자바의 정석

Chapter 1 자바를 시작하기 전에

1 자바(Java programming Language)

1.3 자바언어의 특징

1. 운영체제에 독립적
2. 객체지향 언어
3. 비교적 배우기 쉽다
4. 자동 메모리 관리
5. 네트워크와 분산처리 지원
6. 멀티쓰레드 지원
7. 동적 로딩 지원

1.4 JVM

* 자바로 작성된 애플리케이션은 JVM이 반드시 필요
* 자바애플리케이션은 컴파일된 상태가아니고 실행 시에 해석되기 때문에 속도가 느림
  + JIT컴파일러 사용
* Write once run anywhere
  + 한 번 작성하면 어디서든 실행된다

3 자바로 프로그램 작성하기

3.1 Hello.java

* 모든 코드는 반드시 클래스 안에 존재해야 하며 서로 관련된 코드들을 그룹으로 나누어 별도의 클래스를 구성하게 된다
* 클래스들이 모여 하나의 자바애플리케이션을 이룬다
* main 메서드
  + public static void main(String[] args){ }
  + 자바애플리케이션에 시작점
  + 자바애플리케이션에는 main
  + 메서드를 포함한 클래스가 반드시 하나는 있어야 한다

3.2 자주 발생하는 에러와 해결방법

1. cannot find symbol 또는 cannot resolve symbol
   1. 지정된 변수나 메서드를 찾을 수 없다는 뜻으로 선언되지 않은 변수나 메서드를 사용하거나, 변수 또는 메서드의 일므을 잘못 사용한경우에 발생
   2. 자바에서는 대소문자 구분
2. ‘;’ expected
   1. 세미콜론이 필요한 곳에 없다는 뜻
3. Exception in thread “main” java.lang.NoSuchMethodError: main
   1. ‘main메서드를 찾을 수 없다’는 뜻인데 실제로 클래스 내에 main메서드가 존재하지 않거나 메서드의 선언부 public static void main(String[] args)에 오타
4. Exception in thread “main” java.lang.NoClassDefFoundError: Hello
   1. Hello라는 클래스를 찾을 수 없다는 뜻
5. illegal start of expression
   1. 문장에 문법적 오류가 있다 (),{},if,for,public,static같은 키워드
6. class, interface or enum expected
   1. class / interface / enum 이 없거나 괄호 {} 개수 불일치

3.3 자바프로그램 실행과정

1. 프로그램의 실행에 필요한 클래스(.class)로드
2. 클래스파일 검사(형식, 악성코드)
3. 지정된 클래스에서 main메서드 호출

Chapter 2 변수

1 변수 variable

1.1 변수란?

* 변수 : 단 하나의 값을 저장할 수 있는 메모리 공간

1.2 변수의 선언과 초기화

* 변수타입 변수이름 = 값 // int age = 25

1.3 변수의 명명규칙

1. 대소문자가 구분되며 길이제한 X // True 와 true는 다른 변수
2. 예약어 사용 x
3. 숫자로 시작 X
4. 특수문자는 \_ 와 $ 만 허용

@권장규칙

1. 클래스 이름의 첫 글자는 항상 대문자
2. 여러 단어로 이루어진 이름은 단어의 첫글자를 대문자 ex) StringBuffer
3. 상수의 이름은 모두 대문자, \_로 연결 ex) MAX\_NUMBER

2. 변수의 타입

# 기본형과 참조형

* 기본형 (primitive)
  + 실제 값을 저장
  + boolean / 문자형 / 정수형 / 실수형
  + 계산을 위한 실제 값 저장
* 참조형
  + 객체의 주소를 저장
  + 참조형 변수를 선언할 때는 변수의 타입으로 클래스의 이름을 사용
  + 새로운 클래스를 작성한다는 것은 새로운 참조형을 추가하는 셈
  + 참조변수 선언
    - 클래스이름 변수이름 ; //변수타입이 기본형이 아니면 모두 참조변수
* Date today = new Date();
  + 연산자 new 결과는 생성된 객체주소

# 자료형(data type)과 타입(type)의 차이

* 자료형 : 기본형의 종류를 구분할때
* 타입 : 객체의 종류에 의해 구분
* 타입이 자료형을 포함하는 보다 넓은 의미

2.1 기본형

1. 논리형
   1. boolean
2. 문자형
   1. char
3. 정수형
   1. byte
   2. short
   3. int
   4. long
4. 실수형
   1. float
   2. double

2.2 상수와 리터럴

# 상수

* 변하지 않는 값
* final
* 선언과 동시에 반드시 초기화

# 리터럴

* 그자체로 값을 의미하는 것
* int year = 2020 (2020이 리터럴)
* final int MAX\_VALUE = 100 (100이 리터럴)

# 리터럴 타입과 접미사

* 정수형
  + long type : 접미사 L .또는 l
  + int : 접미사 x
  + byte/short타입 리터럴은 별도로 존재 X
  + 정수형에 구분자 \_ 사용 가능 ex) long big = 100\_000\_000L
* 진수
  + 16진수 0X
  + 8진수 0
* 실수형
  + float : f/F
  + double : d/D ,
  + 실수형 기본형은 double이기 때문에 D/d 생략가능
    - float pi = 3.14 : f가 없으므로 double 로 간주 -> 에러
* p를 이용해 16진수 표현가능

# 타입 불일치

* 넓은타입에 좁읍타입 값 저장 가능

# 문자 리터럴과 문자열 리터럴

* 문자 리터럴 : ‘ ’로 문자 하나를 감싼 것 ‘a’
  + 반드시 안에 하나의 문자 있어야함
* 문자열 리터럴 : “ “두 문자 이상 감싼 것
  + “” 안에 아무런 문자 넣지 않는 것을 허용 -> 빈 문자열 이라고함

# String

* 원래 String은 클래스이므로 new를 사용해야 하지만 다음과 같은 표현도 허용
  + String name = new String(“A)
  + String name = “A”
* 덧셈 연산자 이용시 문자열 결합
  + + 는 피연산자가 모두 숫자일 때는 두수를 더하지만 한쪽이 String 이라면 다른 쪽을 String으로 변환 두 String 을 결합

2.3 형식화된 출력 -printf()

* 지시자를 통해 변수의 값을 여러가지 형식으로 변환하여 출력하는 기능을 가지고 있다
* 지시자는 값을 어떻게 출력할 것인지를 지정해주는 역할
* System.out.pritnf(“age: %d” year : %d , age , year);
* 지시자 ‘%x’ , ‘%o’에 #을 사용하면 접두사 0x와 0이 붙음
  + sout(“hex = %x” , hex)
* Integer.toBinaryString(binNum)
  + 10진수를 2진수로 출력
  + sout(“2bin=%s”, Integer.toBinaryString(2bin));

# 2.4 scanner

* 화면에서 입력받기
* import java.util

Scanner scanner = new Scanner(System.in)

String input = scanner.nextLine();

# 문자 -> 숫자 치환

Integer.parseInt()

---------------------------------------------------------------------------------------------------------------------------------

06 객체지향

1.2 특징

1. 코드의 재사용성이 높다
2. 코드의 관리가 용이
3. 신뢰성이 높은 프로그래밍

2 클래스와 객체

2.1 클래스와 객체의 정의와 용도

* 클래스의 정의 : 클래스란 객체를 정의해 놓은 것
* 클래스의 용도 : 클래스는 객체를 생성하는데 사용된다
* 객체의 정의 : 실제로 존재하는것 (사물/개념)
* 객체의 용도 : 클래스는 객체를 생성하는데 사용된다
* 유형의 객체 : 사물
* 무형의 객체 : 논리나 개념
* 프로그래밍에서의 객체 : 클래스에 정의된 내용대로 메모리에 생성된 것
* 클래스는 단지 객체를 생성하는데 사용
* 객체를 사용하기 위해서는 클래스로부터 객체를 생성하는 과정 선행 되어야함

2.2 객체와 인스턴스

* 클래스의 인스턴스화 : 클래스로부터 객체를 만드는 과정
* 인스턴스 : 클래스로부터 만들어진 객체
* 인스턴스는 객체와 같은 의미이지만 객체는 모든 인스턴스를 대표하는 포괄적인 의미를 갖고있으며 인스턴스는 어떤 클래스로부터 만들어진 것인지를 강조하는 보다 구체적인 의미를 갖고 있다
* 클래스 ---(인스턴스화)--> 인스턴스(객체)

.2.3 객체의 구성요소 - 속성과 기능

* 객체는 속성과 기능의 집합
* 객체가 가지고 있는 속성과 기능을 그 객체의 멤버라 한다

# 구성요소

1. 속성(property) : 멤버 변수 / 특성 / 필드 / 상태
2. 기능(function) : 메서드 / 함수 / 행위

2.4 인스턴스의 생성과 사용

ex)

클래스명 변수명 ;

변수명 = new 클래스명();

Tv t;

t = new Tv();

* 인스턴스는 참조변수를 통해섬나 다룰 수 있으며 참조변수의 타입은 인스턴스의 타입과 일치해야 한다.

2.5 객체 배열

* 객체 배열 생성후 반드시 초기화 수행

2.6 클래스의 또 다른 정의

1. 클래스 - 데이터와 함수의 결합

* 프로그래밍 언어에서 데이터 처리를 위한 데이터 저장형태 발전과정
  + 변수 -> 배열 -> 구조체 -> 클래스

1. 변수 : 하나의 데이터를 저장할 수 있는 공간
2. 배열 : 같은 종류의 여러 데이터를 하나의 집합으로 저장할 수 있는 공간
3. 구조체 : 서로 관련된 여러 데이터를 종류에 관계없이 하나의 집합으로 저장할 수 있는 공간
4. 클래스 : 데이터와 함수의 결합(구조체 + 함수)

* 객체지향 언어에서는 변수와 함수를 하나의 클래스에 정의(클래스)하여 서로 관계가 깊은 변수와 함수들을 함께 다룰 수 있게 함

1. 클래스 - 사용자정의 타입

* 사용자 정의 타입 : 자료형(primitive type)외에 프로그래머가 서로 관련된 변수들을 묶어서 하나의 타입으로 새로 추가하는 것
* 객체지향 언어에서 사용자 정의 타입 = 클래스

3. 변수와 메서드

3.1 선언위치에 따른 변수에 종류

# 변수종류

* 변수의 선언된 위치로 구분
* 멤버 변수를 제외한 나머지 변수들은 모두 지역변수
* 멤버변수중 static이 붙은 것은 클래스 변수, 붙지않은 것은 인스턴스 변수

변수종류 선언위치 생성시기

|  |  |  |
| --- | --- | --- |
| 클래스변수 | 클래스 영역 | 클래스가 메모리에 올라 갈 때 |
| 인스턴스 변수 | 인스턴스가 생성되었을 때 |
| 지역변수 | 클래스 영역 이외의 영역 | 변수 선언문이 수행되었을 때 |

1. 인스턴스 변수

* 클래스 영역에 선언되며 클래스의 인스턴스를 생성할 때 만들어진다
* 인스턴스 변수값을 읽거나 저장하기 위해서 먼저 인스턴스를 생성해야함
* 인스턴스는 독립적인 저장공간을 가지므로 서로 다른 값을 가질 수 있다
* 인스턴스마다 고유한 상태를 유지해야 하는 속성의 경우 인스턴스 변수로 선언

1. 클래스변수

* 인스턴스 변수 앞에 static을 붙이면 된다
* 클래스변수는 모든 인스턴스가 공통된 저장공간(변수)을 공유하게 됨
* 인스턴스를 생성하지 않고도 바로 사용 가능 (클래스이름.클래스변수)
* 클래스가 메모리에 로딩 될 때 생성되어 프로그램이 종료될 때 까지 유지
* public을 붙이면 전역변수의 성격을 갖는다

1. 지역변수

* 메서드 내에 선언되어 메서드 내에서만 사용 가능하며 메서드 종료시 소멸
* 블럭 내에 지역변수는 블록 내에서만 사용가능

3.2 클래스변수와 인스턴스변수

* 인스턴스변수는 인스턴스가 생성될 때 마다 생성되므로 인스턴스마다 각기 다른 값ㅇ르 유지할 수 있지만 클래스 변수는 모든 인스턴스가 하나의 저장공간을 공유하므로 항상 공통된 값을 갖는다.

3.3 메서드

* 메서드 : 특정 작업을 수행하는 일련의 문장들을 하나로 묶은것
* 메서드를 사용하는 이유

1. 높은 재사용성
2. 중복된 코드의 제거
3. 프로그램의 구조화

3.4 메서드의 선언과 구현

* 메서드 구성 : 선언부 / 구현부

ex) 반환타입 메서드이름(타입 변수명){ //선언부

} // 구현부

# 메서드 선언부

* 메서드이름 / 매게변수 선언 / 반환타입 으로 구성

# 메서드 구현부

* return
* 지역변수

3.7 JVM 메모리 구조

1. 메서드 영역

* 프로그램 실행 중 어떤 클래스가 사용되면 JVM은 해당 클래스의 클래스파일을(.class)읽어서 분석하여 클래스에 대한 정보를 이곳에 저장 그 클래스의 클래스변수도 이영역에서 함께 생성

1. 힙

* 인스턴스가 생성되는 공간. 프로그램 실행 중 생성되는 인스턴스는 모두 이곳에 생성된다 즉 인스턴스변수 들이 생성되는 공간

1. 호출스택

* = call stack / execution stack
* 호출스택은 메서드의 작업에 필요한 메모리 공간을 제공한다 이 메모리는 메서드가 작업을 수행한느 동안 지역변수들과 연산의 중간결과 등을 저장하는데 사용도니다 그리고 메서드가 작업을 마치면 할당되었던 메모리공간은 반환되어 비워준다
* 특징

1. 메서드가 호출되면 수행에 필요한 만큼의 메모리를 스택에 할당받는다.
2. 메서드가 수행을 마치고 나면 사용했던 메모리를 반환하고 스택에서 제거
3. 호출스택의 제일 위에 있는 메서드가 현재 실행중인 메서드
4. 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드

3.8 기본형 매개변수와 참조형 매개변수

* 기본형 매개변수 : 변수의 값을 읽기만 할 수 있다
* 참조형 매개변수 : 변수의 값을 읽고 변경할 수 있다

3.11 클래스메서드(static메서드)와 인스턴스 메서드

* 인스턴스 메서드는 인스턴스 변수와 관련된 작업을 하는, 즉 메서드의 작업을 수행하는데 인스턴스 변수를 필요로 하는 메서드이다
* 인스턴스와 관계없는(인스턴스 변수나 인스턴스메서드를 사용하지 않는) 메서드를 클래스 메서드로 정의한다

1. 클래스를 설계 할 때 멤버변수 모든 인스턴스에 공통으로 사용하는 것에 static을 붙인다
2. 클래스변수는 인스턴스를 생성하지 않아도 사용할 수 있다
3. 클래스메서드는 인스턴스 변수를 사용할 수 없다
4. 메서드 내에서 인스턴스 변수를 사용하지 않는다면 static을 붙이는 것을 고려한다

* 클래스 멤버변수 중 모든 인스턴스에 공통된 값을 유지해야 하는 것이 있는지 살펴보고 있으면 static을 붙여준다
* 작성한 메서드 중에서 인스턴스 변수나 인스턴스 메서드를 사용하지 않는 메서드에 static을 붙일 것을 고려한다

4. 오버로딩

* 메서드 오버로딩 : 한 클래스 내에 같은 이름의 메서드를 여러개 정의
* 조건
  + 메서드 이름이 같아야 한다
  + 매개변수의 개수 또는 타입이 달라야 함
  + ex) println()