Part 1. 자바 언어 기초

Chapter 01. 자바 시작하기

1.1 프로그래밍 언어와 자바

- 고급 언어 : 컴퓨터와 대화할 수 있도록 만든 언어 중 사람이 쉽게 이해할 수 있는 언어. 컴파일(compile)이라는 과정을 통해 컴퓨터가 이해할 수 있는 0과 1로 이루어진 기계어로 변환한 후 컴퓨터가 사용.
- 저급 언어 : 기계어에 가까운 언어. 대표적으로 어셈블리어.

자바 : 소스 파일 -> 바이트 코드 파일 -> JVM 해석 -> 실행
(파이썬 : 소스파일 -> 결과. 인터프리터가 한 줄 한 줄 실행)

자바는 1995년 썬 마이크로시스템즈에서 발표한 후 가장 성공한 프로그래밍 언어로서 전세계적으로 다양한 분야에서 사용되고 있다. 오라클에서 라이선스를 가지고 있는데, 오라클은 *자바 개발 도구(Java Development Kit, JDK)* 를 배포하여 자바로 프로그램을 쉽게 개발할 수 있도록 기술적 지원을 하고 있다.

자바의 특징

- 모든 운영체제에서 실행 가능
  - 자바로 작성된 프로그램은 운영체제와 상관없이 모두 실행되기 때문에 윈도우에서 개발된 프로그램을 수정 없이 바로 맥OS 또는 리눅스에서도 실행할 수 있다.
- 객체 지향 프로그래밍
  - 먼저 객체(부품)를 만들고 이 객체들을 서로 연결해서 더 큰 프로그램을 완성시키는 기법을 객체 지향 프로그래밍(Object Oriented Programming, OOP)이라고 한다. 자바는 OOP를 위한 최적의 언어이다.
- 메모리 자동 정리
  - 자바는 메모리(RAM)를 자동 관리하므로, 개발자는 핵심 기능 작성에 집중할 수 있다.
- 무료 라이브러리 풍부
  - 무료 사용이 가능한 오픈 소스 라이브러리가 풍부하기 때문에 프로그램 개발 기간을 단축시켜준다.

1.2 운영체제별 JDK 설치

자바 프로그램을 개발하고 실행하기 위해서는 먼저 Java Se(*Standard Edition*)의 구현체인 JDK를 설치해야 한다. JDK에는 Open JDK와 Oracle JDK가 있으나 비용을 고려한다면 Open JDK를 사용하는 것이 유리하다. JDK LTS (*Long Term Suppot*) 버전은 장기간 기술 지원을 받을 수 있어 안정적이며 JDK 8, JDK 11, JDK 17로 개발 및 실행하는 것이 좋다. (책에서는 Open JDK 17 버전 사용)

1.3 윈도우 환경 변수 설정

운영체제는 프로그램들이 실행하면서 사용할 수 있는 값들을 환경 변수 이름으로 관리한다. JDK를 설치하면 프로그램들이 JDK를 이용할 수 있도록 JAVA_HOME 환경 변수를 생성하고, Path 환경 변수를 수정하는 것이 좋다.

1.4 맥OS 환경 변수 설정

맥OS에 JDK를 설치했다면 위치는 /Library/Java/JavaVirtualMachines/temurin-17.jdk가 된다.
JavaVirtualMachines 디렉토리에 temurin-17.jdk만 있다면 환경 변수 설정이 필요 없다. 만약 다른 JDK가 존재한다면 temurin-17.jdk를 사용하도록 환경 변수를 설정해야 한다.

1.5 바이트코드 파일과 자바 가상 머신

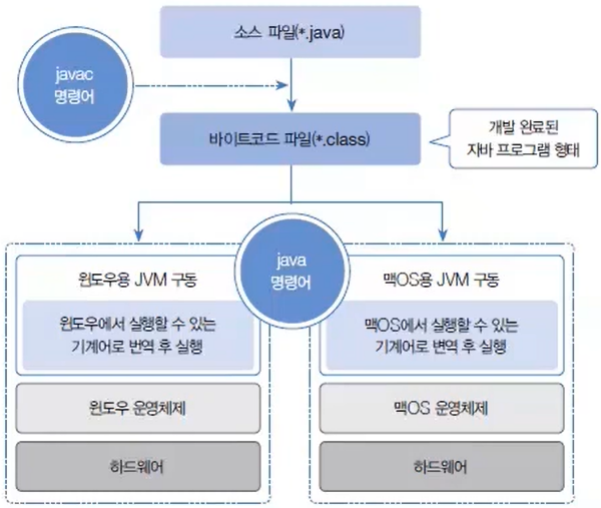
JDK를 설치했다면 자바 언어로 작성된 소스 파일을 만들고 컴파일할 수 있다. 자바 소스 파일의 확장명은 **.java**이다. 텍스트 파일이므로 어떤 텍스트 에디터에서도 작성이 가능하다.

바이트 코드 파일

소스 파일(.java)을 작성한 후에는 컴파일을 해야 한다. **javac(java compiler)** 명령어는 소스 파일을 컴파일하는데, 컴파일 결과는 확장명이 **.class**인 바이트코드 파일로 생성된다. 윈도우, 맥OS, 리눅스 등 어떤 운영체제라 하더라도 동일한 소스 파일을 javac로 컴파일하면 모두 동일한 바이트코드 파일이 생성된다.

자바 가상 머신

바이트코드 파일(~*.class*)을 특정 운영체제가 이해하는 기계어로 변역하고 실행시키는 명령어는 **java**이다. java 명령어는 JDK와 함께 설치된 자바 가상 머신 *JVM*을 구동시켜 바이트 코드 파일을 완전한 기계어로 번역하고 실행시킨다. 바이트코드 파일은 운영체제와 관계없이 모두 동일한 내용으로 생성되지만, 자바 가상 머신은 운영체제에서 이해하는 기계어로 번역해야 하므로 운영체제별로 다르게 설치된다. 그래서 운영체제별로 설치하는 JDK가 다른 것이다.



1.6 소스 작성부터 실행까지

1. <사용자 홈>에서 temp 디렉토리를 다음 구조로 생성하고, Hello.java 소스 파일을 생성한다.

```
temp
|-- src                          <- 소스 파일이 저장되는 디렉토리
|   |-- ch01
|       |-- sec06                <- 패키지 디렉토리
```

```
|           |-- Hello.java          <- 소스 파일
|
|-- bin                             <- 바이트코드 파일이 저장되는 디렉토리
```

패키지란?
자바는 소스 파일 및 컴파일된 바이트코드 파일을 쉽게 관리하기 위해 패키지(package)를 사용한다. 패키지는 마치 파일 시스템의 디렉토리와 비슷하다.

2. Hello.java를 텍스트 에디터에서 열고, 코드를 작성한다.

```
package ch01.sec06;                                 // 바이트코드 파일이 위치할 패키지 선언

public class Hello {                                // Hello 클래스 선언
    public static void main(String[] args) {        // main() 메소드 선언
        System.out.println("Hello, Java");          // 콘솔에 출력하는 코드
    }
}
```

3. 소스 파일을 javac 명령어로 컴파일한다.

javac -d [바이트코드파일저장위치] [소스경로/*.java]
(**javac -d** bin src/ch01/sec06/Hello.java)
=> temp/bin 디렉토리에 바이트코드 파일(ch01/sec06/Hello.class)이 생성

4. java 명령어로 바이트코드 파일을 기계어로 번역하고 실행한다.

java -cp [바이트코드파일위치] [패키지...클래스명]
java -cp bin ch01.sec06.Hello
=> 콘솔에 Hello, Java가 출력

1.7 이클립스 설치

- 현재 기업체에서 가장 많이 사용하는 *통합 개발 환경 (Integrated Development Environment, IDE)*

1.8 이클립스 프로젝트 생성

1.9 이클립스 소스 작성부터 실행까지

1.10 코드 용어 이해

- 패키지 선언

```
package ch01.sec09;
```

소스 파일이 *src/ch01/sec09* 패키지에 있으며 컴파일 후 생성되는 바이트코드 파일도 *bin/ch01/sec09* 패키지에 생성된다.

- 클래스 선언

```
public class Hello { ... }
```

Hello는 *클래스명*으로 소스 파일명과 대소문자가 완전히 일치해야 한다. { ... } 는 *클래스 블록*이라고 부른다.

- main() 메소드

```
public static void main(String[] args) { ... }
```

{ ... } 는 *main() 메소드 블록*이라고 한다. 바이트코드 파일을 실행하면 이 main() 메소드 블록이 실행되어 *프로그램 실행 진입점(entry point)* 이라고 부른다.

1.11 코드 주석 달기

주석은 프로그램 실행과는 상관없이 코드에 설명을 붙인 것이다. 이는 컴파일 과정에서 무시되므로 주석을 많이 작성한다고 해서 바이트코드 파일의 크기가 커지는 것은 아니다.

구분	주석 기호	설명
행 주석	// ...	//부터 행 끝까지 주석으로 처리
범위 주석	/* ... */	/*와 */ 사이에 있는 내용은 모두 주석으로 처리
도큐먼트 주석	/** ... */	/**와 */ 사이에 있는 내용은 모두 주석으로 처리 javadoc 명령어로 API 도큐먼트를 생성하는데 사용

(+) 주석 기호는 문자열("") 내부에서는 작성될 수 없다.

1.12 실행문과 세미콜론

main() 메소드 블록 내부에는 다양한 실행문이 작성되며, 실행문은 변수 선언, 변수값 저장, 메소드 호출에 해당한다.
실행문 끝에는 반드시 세미콜론(;)을 붙여야 하며 그렇지 않을 경우 컴파일 에러가 발생한다.

```
int result =
x + y;

int x = 1; int y = 2;
```

Chapter 02. 변수와 타입

2.1 변수 선언

- 변수(variable) : 하나의 값을 저장할 수 있는 메모리 번지에 붙여진 이름.

1. 자바 소스 파일명(클래스명)은 대문자로 시작하는 것이 관례
   Week.java | *MemberGrade.java* | *ProductKind.java*

2. 변수명은 소문자로 시작하는 것이 관례
   score | *MathScore* | *sportsCar*

변수에 최초로 값을 대입하는 행위를 **변수 초기화**라고 하고, 이때의 값을 **초기값**이라고 한다. 초기화되지 않은 변수는 아직 메모리에 할당되지 않았기 때문에 변수를 통해 메모리 값을 읽을 수 없다.

2.2 정수 타입

값의 분류	**기본 타입**
정수	byte, char, short, int, long
실수	float, double
논리(true/false)	boolean

종류	**byte**	**short**	**int**	**long**
메모리 사용 크기(단위 bit)	8	16	32	64

- byte, short, int, long은 모두 부호 있는(signed) 정수 타입이므로 최상위 bit는 부호 bit로 사용되고, 나머지 bit는 값의 범위를 정한다.
- **리터럴(literal)** : 코드에서 프로그래머가 직접 입력한 값. 변수에 대입할 정수 리터럴은 진수에 따라 작성하는 법이 다르다.
  - 2진수 : 0b 또는 0B로 시작하고 0과 1로 작성
  - 8진수 : 0으로 시작하고 0~7 숫자로 작성
  - 10진수 : 소수점이 없는 0~9 숫자로 작성
  - 16진수 : 0x 또는 0X로 시작하고 0~9 숫자나 A, B, C, D, E, F 또는 a, b, c, d, e, f 로 작성
- long 타입은 수치가 큰 데이터를 다루는 프로그램에서 사용된다. (은행, 과학 분야...) 기본적으로 컴파일러는 정수 리터럴을 int 타입 값으로 간주하기 때문에, int 타입의 허용 범위를 초과하는 리터럴은 뒤에 소문자 'l'이나 대문자 'L'을 붙여 long 타입 값임을 컴파일러에게 알려줘야 한다.

```
long var = 1000000000000L; // 'L' 없을 시 에러 발생
```

2.3 문자 타입

하나의 문자를 작은따옴표(')로 감싼 것을 *문자 리터럴*이라고 하며 이는 유니코드로 변환되어 저장된다. 여기서 유니코드는 세계 각국의 문자를 0~65535 숫자로 매핑한 국제 표준 규약이다. 유니코드가 정수이므로 char 타입도 정수 타입에 속한다. 그렇기에 문자가 아닌 유니코드 숫자를 직접 대입할 수도 있다.

초기화 시 ''와 같은 빈 문자를 대입하면 컴파일 에러가 발생하므로 공백(유니코드:32) 하나를 포함해서 초기화해야 한다.

2.4 실수 타입

타입	**메모리크기**

타입	메모리크기
float	4byte | 32bit
double	8byte | 64bit

- 컴파일러는 실수 리터럴을 기본적으로 double 타입으로 해석하므로 float 타입에 대입하고 싶다면 리터럴 뒤에 소문자 'f' 또는 'F'를 붙여야 한다

2.5 논리 타입

2.6 문자열 타입

2.7 자동 타입 변환

- 값의 허용 범위가 작은 타입이 허용 범위가 큰 타입으로 대입될 때 발생한다.
- char 타입의 경우 int 타입으로 자동 변환되면 유니코드 값이 int 타입에 대입된다.

2.8 강제 타입 변환

- 작은 허용 범위 타입 = (작은 허용 범위 타입) 큰 허용 범위 타입

2.9 연산식에서 자동 타입 변환

- 정수 타입 변수가 산술 연산식에서 피연산자로 사용되면 int 타입보다 작은 byte, short 타입의 변수는 int 타입으로 자동 타입 변환되어 연산을 수행한다.

```
byte x = 10;
byte y = 20;
byte result = x + y;    // 컴파일 에러
int result = x + y;
```

byte 변수 x, y가 피연산자로 사용되면 변수값은 int 타입으로 변환되어 연산되며 결과도 int 타입으로 생성된다.

- long 타입이 피연산자로 사용되면 다른 피연산자는 long 타입으로 변환되어 연산을 수행한다. 따라서 연산 결과는 long 타입 변수에 저장해야 한다.

- 피연산자 중 하나가 double 타입이면 다른 피연산자도 double 타입으로 변환되어 연산되고, 연산 결과 또한 double 타입이 된다.

```
int x = 1;
int y = 2;
double result = x / y;
```

위 코드를 실행하면 0.5가 아닌 0.0이 출력되는데, 자바에서 정수 연산의 결과는 항상 정수가 되기 때문이다.
(x/y -> 0 -> 0.0)

0.5가 되기 위해서는 x와 y 둘 중 하나 또는 둘 모두를 double 타입으로 변환해야 한다.

- + 연산자는 피연산자가 모두 숫자일 경우 덧셈 연산을, 피연산자 중 하나가 문자열일 경우 나머지 피연산자도 문자열로 자동 변환하여 문자열 결합 연산을 수행한다.

```
int value = 1 + 2 + 3;      // 6
String str = 1 + 2 + "3";   // 33
String str = 1 + "2" + 3;   // 123
String str = "1" + 2 + 3;   // 123
```

2.10 문자열을 기본 타입으로 변경

- String -> int
  ```
  String str = "300000;
  int value = Integer.parseInt(str);
  ```
- int -> String
  ```
  String str = String.valueOf(10);
  ```

2.11 변수 사용 범위

main() 메소드 블록에는 다른 중괄호 {} 블록들이 작성될 수 있는데 이러한 중괄호 블록 내에서 선언된 변수는 중괄호 내에서만 사용이 가능하고 밖에서는 사용할 수 없다.
메소드 블록 전체에서 사용하고 싶다면 메소드 블록 첫머리에 선언하는 것이 좋고, 특정 블록 내부에서만 사용된다면 해당 블록 내에서 선언하는 것이 좋다.

2.12 콘솔로 변수값 출력

```
public class PrintfExample {
  public static void main(String[] args) {
    int value = 123;
    System.out.printf("%d원", value);       // 123원
    System.out.printf("%6d원", value);      //    123원 (왼쪽 공백 3개)
    System.out.printf("%-6d원", value);     // 123   원 (오른쪽 공백 3개)
    System.out.printf("%06d원", value);     // 000123원 (공백 대신 0)

    double area = 314.159;
    System.out.printf("%10.2f", area); //     314.16 (정수 7자리+소수점+왼쪽 빈자리 공백)

    String name = "홍길동";
    String job = "도적";

    System.out.printf("%-5s | %5s", name, job); // 홍길동  |    도적
  }
}
```

2.13 키보드 입력 데이터를 변수에 저장

```
Scanner scanner = new Scanner(System.in);
String inputData = scanner.nextLine();
```

scanner.nextLine()은 Enter 키가 입력되기 전까지 블로킹(대기) 상태가 되며, Enter 키가 입력되면 지금까지 입력된 모든 내용을 문자열로 읽는다.