혼자 공부하며 함께 만드는

혼공 용어 노트

목차

가나다 순

객체 지향 OOP: Object Oriented Programming 01
객체 object 10
기본 타입 primitive type 05
다형성 polymorphism 15
디버깅 debugging 03
리터럴 literal 04
리턴 타입 return type 12
리턴 return 12
매개 변수 parameter 12
메소드 선언부 method signature 12
메소드 method 02
무한 루프 infinite loop 09
반복문 loop 08
배열 array 10
버퍼 buffer 20
변수 variable 03
부동 소수점 floating point 06
상속 inheritance 15
상수 constant value 13
생성자 constructor 11
소스 source 01
스레드 thread 19
스택 stack 19
스트림 stream 20
식별자 identifier 12
실행문 02
싱글톤 Singleton 11
쓰레기 수집기 garbage collector 09
연산자 operator 06
열거 타입 enumeration type 10
예약어 reserved word 04
예외 처리 exception handling 17
예외 exception 17
오버라이딩 overriding 15
오버로딩 overloading 14
유니코드 unicode 05
이스케이프 문자 escape character 04
익명 객체 anonymous object 17
인덱스 index 10
인스턴스 instance 11
인터페이스 interface 16
자료형 data type 05
자바 가상 기계 JVM: Java Virtual Machine 01
자바 개발 도구 JDK: Java Development Kit 02
자바 실행 환경 JRE: Java Runtime Environmen 02
접근 제한자 access modifier 13
제어문 control statement 06
조건문 conditional statement 07
주석 comment 02
중첩 클래스 nested class 17
참조 타입 reference type 05
초기값 initial value 04
초기화 initialize 04
추상 메소드 abstract method 16
추상 클래스 abstract class 15
컴파일 compile 01
큐 queue 20
클래스 class 02
통합 개발 환경 IDE: Integrated Development Environment 03
프로세스 process 18
피연산자 operand 06
필드 field 12
형 변환 14

ABC 순

API Application Programming Interface 18
abstract class 추상 클래스 15
abstract method 추상 메소드 16
access modifier 접근 제한자 13
anonymous object 익명 객체 17
array 배열 10
buffer 버퍼 20
class 클래스 02
comment 주석 02
compile 컴파일 01
conditional statement 조건문 07
constant value 상수 13
constructor 생성자 11
control statement 제어문 06
data type 자료형 05
debugging 디버깅 03
enumeration type 열거 타입 10
escape character 이스케이프 문자 04
exception 예외 17
exception handling 예외 처리 17
field 필드 12
floating point 부동 소수점 06
garbage collector 쓰레기 수집기 09

목차

identifier 식별자 12
IDE: Integrated Development Environment 통합 개발 환경 03
index 인덱스 10
infinite loop 무한 루프 09
inheritance 상속 15
initial value 초기값 04
initialize 초기화 04
instance 인스턴스 11
interface 인터페이스 16
JDK: Java Development Kit 자바 개발 도구 02
JRE: Java Runtime Environmen 자바 실행 환경 02
JVM: Java Virtual Machine 자바 가상 기계 01
literal 리터럴 04
loop 반복문 08
method 메소드 02
method signature 메소드 선언부 12
nested class 중첩 클래스 17
null 09
object 객체 10
OOP: Object Oriented Programming 객체 지향 01
operand 피연산자 06
operator 연산자 06
overloading 오버로딩 14
overriding 오버라이딩 15
parameter 매개 변수 12
polymorphism 다형성 15
primitive type 기본 타입 05
process 프로세스 18
queue 큐 20
reference type 참조 타입 05
reserved word 예약어 04
return 리턴 12
return type 리턴 타입 12
Singleton 싱글톤 11
source 소스 01
stack 스택 19
stream 스트림 20
thread 스레드 19
unicode 유니코드 05
variable 변수 03

01장

□ **소스**

source [1장 2쪽]

고급 언어로 작성된 내용

□ **컴파일**

compile [1장 2쪽]

컴퓨터가 이해할 수 있도록 0과 1로 이루어진 기계어로 변환하는 과정

□ **객체 지향 프로그래밍**

OOP: Object Oriented Programming [1장 6쪽]

프로그래밍을 개발하는 기법으로, 부품에 해당하는 객체들을 먼저 만들고 이것들을 하나씩 조립 및 연결해서 전체 프로그램을 완성하는 기법

그것이 알고싶다 **객체 지향 프로그래밍을 할 수 있는 언어는 따로 있다?**

흔히 자바는 객체지향 언어, C언어는 절차지향 언어라고 칭하는데, 그렇다고 해서 자바로 절차 지향 프로그래밍을 할 수 없는 것은 아니다. 단지 자바는 객체 지향 프로그래밍에 적합한 구조로 만들어진 언어일 뿐. 객체 지향 프로그래밍은 프로그래밍의 '패러다임'이다!

□ **자바 가상 기계**

JVM: Java Virtual Machine [1장 24쪽]

자바 프로그램은 완전한 기계어가 아닌 중간 단계의 바이트 코드이기 때문에, 이것을 해석하고 실행할 수 있게 해주는 가상의 운영체제

* JVM의 메모리 영역

- 메소드 영역: 클래스(~.class)들을 클래스 로더로 읽어 클래스별로 런타임 상수 풀, 필드 데이터, 메소드 데이터, 메소드 코드, 생성자 코드 등을 분류해서 저장한다.
- 힙 영역: 객체와 배열이 생성되는 영역. JVM 스택 영역에서 생성된 변수나 다른 객체의 필드에서 이 곳에 있는 객체와 배열을 참조한다.
- JVM 스택 영역: 스레드마다 하나씩 존재. 스레드가 시작할 때마다 JVM 스택 영역이 할당된다.

□ **자바 개발 도구**

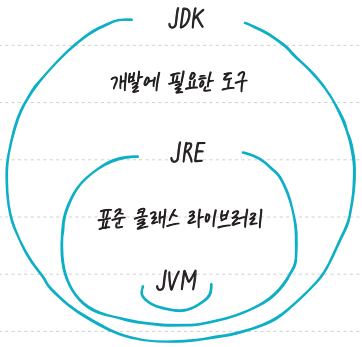
JDK: Java Development Kit 참고 용어 필드, 스레드 [1장 3쪽]

프로그램에 필요한 자바 가상 기계(JVM), 라이브러리 API, 컴파일러 등의 개발 도구가 포함된 소프트웨어 패키지

□ **자바 실행 환경**

JRE: Java Runtime Environment 참고 용어 API [1장 26쪽]

자바 프로그램 실행에 필요한 자바 가상 기계(JVM), 라이브러리 API가 포함된 소프트웨어 패키지



□ **실행문**

변수 선언, 값 저장, 메소드 호출에 해당하는 코드 [1장 39쪽]

□ **클래스**

class [1장 37쪽]

객체를 생성하기 위한 필드와 메소드가 정의된 것. 객체의 설계도에 해당한다.

□ **메소드**

method [1장 37쪽]

어떤 일을 처리하는 실행문들을 모아 놓은 블록

- 메소드 호출: 메소드를 코드 내에서 사용하는 것.

□ **주석**

comment [1장 38쪽]

프로그램 실행과는 상관없이 코드에 설명을 붙인 것.

- // : //부터 라인 끝까지 주석으로 처리한다.
- /* ~ */ : /*와 */ 사이에 있는 모든 범위를 주석으로 처리한다.


```
//이 책을 다 보고 나서 무엇을 공부하면 될까요?

/*자바 프로그래머가 되고 싶다면 <자바 객체지향 디자인 패턴>을 추천드리고요,
또 다른 언어를 공부하고 싶다면 혼공 파이썬 / 혼공 C언어를 추천드립니다.
전국 서점에서 절찬 판매중...
*/
```

□ **통합 개발 환경**

IDE: Integrated Development Environment [1장 15쪽]

프로젝트 생성, 자동 코드 완성, 디버깅 기능을 제공하는 환경

ex) Eclipse(자바), Visual Studio(C/C++)

□ **디버깅**

debugging [1장 15쪽]

모의 실행을 해서 코드의 오류를 찾는 행위

02장

□ **변수**

variable [2장 48쪽]

하나의 값을 저장할 수 있는 메모리의 공간. 프로그램에 의해서 수시로 값이 변동될 수 있다. 하나의 변수에는 오로지 하나의 값만 저장할 수 있고, 자바에서는 선언된 변수에 한 가지 종류의 값만 저장할 수 있다.

- 로컬 변수(지역 변수): 메소드 블록 내에서 선언된 변수. 로컬 변수는 메소드 실행이 끝나면 메모리에서 자동으로 없어진다.

* 변수와 관련된 용어

- 변수 선언: 어떤 식별자를 갖고, 어떤 데이터를 가지고 시작하는 변수라는 것을 알리는 것.
- 할당: 변수에 값을 넣거나 저장하는 것을 '할당한다'고 한다.
- 참조: 변수에 접근하는 것을 '참조한다'고 한다.

→ 메모리에 변수의 값이 들어갈 공간을 할당해야 하기 때문.

□ 예약어

reserved word [2장 49쪽]

이미 해당 프로그래밍 언어에서 사용하기 위해 의미가 정해져 있는 것.

그것이 알고싶다 예약어를 식별자(변수명/메소드명/함수명)로 사용하면 안되는 걸까?

프로그래밍 언어 내에서 이미 문법적인 용도로 사용되고 있기 때문에 사용하면 안 된다. 자바에서 예약어를 식별자로 사용할 경우 컴파일러에서 에러 처리하여 실행되지 않는다.

□ 초기값

initial value [2장 50쪽]

변수를 선언하고 처음 저장하는 값

□ 초기화

initialize [2장 50쪽]

변수에 초기값을 주는 행위

□ 리터럴

literal 참고 용어 상수 [2장 60쪽]

소스 코드 내에서 직접 입력된 값

그것이 알고싶다 상수와 리터럴

사실 리터럴은 상수와 같은 의미지만, 프로그램에서는 상수를 "값을 한 번 저장하면 변경할 수 없는 변수"로 정의하기 때문에 이와 구분하기 위해 "리터럴"이라는 용어를 사용한다.

□ 이스케이프 문자

escape character [2장 46쪽]

역슬래시(\) 기호가 붙은 특수한 문자 리터럴

이스케이프 문자	출력
\t	수평 탭
\n ——→ 이 때 n은 new line의 약자!	줄 바꿈
\r	캐리지 리턴
\"	" (큰 따옴표)
\'	' (작은 따옴표)
\\	\
\u16진수	16진수에 해당하는 유니코드

□ 유니코드

unicode [2장 63쪽]

세계 각국의 문자들을 코드값으로 매핑한 국제 표준 규약이다. 유니코드는 하나의 문자에 대해 하나의 코드값을 부여하기 때문에 영문 'A' 및 한글 '가'도 하나의 코드값을 가진다. 자바는 모든 문자를 유니코드로 처리한다.

```
char c1 = 'A';
char c2 = '\u0041';
```

그것이 알고싶다 아스키(ASCII) 코드와 유니코드

유니코드는 영문자 외에 전 세계의 문자를 표현할 수 있도록 설계된 반면 아스키(ASCII: American Standard Code for Information Interchange, 미국 정보 교환 표준 부호) 코드는 7비트로 표현되는 영문자 기반 인코딩이다. 유니코드의 앞부분은 아스키 문자로 할당되어 있다.

□ 자료형

data type [2장 49쪽]

자료의 형태. 자료형에 따라 컴퓨터가 어떻게 처리하는지 달라진다. 자바에서는 기본 타입과 참조 타입으로 구분됨

□ 기본 타입

primitive type [2장 59쪽]

원시 타입이라고도 한다. 정수, 실수, 문자, 논리 리터럴을 저장하는 자료형

□ 참조 타입

reference type [2장 121쪽]

객체의 번지를 참조하는 타입. 배열, 열거, 클래스, 인터페이스 타입이 있다.

그것이 알고싶다 값에 의한 호출call by value과 참조에 의한 호출call by reference

값에 의한 호출을 할 땐 메소드가 전달인자를 복사하여 사용하고, 메소드 실행이 종료되면 반환하는 방식이기 때문에 전달인자를 직접 수정하는 것이 의미가 없다. 참조란 대상의 주소값을 통해 접근하는 것이고, 참조에 의한 호출은 메소드가 전달인자로 주소값을 넘겨받아 해당 위치에 있는 값에 접근하여 다른 곳에서 쓰일 수 있도록 수정할 수 있다. 자바에서는 무조건 call by value인 것이 아닌가 싶지만, 전달인자로 받은 데이터의 타입에 따라 그 값value이 주소값이 되기도 하고, 객체가 가리키는 값이 되기도 한다.

03장

□ 연산자 operator [3장 102쪽]

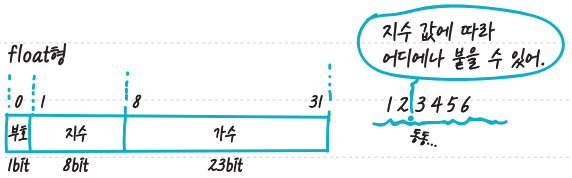
연산에 사용되는 표시나 기호

□ 피연산자 operand [3장 102쪽]

연산식에서 연산되는 데이터

□ 부동 소수점 floating point [3장 120쪽]

소수점이 있는 실수 데이터를 저장하는 방식. '부동'은 '떠다니다'의 의미, 소수점이 떠다닌다는 의미에서 부동 소수점이다. 이 방식에서는 최상위 비트MSB: Most Significant Bit를 부호로 결정한다. 최상위 비트가 0이면 양수, 1이면 음수를 뜻한다.



04장

□ 제어문 control statement [4장 134쪽]

실행 흐름을 개발자가 원하는 방향으로 바꿀 수 있도록 해주는 것.

- 조건문: if문, switch 문
- 반복문: for문, while문, do-while문

□ 조건문 conditional statement [4장 134쪽]

특정 조건을 만족할 때 코드를 실행하는 문법

- if문: 괄호 내의 조건식이 참이면 블록 내의 문장을 실행한다.
- else문: if문의 조건식이 거짓이면 블록 내의 문장을 실행한다. 필요 없으면 없어도 된다.
- else if문: if문의 조건식이 거짓일 때 실행시킬 코드에 추가 조건을 걸고 싶을 때 사용한다. 마찬가지로 필요 없으면 else if를 사용하지 않아도 된다.

```
if (a > 0)
  //중괄호를 사용하지 않으면 조건식이 참일 때 한 줄만 실행
  System.out.println("a는 양수입니다.");
else if (a == 0) {
  System.out.println("a는 0입니다.");
} else {
  System.out.println("a는 음수입니다.");
}

switch(ch) {
case 'a':
  //이 케이스에서 break 문이 없기 때문에 블록을 빠져나가지 않고 다음
  case 문으로 넘어간다
case 'b':
  System.out.println("ch는 a 혹은 b입니다"); //a일 경우에도 출력된다.
  break;
case 'c':
  System.out.println("ch는 c입니다");
  break;
default:
  System.out.println("a도 b도 c도 아닙니다");
  break; //이 break문을 적지 않아도 이 블록을 빠져나갈 수 있다.
}
```

□ 반복문

loop [4장 148쪽]

특정 조건을 만족하는 동안 반복해서 실행하는 문법

• for문: 반복 횟수가 정해진 경우에 주로 사용

```
//for문 예시
for(int i = 0; i < n; i++)
  //0부터 n-1까지 i를 증가시키며 코드 실행 (n회 반복)
```

• while문: 반복 횟수를 모를 때 주로 사용

```
//while문 예시
while(true) {
  result *= 3;
  if(result >= 100) //이 조건을 만족시켜야 반복문 종료
        break;
}
```

• do-while문: 조건 만족 여부와 상관없이 코드를 먼저 실행하고, 그 다음 루프부터 조건을 검사

```
//do-while문 예시
do {
  printf("어머 이건 꼭 출력 되어야 해...! ");
} while(false); //1회만 출력
```

□ 무한 루프

infinite loop [4장 155쪽]

무한히 반복을 하고 싶거나 정해진 횟수 없이 일정한 조건을 충족하면 빠져나오게 하고 싶을 때 사용하는 것.

```
//for문을 사용한 무한 루프
for(;;) {
  //반복하고 싶은 코드
}

//while문을 사용한 무한 루프
while(true){
  //반복하고 싶은 코드
}
```

05장

□ 쓰레기 수집기

garbage collector 참고 용어 오버라이딩, 인스턴스 [5장 167쪽]

쓰레기 수집은 메모리 관리 기법 중 하나이다. 자바에선 JVM의 쓰레기 수집기를 이용해 자동적으로 사용하지 않는 객체를 메모리에서 제거한다.

그것이 알고싶다 **객체를 제거하기 전 수행하고 싶은 일이 있을 때**

쓰레기 수집기가 객체를 제거하기 전에 중요한 데이터를 저장하고 싶을 때, 클래스 내에 객체 소멸자(finalize())를 재정의하여 객체가 소멸될 때 실행할 코드를 입력할 수 있다.

□ null

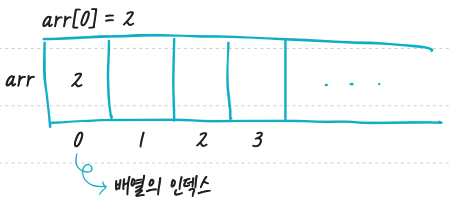
참조 타입 변수가 객체를 참조하지 않는다는 의미의 값. '널'이라고 읽는다. null값도 초기값으로 사용할 수 있기 때문에 null로 초기화된 참조 변수는 스택 영역에 저장된다. [5장 169쪽]

□ **배열**

array [5장 170쪽]

데이터를 연속된 공간에 나열하고, 각 데이터에 인덱스를 부여해 놓은 자료구조

→ 이러한 성질 때문에 중간 인덱스의 값을 삽입하거나 삭제하기 어렵고, 선언 후 배열의 인덱스 수를 늘릴 수 없다.



□ **인덱스**

index [5장 178쪽]

0에서부터 "문자열(혹은 배열) 길이−1"까지의 번호를 매긴 것. 배열과 같은 선형 자료구조는 인덱스로 원소에 접근할 수 있다.

□ **열거 타입**

enumeration type [5장 203쪽]

한정된 값만을 갖는 자료형

```
enum Week {
  SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

↓ 열거 상수

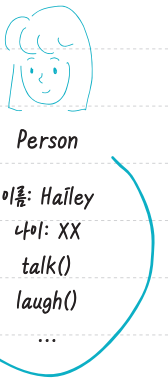
06장

□ **객체**

object [6장 212쪽]

물리적으로 존재하거나 추상적으로 생각할 수 있는 것 중에서 자신의 속성을 가지고 있고 다른 것과 식별 가능한 것을 말함.

ex) 자동차, 자전거, 책, 사람 / 학과, 강의, 주문 등...



□ **인스턴스**

instance [6장 215쪽]

클래스로부터 만들어진 객체

그것이 알고싶다 인스턴스 멤버와 정적 멤버

인스턴스 멤버란 객체(인스턴스)를 생성한 후 사용할 수 있는 필드와 메소드를 말한다. 이들을 각각 인스턴스 필드, 인스턴스 메소드라고 부른다. 정적 멤버는 클래스에 고정된 멤버로서 객체를 생성하지 않고 사용할 수 있는 필드와 메소드를 말한다. 이들을 각각 정적 필드, 정적 메소드라고 부른다. 정적 멤버는 객체가 아닌 클래스에 소속된 멤버이기 때문에 클래스 멤버라고도 한다.

□ **생성자**

constructor [6장 221쪽]

new 연산자로 호출되는 특별한 중괄호 블록. 객체 생성 시 객체를 초과하는 역할을 한다. 메소드와 비슷하게 생겼지만, 클래스 이름으로 되어 있고 반환형이 없다. 객체 초기화란 필드를 초기화하거나, 메소드를 호출해서 객체를 사용할 준비를 하는 것을 말한다.

□ **싱글톤**

Singleton [6장 279쪽]

전체 프로그램에서 하나의 클래스에 단 하나의 객체만 만들도록 보장해야 하는 경우에 사용하는데, 이 때 생성된 객체를 싱글톤이라고 부른다. 생성자를 호출한 만큼 객체가 생성되기 때문에 클래스 외부에서 new 연산자로 생성자를 호출할 수 없도록 막아야 한다. 생성자 앞에 private 접근 제한자를 붙이고, 필드에 자기 자신의 객체를 static으로 선언하면 된다.

□ **식별자** — **identifier** [6장 216쪽]

프로그래밍 언어에서 프로그래머가 변수, 메소드, 클래스 등에 부여하는 이름

□ **리턴 타입** — **return type** [6장 221쪽]

메소드가 실행 후 리턴하는 값의 타입. 반환형이라고도 한다. void로 선언된 리턴 값(return value)이 없는 메소드에서도 return문을 사용할 수 있지만 리턴값 없이 사용해야 한다.

□ **리턴** — **return** [6장 221쪽]

함수를 실행했던 위치로 돌아가게 하는 것. 리턴값을 가지는 함수는 반드시 리턴할 때 반환하는 값이 있어야 한다.

□ **메소드 선언부** — **method signature** [6장 247쪽]

리턴형, 메소드 이름, 매개 변수 선언을 포함하는 것. 영문명 그대로 메소드 시그니처라고도 한다.

□ **매개 변수** — **parameter** [6장 249쪽]

메소드가 실행할 때 필요한 데이터를 외부로부터 받기 위해 사용. 매개 변수가 필요한 경우가 있고 필요 없는 경우가 있다.

그것이 알고싶다 전달인자ARGUMENT와 매개변수PARAMETER

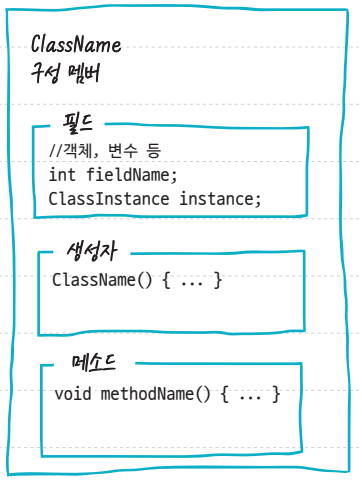
```
int mySum(int[] arr) {
  return Arrays.stream(arr).sum();
}
...
int myVar = mySum(myArray);
...
```

(arr → 매개변수, myArray → 전달인자)

□ **필드** — **field** (→ 즉, 멤버 변수) [6장 212쪽]

객체의 데이터가 저장되는 곳. 생성자와 메소드 전체에서 사용되며 객체가 소멸되지 않는 한 객체와 함께 존재한다.

* 클래스의 구조



□ **상수** — **constant value** [6장 281쪽]

값을 한 번 저장하면 변경할 수 없는 변수. 상수의 이름은 모두 대문자로 작성하는 것이 관례이다. 서로 다른 단어가 혼합된 이름이라면 언더바(_)로 단어들을 연결해준다.

```
static final double EARTH_SURFACE_AREA;
static final double EARTH_RADIUS = 6400;
static {
  EARTH_SURFACE_AREA = 4 * MATH.PI * EARTH_RADIUS * EARTH_RADIUS;
}
...
EARTH_SURFACE_AREA = 10; //에러!
```

□ **접근 제한자** — **access modifier** 참고 용어 캡슐화 [6장 286쪽]

객체의 필드와 메소드의 사용 범위를 제한하여 외부로부터 보호한다. 캡슐화된 멤버를 노출시킬 것인지, 숨길 것인지를 결정하기 위해 접근 제한자를 사용한다.

그것이 알고싶다 GETTER와 SETTER

외부에서 마음대로 읽고 변경할 수 없도록 제어하는 메소드. 필드에 직접 접근하지 않고 데이터를 조회, 수정할 수 있도록 하는 역할을 한다. 만약 외부에서 필드값을 읽을 수만 있고 변경하지 못하도록 하려면(읽기 전용) Getter 메소드만 선언하거나, 아니면 Setter 메소드를 필요에 따라 private 혹은 protected 접근 제한을 갖도록 선언하면 된다.

□ 오버로딩

overloading [6장 238쪽]

오버로딩의 사전적 의미는 많이 싣는 것을 뜻한다. 하나의 메소드 이름으로 여러 기능을 담는다 하여 붙여진 이름이라 생각할 수 있다.

- 메소드 오버로딩: 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것. 매개 변수의 타입, 개수, 순서 중 하나가 달라야 한다.
- 생성자 오버로딩: 매개 변수를 달리하는 생성자를 여러 개 선언하는 것.

> **그것이 알고싶다** 연산자 오버로딩
>
> 자바에서는 지원하지 않지만 C++과 같은 언어에서는 연산자도 오버로딩하여 사용할 수 있다.

□ 타입 변환

자료 타입을 다른 자료 타입으로 변환하는 것 [6장 72, 333쪽]

- 자동 타입 변환: 프로그램 실행 도중에 자동적으로 타입 변환이 일어나는 것을 말한다.

```
Animal animal1 = new Cat();
       ↓
   Cat으로 자동 타입 변환
```

- 강제 타입 변환casting: 프로그래머가 강제적(명시적)으로 타입을 변환하는 것을 말한다.

```
float f1 = 1.23F;
int n1 = (int)f1; //n1에 1 저장
int n2 = f1; //자료형 불일치로 에러
```


07장

□ 상속

inheritance [7장 310쪽]

상위(부모) 객체를 기반으로 하위(자식) 객체를 생성하는 관계를 말한다. 상위 객체는 자기가 가지고 있는 필드와 메소드를 하위 객체에게 물려주어 하위 객체가 사용할 수 있도록 한다. 일반적으로 상위 객체는 종류를 의미하고, 하위 객체는 구체적인 사물에 해당한다. 자식 클래스를 선언할 때 어떤 부모 클래스를 상속받을 것인지 결정한다.

- 부모(parent class, super class / 상위 클래스): 해당 클래스에 선언된 변수와 함수를 물려주는 클래스
- 자식(child class, sub class / 하위 클래스): 다른 클래스로부터 물려받고자 하는 클래스

□ 오버라이딩

overriding [7장 317쪽]

부모에게서 상속받은 메소드의 내용이 자식 클래스에 맞지 않을 경우, 자식 클래스에서 동일한 메소드를 재정의하는 것을 말한다.

□ 다형성

polymorphism [7장 332쪽]

같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질을 말한다. 코드 측면에서 보면 다형성은 하나의 타입에 여러 객체를 대입함으로써 다양한 기능을 이용할 수 있도록 해준다.

□ 추상 클래스

abstract class [7장 357쪽]

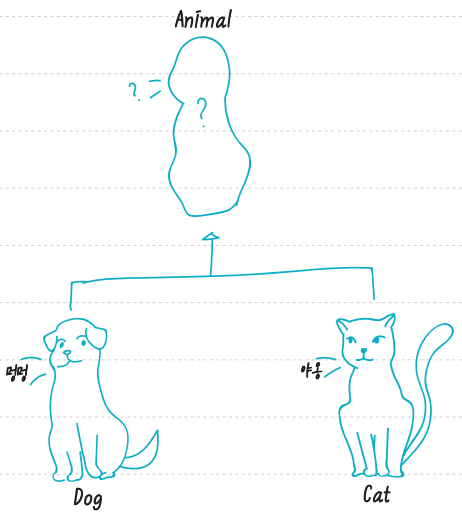
실체 클래스가 공통적으로 가져야 할 필드와 메소드들을 정의해 놓은 추상적인 클래스. 추상 클래스는 실체 클래스의 공통되는 필드와 메소드를 추출해서 만들었기 때문에 객체를 직접 생성해서 사용할 수 없다.

□ **추상 메소드**

abstract method 참고 용어 오버라이딩 [7장 362쪽]

하위 클래스가 반드시 오버라이딩하도록 하고 싶은 경우 사용하는 메소드. 메소드의 선언부만 있고 메소드 실행 내용인 중괄호가 없다. 추상 메소드는 추상 클래스 안에서만 선언 가능함.

```
Animal animal1 = new Cat();
Animal animal2 = new Dog();
animal1.sound(); //야옹
animal2.sound(); //멍멍
```



08장

□ **인터페이스**

interface [8장 370쪽]

객체의 사용 방법을 정의한 타입이고 개발 코드와 객체가 서로 통신하는 접점 역할을 한다. 개발 코드가 인터페이스의 메소드를 호출하면 인터페이스는 객체의 메소드를 호출시킨다. 그렇기 때문에 개발 코드는 객체의 내부 구조를 알 필요가 없고 인터페이스의 메소드만 알고 있으면 된다.

09장

□ **중첩 클래스**

nested class [9장 406쪽]

클래스 내부에 선언한 클래스. 두 클래스의 멤버들에 서로 쉽게 접근 할 수 있고, 외부에는 불필요한 관계 클래스를 감춤으로써 코드의 복잡성을 줄일 수 있다.

- 멤버 클래스: 클래스의 멤버로써 선언되는 중첩 클래스
- 로컬 클래스: 메소드 내부에서 선언되는 중첩 클래스

□ **익명 객체**

anonymous object [9장 424쪽]

이름이 없는 객체를 말한다. 익명 객체를 만들려면 어떤 클래스를 상속하거나 인터페이스를 구현해야 한다.

10장

□ **예외**

exception [10장 444쪽]

사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인해 발생하는 프로그램 오류

□ **예외 처리**

exception handling [10장 444쪽]

일반적으로 프로그램이 처리되는 동안 특정한 문제가 일어났을 때 처리를 중단하고 다른 처리를 하는 것. 자바에서는 예외exception라고 부르는 오류를 처리하는 과정을 말하기도 한다.

- 일반 예외: 컴파일러 체크 예외라고도 한다. 만약 예외 처리 코드가 없다면 컴파일 오류가 발생한다. 일반 예외는 Exception을 상속받지만 Runtime Exception은 상속받지 않는다.

• 실행 예외: 컴파일하는 과정에서 예외 처리 코드를 검사하지 않는 예외를 말한다. 실행 예외는 RuntimeException을 상속받은 클래스들이다.

11장

□ **API**

Application Programming Interface [11장 467쪽]

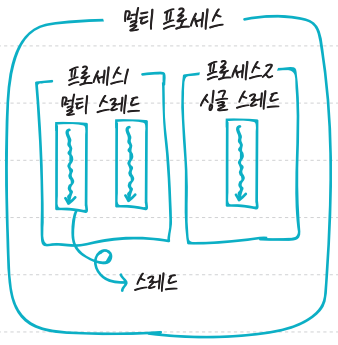
프로그램 개발에 자주 사용되는 클래스 및 인터페이스의 모음. 라이브러리library라고도 부른다.

12장

□ **프로세스**

process [12장 520쪽]

운영체제로부터 메모리를 할당받아 실행하는 애플리케이션의 코드



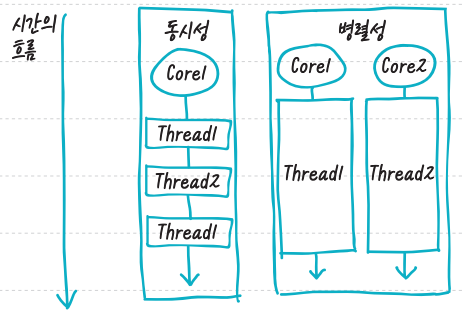
□ **스레드**

thread [12장 520쪽]

한 가지 작업을 실행하기 위해 순차적으로 실행할 코드를 실처럼 이어 놓았다고 해서 유래된 이름. 하나의 스레드는 하나의 코드 실행 흐름이기 때문에 한 프로세스 내에 스레드가 두 개라면 두 개의 코드 실행 흐름이 생긴다는 의미.

그것이 알고싶다 **동시성**concurrency, **병렬성**parallelism

멀티 스레드는 동시성 또는 병렬성으로 실행된다. 동시성은 멀티 작업을 위해 하나의 코어에서 멀티 스레드가 번갈아가며 실행하는 성질을 말하고, 병렬성은 멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질을 말한다.



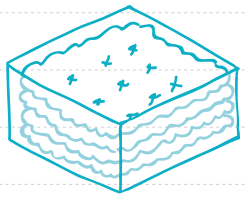
13장

□ **스택**

stack [13장 585쪽]

후입선출LIFO: Last In First Out 자료구조. 나중에 넣은 객체가 먼저 빠져나간다.

위에 있는 과자를 꺼내야
아래에 있는 과자를 꺼낼 수 있는
스택의 형태를 하고 있다.



□ **큐**

queue [13장 585쪽]

선입선출FIFO: First In First Out 자료구조. 먼저 넣은 객체가 먼저 빠져나간다.

14장

□ **스트림**

stream [14장 592쪽]

프로그램이 외부 파일, 외부 네트워크 등과 통신할 때 데이터가 흐르는 길. 이 때 외부란 같은 컴퓨터 내부라도 프로그램의 바깥에 별도로 존재하는 것을 말한다.

□ **버퍼**

buffer [14장 595쪽]

데이터를 한 곳에서 다른 한 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 메모리의 영역. 버퍼는 큐의 한 형태로, 요청된 자료는 버퍼에 도착한 순서대로 처리된다.