Java의 구성요소

구문법(Syntax)

●구성요소를 이용하여 문장/프로그램을 구성하는 방법

●문법을 이용하여 기술할 수 있음

Context-free grammar, BNF(Backus-Naur Form)

●어휘 구조 (Lexical Structure)

언어에서 사용하는 단어의 구조, 철자법 등을 의미

의미론(Semantics)

●문장/프로그램의 의미를 정하는 것

●자연어 혹은 수학적으로 기술

이것은 프로그램 언어를 공부하는데 필요한 기본기이다.

프로그램 언어의 처리 기법

프로그래밍 언어 처리 과정은 총 3개의 단계로 진행된다.

1. 입력 프로그램으로 받은 소스 프로그램을 구문법에 맞는지 검사한다.
2. 그 의미에 맞게 동작하도록 해석하거나 기계어 명령어들로 번역해야한다.
3. 컴퓨터가 기계어를 해석하고 실행을 한다.

고급언어 처리 기법에는 3가지가 있다.

1. 컴파일러 기법

고급 언어로 작성된 프로그램을 컴퓨터가 바로 실행할 수 있는 프로그램으로 변환하는 방식.

번역이 완료되면 빠르게 프로그램을 실행시킬 수 있는 장점

컴파일 하는 프로그램은 컴파일러이다.

1. 해석 기법

고급 언어로 작성된 프로그램을 바로 실행

코드를 한 단계씩 해석하여 실행하며, 속도가 느리다.

해석하는 프로그램은 인터프린터다.

1. 하이브리드 기법

컴파일 기법과 해석 기법을 혼합한 상태이다.

고급 언어로 작성된 프로그램을 쉽게 해석할 수 있도록 중간 코드 형태로 번역한다.

“Java”

byteode라고 하는 중간 코드로 번역한다.

운영체제마다 별도로 존재하는 자바가상머신이 Byte Code 실행

선마이크로시스템즈의 제임스 고블링이 만듦

Java의 활용 분야

웹 어플리케이션은 물론 기업용 어플리케이션에 이용됨

빅데이터, 소셜, 클라우드 등 혁신 기술에서 업무 전반에 밀접하게 연관되어 있음

Java의 특징

단순성

배우기 쉬움

국내외 수 많은 개발자 사용 – 인기 프로그래밍 언어

플랫폼 독립성

다른 프로그래밍 언어는 플랫폼 종속

하드웨어 아키텍처에 따라 사용하는 기계어 종류가 다름

운영체제마다 사용하는 응용 프로그래밍 인터페이스와 실행 파일 형식이 다름

프로그램을 실행하려면 운영체제가 제어하는 메모리를 사용해야 하는데, 운영체제마다 메모리를 관리하는 방식이 다음.

객체 지향 언어

가비지 컬렉션

멀티스레드 지원/분산 처리 지원/견고성/안전성 좋음

* **Java Program Basic Structure**
* public class /\* 클래스를 정의하는 키워드 \*/ Hello /\* 클래스 이름 \*/ { /\* 클래스 시작 \*/
* public static void /\* 메소드를 실행한 후 반환 값이 없음을 의미 \*/ main /\* 메소드 이름 \*/ ( String[] args /\* 메소드의 매개변수 타입과 매개변수 \*/ { /\* 메소드 본체 시작 \*/
* // TODO Auto-generated method stub /\* 주석 \*/
* System.out.println("안녕하세요!") /\* 실행문 \*/
* } /\* 메소드 본체 끝 \*/
* } /\* 클래스 끝 \*/
  + 식별자(Identifier)

보고 느낄 수 있는 모든 사물(객체)들을 각각 구별할 수 있는 것을 의미

* + - 규칙
      * 문자, 언더바(\_), $로 시작해야 한다.
      * 한글도 가능하며, 영문자는 대, 소문자를 구분해야한다.
      * +, - 등 연산자를 포함하면 안된다.
      * Java Keyword를 사용하면 안 된다.
      * 길이에 제한이 없다.
    - 관례 = 카멜 표기법 (Camel Case)
      * 변수와 메서드는 모두 소문자로 표기

단, 복합 단어일 때는 두 번째 단어부터 단어의 첫 글자만 대문자로 표기

* + - * 클래스와 인터페이스는 첫 자만 대문자로 표기하고 나머지는 소문자로 표기

단, 복합 단어일 때는 두 번째 단어부터 단어의 첫 글자만 대문자로 표기

* + - * 상수는 전체를 대문자로 표기

단, 복합 단어일 때는 단어를 언더바(\_)로 연결

* + 변수(Variable)
    - 프로그램은 메모리에 데이터를 보관, 각 메모리를 변수(Variable)로 구분
    - 변수는 데이터를 담는 상자와 같은 것으로 종류가 다양하며 이를 구분하기 위해 데이터 타입을 사용
  + 데이터 타입(Data Type)

타입이란 데이터의 형태로 변수가 이름 붙여진 상자라면, 타입은 상자에 담길 내용물의 종류

* + - 값과 값을 다룰 수 있는 연산의 집합을 의미

Java에서는 기본 타입 & 참조 타입을 지닌다.

* + - 기본 타입(Primitive Type)

값을 직접 갖는 데이터 형태로 기본 타입은 정수, 실수, 문자, 논리 타입으로 나뉜다.

* + - * 기억 공간 크기 및 기본값 정수와 실수는 다시 세부 타입으로 나뉘는데, 이는 기억 공간 크기인 bit가 기준 (따라서 숫자를 표현할 때는 타입의 표현 범위를 주의)
    - 참조 타입 (Reference Type)

주소값을 갖는 데이터 형태로 문자열이 대표적이며 그 외 배열형, 클래스형, 인터페이스형, 열거형 등이 있다.

* + 연산자 (Operator)

데이터 연산을 위한 기호

Java는 다양한 연산자가 제공된다.

* + - 대입 연산자

오른쪽 값을 왼쪽 변수에 저장하는 기호

* + - 산술 연산자

수치 계산을 위한 연산자

* + - 증감 연산자

변숫값을 1만큼 증가 또는 감소

값의 증가는 ++ 기호를, 감소는 -- 기호를 사용 변수의 앞에 있는 경우에는 전위 연산, 뒤에 있는 경우엔 후위 연산을 수행 문장의 구분은 세미콜론(;)을 기준

* + - 비교 연산자

논리 타입을 제외한 기초 타입에만 사용할 수 있지만, ==와 !=는 모든 기초 타입에 사용

* + - 논리 연산자

피연산자의 조건을 결합해서 true와 false를 조사하며, 논리 타입에만 사용

* Operator Precedenc(연산자 우선순위)
  + 변수의 활용
    - 변수 초기화

변수는 생성과 동시에 초기화될 수 있으며 오류 방지를 위해 초기화 작업이 필요

* + - Const(상수)

프로그램 실행 중 변경할 수 없는 데이터를 담는 변수 ex) 원주율 값(3.14159), 빛의 속도(3×108m/s), …

final 키워드를 이용하여 지정 가능

* + - var 예약어

Java 10부터 지원

초깃값을 통하여 데이터 타입 추론 가능하며 식별자로 사용 가능

* + - 변수의 복사 및 문자열 연결 문자열끼리 연결 및 문자열 & 숫자 연결 가능
  + 기본 입출력
    - 키보드로 데이터 입력 & 화면에 데이터 출력
    - 화면에 데이터 출력
      * System.out.println() : 내용을 출력한 후 행을 바꾼다.
      * System.out.print() : 내용을 출력만 하고 행은 바꾸지 않는다.
      * System.out.printf() : 포맷을 지정(형식 문자열)해서 출력
    - 형식 문자열 출력
      * %s : 해당 위치에 String 값을 삽입
      * %d : 해당 위치에 int 값을 삽입
      * %f : 해당 위치에 double 값을 삽입
      * %b : 해당 위치에 boolean 값을 삽입
  + 타입 변환
    - 숫자 연산의 결과 타입
      * 정수와 정수의 연산 > 정수
      * 정수와 실수의 연산 > 실수
    - 타입 불일치

대입할 값과 변수의 타입이 서로 다른 경우, 문제가 생길 수 있다.

이때 해결 방법은 변수와 대입값의 타입을 일치시켜야 한다.

* + - 타입 변환과 Casting(캐스팅)
      * 자동 타입 변환 : 암묵적으로 타입 변환 작은 타입의 데이터는 같은 범주의 큰 데이터 타입으로 자동 변환된다. ex) 정수는 실수로 자동 변환 : 실수는 정수 뿐 아니라 소수점 이하의 정보도 포함 할 수 있기 때문이다!
      * 강제 타입 변환 : 명시적으로 타입을 강제 변환 타입을 강제로 바꾸기 위해 Casting을 사용
      * Casting(캐스팅) 같은 범수인 데이터의 해석을 달리하는 문법으로, 소괄호 사이에 해석 타입을 명시
        + Down-casting(다운 캐스팅) : 넓은 범위에서 좁은 범위로 해석
        + Up-casting(업 캐스팅) : 좁은 범위에서 넓은 범위로 해석
    - 문자열과 숫자 간 타입 변환
      * 문자열끼리의 덧셈 연산 > 숫자 계산이 아닌 문자열 연결이 되므로 이를 위해서는 문자열과 숫자(정수 & 실수) 간 변환을 위한 메소드를 사용!
        + 문자열 > 숫자 변환

Integer.parseInt() : 문자열을 정수로 변환

Double.parseDouble() : 문자열을 실수로 변환

* + - * + 숫자 > 문자열 변환

Integer.toString() : 정수를 문자열로 변환

Double.toString() : 실수를 문자열로 변환

* **Method(메소드)**

입력값에 따른 결과를 반환하는 일련의 코드 집합으로 간단한 호출만으로도 여러 줄의 코드를 수행할 수 있다.

* + 동작 원리

호출부와 정의부로 나뉘어 동작

* + - 호출부: Method를 사용하는 영역, 입력 값을 전달하여 그 결과를 돌려받는 코드
    - 정의부 : 입력값을 받아 결과를 만드는 일련의 코드 집합

Method를 만드는 것을 정의(define)라 하고, 이를 사용하는 것을 호출(call)이라 한다!

* + 구조
  + public static TYPE NAME PARAMETERS {
  + ...
  + return VALUE;
  + }
    - Method name(메소드 이름) : Method의 이름으로, 호출 시 사용
    - Parameters(매개변수) : 실행 시 입력값을 저장하는 변수, 입력변수 혹은 파라미터라고도 부른다.
    - Return value(반환값) : 최종 결과로, 반환될 값을 의미
    - Return type(반환 타입): 반환값의 타입을 의미
  + 호출
  + 메소드명(입력값1, 입력값2, ...)
    - 호출과 반환 과정

호출하면 제어가 호출된 Method(callee)로 넘어갔다가, 실행을 마친 후 호출한 Method(caller)로 다시 돌아온다. 단, return 문을 사용하면 실행 도중에도 호출한 Method로 제어를 넘길 수 있다.

* + - 매개변수 종류
      * formal parameter(형식 매개변수) : 선언에서 사용된 매개변수
      * actual parameter(실 매개변수) : 호출에서 사용된 매개변수로 argument(인자)라고도 한다.
    - 매개변수 전달

호출 시에 실 매개변수와 형식 매개변수를 매칭하는 것

* + - * 전달 방식 종류
        + pass by value(값 전달)
        + pass by reference(참조 전달)
        + pass by value-result(값-결과 전달)
        + pass by name(이름 전달)
* 유형
  + 매개변수의 유무?

매개변수가 없는 유형, 단일 매개변수, 다중 매개변수

* + 변환값의 유무?
    - void

Method의 반환값이 존재하지 않는 메소드로

return type은 void가 되고, return keyword는 생략 가능

* **주석문(commentary)**
  + 행 주석 : //
  + 범위 주석 : /\* \*/
  + 문서 주석 : /\*\* \*/
* **제어문(control statement)**

프로그램의 흐름에 영향을 주고 때에 따라 제어가 가능하도록 하는 것으로

즉, 실행문의 수행 순서를 변경하는 것

* + 조건문

상황별 실행 흐름을 만드는 문법

* + - 조건식을 만족하는 경우와 그렇지 못한 경우를 나누어 실행을 달리하는 기법
    - 조건문을 사용하면 다양한 결과를 만들 수 있어, 더욱 풍성한 프로그래밍이 가능

ex) 버스 요금은 나이 구간에 따라 다르게 결정

성적 구간에 따른 학점 구분 : 조건문을 사용해 구분하여 구현 가능

* + - 조건문의 종류
      * if 문
        + if-else 문
        + 다중 if 문
        + 중첩 if 문
      * switch 문
      * 삼항 연산자
* **조건문(conditional statement)**
  + if 문
    - if 문은 조건에 따른 수행 여부를 결정
  + if (조건식) {
  + 조건식이 참인 경우 수행할 내용
  + }
  + if-else 문
    - if-else 문은 두 조건식 중 하나를 선택할 때 사용
    - 조건식이 참인 경우 if 문의 코드가, 거짓인 경우 else 문의 코드가 수행
  + if (조건식) {
  + 조건식이 참인 경우 수행할 내용
  + } else {
  + 조건식이 거짓인 경우 수행할 내용
  + }
  + 다중 if 문
    - 추가 조건을 붙여야 할 때 사용
      * 조건문의 수행은 윗줄에서부터 차례대로 진행
      * 다중 if 문을 활용하면 여러 상황 중 하나 선택 가능
  + if (조건식1) {
  + 실행문(들); // 조건식1이 참일 때만 실행
  + } else if (조건식2) {
  + 실행문(들); // 조건식1이 거짓이며 조건식2가 참일 때만 실행
  + ...
  + } else {
  + 실행문(들); // 모든 조건을 만족하지 않을 때만 실행
  + }
  + 중첩 if 문
    - if 문에 다른 if 문이 포함되는 것
    - indentation에 주의
  + switch 문
    - case에 따라 수행 코드를 나누는 문법으로 여러 경로 중 하나를 선택할 때 사용
    - 콜론 레이블을 사용하는 기존 switch 문
      * 기존 switch 문은 낙하 방식으로 콜론 case 레이블 사용
      * switch (변수) // 연산식도 가능하다. {
      * case 상수1 : // case 레이블이다.
      * 0개 이상의 실행문 // break 문과 같은 실행문이 없으면
      * // 다음 case 레이블에 있는 실행문을 수행
      * ...
      * default :
      * 0개 이상의 실행문 // 선택 사항이다.
      * }
      * Java 14부터 화살표 case 레이블, switch 연산식, 다중 case 레이블, yield 예약어 도입올 통한 개선
    - 주의 사항
      * 가능한 모든 값에 대해 일치하는 case 레이블이 없으면 오류 발생
  + 삼항연산자
    - 삼항 연산자는 조건식에 따른 반환값을 지정하는 문법
    - 조건식이 참이면 A, 거짓이라면 B가 반환
  + (조건식) ? A : B;
* **반복문(loop statement)**

규칙적 반복 코드를 단순화하는 문법으로 조건에 따라 같은 처리를 반복하게 하는 문법

반복문을 사용하면 쉽고 빠르게 코드 완성 가능

반복문 사용 시 무한 루프에 빠지지 않도록 주의해야한다.

* + while 문

반복 횟수 불명확 시(조건에 따라 변할 시) 사용하기 적합

while (조건식) {

반복 내용 // 조건식이 거짓이 될 때까지 반복

}

* + - do-while 문

do {

반복 실행문(들); // 본체

} while (조건식);

* + for 문

반복 횟수 명확 시 사용하기 적합

for (초기화; 조건식; 갱신) {

반복 내용

}

* **분기문(branch statement)**
  + break 문

반복문을 탈출하는 문법

* + continue 문

실행 흐름을 다음 반복으로 넘기는 문법

반복문의 모든 코드를 수행하지 않은 채 다음 반복문으로 이동