, 월, 일, 요일, 시, 분, 초 등의 정보를 통해 얻을 수 있기 때문에, get() 메서드에 다음의 표의 값을 전달하면 해당하는 정보를 얻을 수 있다.
- 클래스에 든 메서드를 사용하는 방법은 다음처럼 클래스 객체 뒤에 점을 찍고 메서드 이름과 인수를 적으면 된다.

```
점(dot)      인수  
  ↓          ↓  
myClass.myMethod(x);  
  ↑          ↑  
클래스이름  메서드이름
```

## 7. Calendar 클래스

### ■ Calendar 클래스의 상수

YEAR	년도(year)	1996
MONTH	월(month in year)	07
DATE	일(day in month)	10
HOURL	시(hour in am/pm (1~12))	12
HOUROFDAY	시(hour in day (0~23))	0
MINUTE	분(minute in hour)	30
SECOND	초(second in minute)	55
MILLISECOND	1000분의 1초(milliseconds)	978
DAYOFWEEK	요일(day in week)	Tuesday
AMPM	오전/오후(am/pm marker)	PM

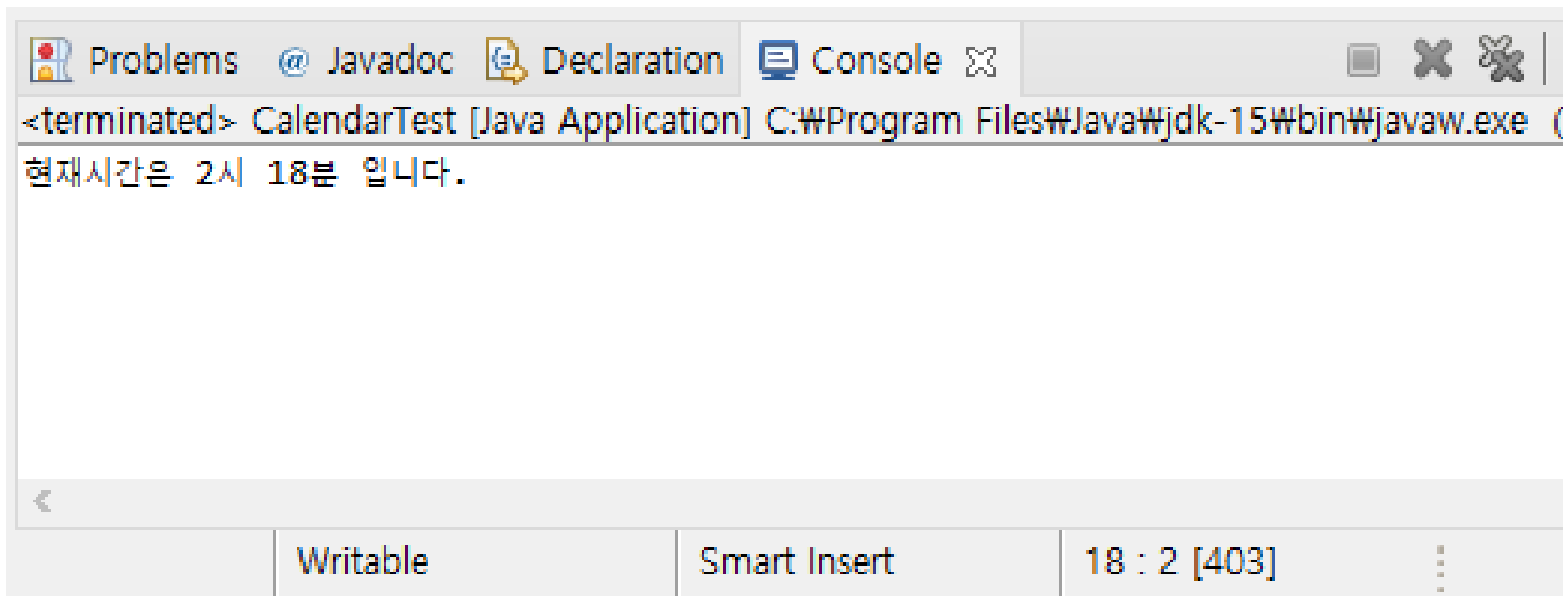
## 7. Calendar 클래스

■ 다음은 Calendar 클래스를 사용해서 현재시간을 출력하는 예제 프로그램이다.

```
1  import java.util.Calendar;
2
3  public class CalendarTest {
4      public static void main(String[] args) {
5          // 반드시 getInstance() 메서드로 생성
6          Calendar now = Calendar.getInstance();
7
8          // get() 메서드로 시간 정보 얻기 선언
9          int hour = now.get(Calendar.HOUR);
10         int min = now.get(Calendar.MINUTE);
11
12         System.out.println("현재시간은 " + hour + "시 " + min + "분 입니다.");
13     }
14 }
```

## 7. Calendar 클래스

- Calendar 클래스를 사용하면 클래스 생성 시점과 상관없이 현재의 날짜, 시간 정보를 얻을 수 있습니다.
- 따라서 결과에 나타난 시간은 7행 또는 9행, 10행의 명령이 실행된 시간이 아니고, 12행의 명령이 실행되는 현재 시간입니다.



The screenshot shows an IDE's console window with the following tabs: Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of a Java application named 'CalendarTest'. The output text is: `<terminated> CalendarTest [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (현재시간은 2시 18분 입니다.)`. The status bar at the bottom indicates 'Writable', 'Smart Insert', and '18 : 2 [403]'.

```
<terminated> CalendarTest [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (
현재시간은 2시 18분 입니다.
```

## 8. 난수를 구하는 Random 클래스

- 게임을 만드는데 난수는 반드시 필요하다.
- 비행기를 맞추는 슈팅게임에서 날아다니는 비행기가 일정하게 움직이면 정말 재미없을 것이다.
- 몇 가지 종류의 비행 패턴이 있더라도, 매번 다른 순서로 움직이는 것이 필요하다.
- 또, 카드 게임에서 나오는 카드의 순서가 똑같으면, 한두 번만 하면 순서를 외우지 않을까?
- 이런 일을 방지하기 위해서 많이 쓰는 방법 중의 하나는 현재시간을 이용하는 것이다.
- 예를 들어 현재시간이 4시 47분 25초라면 각 숫자를 더하거나 빼서 어떤 숫자를 만들고 그 숫자를 이용해서 게임을 진행하는 것이다.

## 8. 난수를 구하는 Random 클래스

- 그러나 현재시간을 이용하는 방법은 전통적으로 많이 사용된 방법이지만, 공교롭게도 현재시간에서 구한 값이 같아지거나 특정한 숫자가 더 많이 나오는 경우가 생기는 등 문제가 있었다.
- 무엇보다도 따로 프로그램을 만들어서 쓴다는 것도 불편한 점 중 하나였습니다.
- 그래서 자바에서는 난수를 만드는데 쓰는 Random 클래스를 미리 준비하여 제공하고 있다.
- Random 클래스를 사용하면 바로 복잡한 프로그램을 만들거나 고민할 것 없이 간단히 난수를 만들 수 있다.
- 우선 Random 클래스가 들어있는 패키지가 java.util이기 때문에, Date나 Calendar 클래스에서 했던 것처럼 java.util 패키지 내의 Random 클래스를 impor하겠다고 선언하면 된다.

```
import java.util.Random;
```

## 8. 난수를 구하는 Random 클래스

- Random 클래스에는 nextInt()라는 메서드가 들어 있다.
- Random 클래스를 생성한 후, 이 nextInt() 메서드를 부르면 -2147483648 ~ 2147483647 사이의 숫자 중 하나를 돌려준다.
- 이 값을 적절히 고쳐 쓰면 된다.
- 예를 들어 -9 ~ 9 사이의 값을 만들고 싶으면, %(나머지(mod) 연산자)를 이용해서 다음과 같이 작성하면 된다.

```
Random r = new Random();  
int randomNum = r.nextInt() % 10;
```



## 8. 난수를 구하는 Random 클래스

- 만약 0 ~ 99와 같이 양의 값(+)으로만 제한하고 싶으면, 다음의 예제 프로그램에서처럼 Math 클래스의 abs() 메서드를 쓰면 된다.
- abs() 메서드는 주어진 값을 양수로 바꿔 돌려주는 절댓값 함수이다.

```
1 import java.util.Random;
2
3 public class RandomTest {
4     public static void main(String[] args) {
5         // Random 클래스의 객체 생성
6         Random r = new Random();
7
8         // 난수 값을 100으로 나눈 나머지를 양수로 바꿈
9         System.out.println("0~99 범위의 난수: " + Math.abs(r.nextInt() % 100));
10    }
11 }
```

## 8. 난수를 구하는 Random 클래스

- Random 클래스의 객체 r이 돌려주는 값은 -2147483648 ~ 2147483647 사이의 숫자 중 하나이다.
- 이 수를 100으로 나눈 나머지는 -99 ~ 99 사이의 숫자 중 하나가 된다.
- abs() 메서드로 양수로 바꾸면, 결과는 0-99 사이의 숫자 중 하나이다.
- 만일 결과로 1 ~ 100 사이의 값을 얻고 싶다면, 이 식의 결과에 +1을 더하면 된다.
  - `Math. abs(r.nextInt() % 100))+1`

## 9. 오늘의 운세 게임 만들기

- 앞에서 우리는 자바의 기본 데이터형과 변수와 상수를 사용하는 방법에 대해서 배웠다.
- 또, 날짜와 시간을 구하고 난수를 만드는 방법에 대해서도 배웠다.
- 배운 것들을 종합해서 오늘의 날짜와 오늘의 금전운을 퍼센트(%)로 알려주는 게임을 만들면 다음과 같다.

## 9. 오늘의 운세 게임 만들기

```
1  import java.util.Date;
2  import java.util.Random;
3  import java.text.SimpleDateFormat;
4
5  public class GameJava2_02 {
6      public static void main(String[] args) {
7          // Date 클래스의 객체 생성
8          Date today = new Date();
9
10         // 오늘 날짜를 어떻게 출력할 것인지를 나타내는 출력형식
11         SimpleDateFormat dateForm = new SimpleDateFormat("yyyy년 MM월 dd일
12         의 ");
13
14         // print()는 문자열을 출력하고 줄을 바꾸지 않는 명령
15         System.out.print(dateForm.format(today));
```

## 9. 오늘의 운세 게임 만들기

```
16      Random r = new Random();
17      // 100으로 나눈 나머지의 양수에 1을 더한 값
18      int randomNum = Math.abs(r.nextInt() % 100) + 1;
19
20      // println()은 문자열을 출력하고 줄을 바꾸는 명령
21      System.out.println("금전운(100): " + randomNum + "%");
22  }
23 }
```



**Thank You**

---