**Java 내용정리**

□ Java 특징

∙ 운영체제에 독립적

- 자바가상머신(JVM)을 통해서 가능

- 자바로 작성된 프로그램은 운영체제와 하드웨어에 관계없이 실행 가능하며 한 번 작성하면,

어디서나 실행

∙ 객체지향언어

- 객체지향의 특징으로는 상속, 캡슐화, 다형성 등이 있음

∙ 자동 메모리 관리

- 가비지 컬렉터가 자동적으로 메모리를 관리해주기 때문에 프로그래머는 메모리를 따로

관리하지 않아도 됨

∙ 동적 로딩을 지원

- 실행 시에 모든 클래스가 로딩되지 않고 필요한 시점에 클래스를 로딩하여 사용할 수

있다는 장점

□ 연산자(operator)

∙ 연산자의 종류

- 산술 연산자

- 비교 연산자

- 논리 연산자

- 대입 연산자

∙ 연산자 우선순위 및 결합규칙

- 산술 > 비교 > 논리 > 대입 순으로 수행

- 단항 연산자 > 이항 연산자 > 삼항 연산자

- 단항 연산자와 대입 연산자를 제외한 모든 연산은 왼쪽에서 오른쪽으로 진행

- 두 피연산자의 타입을 둘 중 더 큰 타입으로 일치시킴 (ex) int + long + double => double)

- 피연산자의 타입이 int 보다 작은 타입이면 int로 반환됨 (ex) byte + short => int)

□ 객체지향언어

∙ 기본 개념

- 실제 세계는 사물(객체)로 이루어져 있으며, 발생하는 모든 사건들은 사물 간의 상호작용

∙ 객체지향언어 주요 특징

- 코드의 재 사용성이 높음 : 새로운 코드를 작성할 때 기존의 코드를 이용하여 쉽게 작성

- 코드 관리가 용이 : 코드 간의 관계를 이용하여 적은 노력으로 쉽게 코드를 변경할 수 있음

- 신뢰성이 높은 프로그래밍을 가능하게 함 : 제어자와 메서드를 이용해서 데이터를 보호하고

올바른 값을 유지하도록 하며, 중복 코드를 제거하여 코드의 불일치로 인한 오동작을 방지

∙ 장점

- 코드의 재사용성이 높고, 유지보수가 용이

- 재사용성, 중복된 코드의 제거, 유지보수

- 너무 객체지향 개념에 얽매여서 고민하기 보다는 일단 프로그램을 기능적으로 완성한 다음

보다 객체지향적으로 코드를 개선할 수 있을지를 고민하여 개선

∙ 클래스(class)와 객체(object)

- 클래스 : 객체를 정의해 놓은 것. 객체의 설계도 또는 틀. 객체를 생성하는데 사용

- 객체 : 실제로 존재하는 것, 사물 또는 개념. 객체는 클래스에 정의된 대로 생성

∙ 클래스 : 데이터와 함수의 결합 (변수 -> 배열 -> 구조체 -> 클래스)

- 변수 : '하나의 데이터를 저장'할 수 있는 공간

- 배열 : 같은 종류의 여러 데이터를 '하나의 집합으로 저장'할 수 있는 공간

- 구조체 : 서로 관련된 여러 데이터를 '종류에 관계 없이 하나의 집합으로 저장하는 공간

- 클래스 : 데이터와 함수의 결합(구조체 + 함수==메서드)

□ 변수의 종류와 특징

∙ 인스턴스 변수

- 클래스 영역에 선언되며, 클래스의 인스턴스를 생성할 때 만들어짐

- 인스턴스 변수의 값을 읽어 오거나, 저장하기 위해서는 먼저 인스턴스를 생성해야 함

- 인스턴스는 '독립적인 저장공간'을 가지므로 서로 다른 값을 가질 수 있음

- 인스턴스마다 고유한 상태를 유지해야하는 속성의 경우 인스턴스 변수로 선언

∙ 클래스 변수 (스태틱 변수)

- 클래스 변수를 선언하는 방법은 인스턴스 변수 앞에 static을 붙이면 됨

- 인스턴스 마다 독립적인 저장공간을 갖는 인스턴스 변수와는 달리, 클래스 변수는 모든

인스턴스가 공통된 저장공간(변수)을 공유하게 된다. 한 클래스의 모든 인스턴스들이

공통적인 값을 유지해야 하는 속성의 경우 클래스 변수로 선언

- 클래스 변수는 인스턴스 변수와 달리 인스턴스를 생성하지 않고도 언제라도 바로 사용할

수 있다는 특징이 있으며, '클래스이름.클래스변수'와 같은 형식으로 사용

- 클래스가 메모리에 로딩될 때 생성되어 프로그램이 종료될 때까지 유지

∙ 지역 변수

- 메서드 내에 선언되어 메서드 내에서만 사용 가능하며, 메서드가 종료되면 소멸되어 for,

while문의 블럭 내에 선언된 지역 변수는 지역변수가 선언된 블럭{} 내에서만 사용

가능하며, 블럭을 벗어나면 소멸되어 사용할 수 없게 됨

□ 스태틱 메서드와 인스턴스 메서드

∙ 인스턴스 메서드

- 인스턴스 변수와 관련된 메서드의 작업을 수행하는데 인스턴스 변수를 필요로 하는 메서드

- 인스턴스와 관계없는 메서드를 클래스 메서드(static 메서드)로 정의

- 클래스 영역에 선언된 변수를 멤버변수라고 함

- 멤버변수 중에 static이 붙은 것은 스태틱 변수, static이 붙지 않은 것은 인스턴스 변수

- 멤버변수는 instance 변수와 static 변수 모두를 총칭하는 말

∙ 클래스를 설계할 때, 멤버변수 중 모든 인스턴스에 공통적으로 사용해야 하는 것에 static붙임

- 생성된 각 인스턴스는 서로 독립적이기 때문에 각 인스턴스의 변수는 서로 다른 값을 유지

- 모든 인스턴스에서 값은 값이 유지되어야 하는 변수는 static을 붙여서 정의

∙ 스태틱 변수는 인스턴스를 생성하지 않아도 사용할 수 있음

- static이 붙은 변수는 클래스가 메모리에 올라갈 때, 이미 자동적으로 생성

∙ 스태틱 메서드는 인스턴스 변수를 사용할 수 없음

- 인스턴스 변수는 인스턴스가 반드시 존재해야 만 사용할 수 있는데, 스태틱 메서드는

인스턴스 생성 없이 호출 가능하므로, 클래스 메서드가 호출되었을 대, 인스턴스가

존재하지 않을 수도 있음. 따라서 클래스 메서드에서 인스턴스 변수와 사용을 금지

- 인스턴스 변수나 인스턴스 메서드에서는 static이 붙은 멤버들을 사용하는 것이 언제나

가능. 인스턴스 변수가 존재한다는 것은 static 변수가 이미 메모리에 존재한다는 것을 의미

□ JVM 메모리 구조

∙ 메서드(Method) 영역 -> 스태틱 영역

- 프로그램 실행 중 어떤 클래스가 사용되면 JVM은 해당 클래스의 클래스파일(\*class)을

읽어서 분석하여 클래스에 대한 정보(클래스 데이터)를 이곳에 저장

- 이때 그 클래스의 스태틱 변수도 이 영역에 함께 생성

∙ 힙(heap) 영역

- 인스턴스가 생성되는 공간. 프로그램 실행 중 생성되는 인스턴스는 모두 이곳에 생성.

인스턴스 변수들이 생성되는 공간

∙ 호출스택(call stack → stack area)

- 호출 스택은 메서드의 작업에 필요한 메모리 공간을 제공. 메서드가 호출되면 호출 스택에

호출된 메서드를 위한 메모리가 할당되며, 이 메모리는 메서드가 작업을 수행하는 동안

지역변수(매개변수 포함)들과 연산의 중간결과 등을 저장하는데 사용

- 메서드가 작업을 마치면 할당되었던 메모리 공간은 반환되며 비워짐

∙ 스택의 특징

- 메서드가 호출되면 수행에 필요한 만큼의 메모리를 스택에 할당 받음

- 메서드가 수행을 마치고 나면 사용했던 메모리를 반환하고 스택에서 제거됨

- 스택의 제일 위에 있는 메서드가 현재 실행중인 메서드임

- 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드임

□ 인터페이스 (Interface)

∙ 일종의 추상 클래스

- 인터페이스는 추상 메서드를 갖지만 추상 클래스보다 추상화 정도가 높아서 추상 클래스와

달리, 몸통을 갖춘 일반 메서드 또는 멤버변수를 구성원으로 가질 수 없음

- 추상 메서드와 상수 만을 멤버로 가질 수 있으며, 그 외 다른 어떠한 요소도 허용되지 않음

- 추상 클래스를 부분적으로만 완성된 '미완성 설계도'라고 한다면, 인터페이스는 구현된 것은

아무것도 없고 밑그림만 그려져 있는 '기본 설계도'라 할 수 있음

- 인터페이스도 추상 클래스처럼 완성되지 않은 불 완전한 것이기 때문에 그 자체만으로

사용되기 보다는 다른 클래스를 작성하는데 도움 줄 목적으로 작성됨

∙ 인터페이스 작성

- 클래스를 작성하는 것과 유사하게 키워드로, class 대신 interface를 사용한다는 것은 다름

- interface에도 클래스와 같이 접근제어자로 public을 사용할 수 있음

- 추상성은 상대적인 개념. 정확히 알 때는 클래스. 정확히 알기 전에는 인터페이스

∙ 제약사항

- 모든 멤버변수는 public static final 이어야 하며, 이를 생략할 수 있음

- 모든 메서드는 public abstract 이어야 하며, 이를 생략할 수 있음

- 단, static 메서드와 디폴트 메서드는 예외 (jdk 1.8이후로는 가능해짐)

- final은 단 한번만 할당. override는 불가

- new라는 객체를 만드는 것은 인터페이스 대상으로는 불가

∙ 인터페이스의 장점

- 개발 시간 단축 : 인터페이스가 작성되면, 이를 통해 프로그램을 작성하는 것이 가능

- 메서드를 호출하는 쪽에서는 메서드의 내용에 관계 없이 선언부만 알면 되기 때문

- 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를

구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행 가능

- 표준화가 가능 : 프로젝트의 기본 틀을 제공함으로써 보다 일관되고 정형화 된

프로그래밍이 가능

- 독립적인 프로그래밍이 가능 (다형성) : 인터페이스를 이용하면 클래스의 선언과 구현을

분리시킬 수 있기 때문에 실제 구현에 독립적인 프로그램을 작성하는 것이 가능.

한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍 가능

- 추상성 : 공통되는 요소들(ex) 동물의 뼈, 다리 등)의 모음. 몇몇에만 존재하는 차별점은

추상성의 대상에서 제외

□ 오버라이딩(Overriding)

∙ 상속받은 메서드를 그대로 사용하기도 하지만, 자식 클래스 자신에 맞게 변경해야하는 경우가

많고, 이런 경우, 부모의 메서드를 오버라이딩 함

∙ 오버라이딩 조건

- 자식클래스에서 오버라이딩하는 메서드는 부모 클래스의 메서드와 이름이 같아야 함

- 매개변수가 같아야 하고, 반환타입이 같아야 함

- 인스턴스 메서드를 static메서드로 또는 그 반대로 변경할 수 없음

∙ 오버로딩 vs 오버라이딩

- 오버로딩 : 기존에 없는 새로운 메서드를 추가하는 것 (new)

- 오버라이딩 : 부모로부터 상속받은 메서드의 내용을 변경하는 것 (change, modify)

□ 상속, 다형성, 캡슐화

∙ 상속(Inheritance)

- 기존의 클래스를 재사용하여 새로운 클래스를 작성하는 것

- 상속을 통해서 클래스를 작성하면 보다 적은 양의 코드로 새로운 클래스를 작성할 수 있고,

코드를 공통적으로 관리할 수 있기 때문에 코드의 추가 및 변경이 매우 용이

- 상속 구현 방법 class 클래스명 extends parentClass명{ }

- 조상 클래스 : 부모(parent) 클래스, 상위(super) 클래스 등

- 자손 클래스 : 자식(child) 클래스, 하위(sub) 클래스 등

- 자식 클래스에 새로운 코드가 추가되어도 부모클래스에는 아무런 영향도 받지 않음

- 부모 클래스가 변경되면 자동적으로 자식에게 영향을 주게 됨

- 자식 클래스는 부모 클래스의 모든 멤버를 상속받으므로 항상 부모 클래스 보다 같거나

많은 멤버를 갖음

- 상속을 받는다는 것은 부모 클래스를 확장(extend)한다는 의미로 해석할 수도 있으며,

이것이 상속에 사용되는 키워드인 이유

- 생성자는 상속되지 않는다. 멤버만 상속됨

- 자식 클래스의 멤버 개수는 부모 클래스보다 항상 같거나 많음

- 클래스의 상속은 무조건 단일 상속임

∙ 패키지(package)

- 클래스의 묶음. 서로 관련된 클래스들끼리 그룹 단위로 묶어 놓음으로써 클래스를

효율적으로 관리할 수 있음

- 다른 개발자가 개발한 클래스 라이브러리의 클래스와 이름이 충돌하는 것을 피할 수 있음

- 클래스의 실제 이름은 패키지명을 포함한 것 (full name)

- 같은 이름의 클래스 일지라도 서로 다른 패키지에 속하면 패키지명으로 구별이 가능함

- 클래스가 물리적으로 하나의 클래스파일(.class)인 것과 같이 패키지는 물리적으로 하나의

디렉토리

- 하나의 소스파일에는 첫 번쨰 문장으로 단 한 번의 패키지 선언만을 허용함

- 모든 클래스는 반드시 하나의 패키지에 속해야 함

- 패키지는 점(.)을 구분자로 하여 계층구조로 구성할 수 있음

- 패키지는 물리적으로 클래스 파일(.class)을 포함하는 하나의 디렉토리임

- 패키지 선언문은 반드시 소스파일에서 첫 번째 문장이어야 하며, 하나의 소스파일에

단 한 번만 선언될 수 있음

- 패키지명은 대소문자를 모두 허용하지만, 클래스명과 쉽게 구분하기 위해서 소문자 표기를

원칙으로 함

(ex) indexLast (X)

index.last or indexlast.blah.blah (O))

- 모든 클래스는 반드시 하나의 패키지에 포함되어야 함

∙ import문

- 소스코드를 작성할 때, 다른 패키지의 클래스를 사용하려면 패키지명이 포함된 클래스

이름을 사용해야 함

- 매번 패키지명을 붙여서 작성하기란 불편하기에, 클래스의 코드를 작성하기 전에

import문으로 사용하고자 하는 클래스의 패키지를 미리 명시해주면 소스코드에 사용되는

클래스 이름에서 패키지명은 생략할 수 있음

- 이클립스 단축키 ctrl + shift + o를 누르면 자동으로 import 문을 추가해줌

- import 선언

일반적인 소스파일(\*.java)의 구성은 package문 → import문 → 클래스 선언

import 패키지명.클래스명; (하나의 클래스를 불러옴)

or import 패키지명; (패키지에 속한 모든 클래스를 불러옴)

∙ 다형성(Polymorphism)

- 상속과 함께 객체지향개념의 중요한 특징인 다형성은 상속과 깊은 관계가 있으므로 꼭

상속에 대해 알고 있어야 함

- 다형성이란 여러 가지 형태를 가질 수 있는 능력을 의미하며, Java에서는 한 타입의 참조

변수로 여러 타입의 객체를 참조할 수 있도록 함

- 조상 클래스 타입의 참조변수로 자손 클래스의 인스턴스를 참조할 수 있도록 하였다는 것

- Tv와 CaptionTv 클래스가 서로 상속관계에 있을 경우, 부모 클래스 타입의 참조변수로

자식 클래스의 인스턴스를 참조하도록 하는 것이 가능

- Tv 타입의 참조변수로는 CaptionTv인스턴스 중에서 Tv클래스의 멤버들만 사용할 수 있음

- 둘 다 같은 타입의 인스턴스지만, 참조변수의 타입에 따라 사용할 수 있는 멤버의 개수가

달라짐

- 핵심 : 부모 타입의 참조변수로 자식 타입의 인스턴스를 참조할 수 있고, 반대로 자식

타입의 참조변수로 부모 타입의 인스턴스를 참조할 수는 없음

□ 예외처리(Exception handling)

∙ 프로그램 오류

- 프로그램 실행 중 어떤 원인에 의해서 오작동 하거나 비정상적으로 종료되는 경우에,

이러한 결과를 초래하는 원인을 프로그램 에러 또는 오류라고 함

- 발생시점에 따라 '컴파일 에러'(compile-time error)와 '런타임에러'(run-time error)로 나뉨

- 컴파일 에러 : 컴파일 시에 발생하는 에러. 코드 작성 시점의 에러로 생각

(ex) 작성한 코드 자체의 오류)

- 런타임 에러 : 실행 시에 발생하는 에러. 실행 시점의 에러. 방지를 위해서 프로그램의 실행

도중 발생할 수 있는 모든 경우의 수를 고려하여 이에 대한 대비를 하는 것이 필요

(ex) 일부 코드가 실행되고, 에러가 발생)

- '논리적 에러'(logical error)는 컴파일 및 실행 문제는 없지만, 의도와는 다르게 동작하는 것

(ex) 음수의 재고, 게임에서 총알을 맞아도 죽지 않는 경우 등)

- 컴파일러가 알려준 에러들은 모두 수정해서 컴파일을 성공적으로 마치고 나면 클래스 파일

(\*.class)이 생성되고, 생성된 클래스 파일을 실행할 수 있게 됨

- 자바에서 실행 시(runtime) 발생할 수 있는 프로그램 오류를 에러와 예외로 구분

- 에러(Error) : 메모리 부족(Out of Memory Error)이나 스택오버플로우(Stack Overflow Error)

같은 일단 발생하면 복구할 수 없는 심각한 오류

- 에러가 발생하면, 프로그램의 비정상적인 종료를 막을 길이 없지만, 예외는 발생하더라도

프로그래머가 적절한 코드를 통해 프로그램의 비정상적인 종료를 막을 수 있음

∙ 예외 클래스의 계층구조

Object

Throwable

Exception Error

IOException RuntimeException OutOfMemoryError

ArithmeticException

∙ try블럭 내에서 예외가 발생한 경우

- 발생한 예외와 일치하는 catch문이 있는지 확인

- 일치하는 catch문을 찾게 되면, 그 catch 블럭 내의 문장들을 수행하고, 전체 try-catch문을

빠져나가서 그 다음 문장을 계속해서 수행. 일치하는 catch문을 찾지 못하면, 예외는

처리되지 못함

∙ try문 내에서 예외가 발생하지 않는 경우

- catch문을 거치지 않고 전체 try-catch문을 빠져나가서 계속 수행

∙ finally 문장은 위의 어느 상황에서도 무조건 수행됨

- try문 끝나고 수행 / catch문 끝나고 수행

∙ 예외처리하기

- try catch 문 : 에러는 어쩔 수 없지만, 예외는 미리 처리해주어야 함

∙ 예외처리의 정의

- 정의 : 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것

- 목적 : 프로그램의 비정상적인 종료를 막고, 정상적인 실행 상태를 유지하는 것

expression

try { (예외가 발생할 가능성이 있는 문장) 트랜잭션 단위들

} catch(Exception1 e1) {

exception1이 발생했을 경우, 이를 처리하기 위한 문장을 적는다

} catch(Exception2 e2) {

exception2이 발생했을 경우, 이를 처리하기 위한 문장을 적는다

} catch(ExceptionN n) {

exceptionN이 발생했을 경우, 이를 처리하기 위한 문장을 적는다

}

□ 컬렉션 프레임워크 (Collection Framework)

∙ 컬렉션 프레임워크란?

- 데이터 군을 저장하는 클래스들을 표준화 한 설계

- 컬렉션은 다수의 데이터, 즉 데이터 그룹을, 프레임웍은 표준화 된 프로그래밍 방식을 의미

- 컬렉션 프레임워크는 다수의 데이터들을 다루는 데 필요한 다양하고 풍부한 클래스들을

제공하기 때문에 프로그래머의 부담을 덜어줄 수 있음

- 인터페이스와 다형성을 이용한 객체지향적 설계를 통해 표준화되어 있기 때문에 이를

활용하면 재사용성이 높은 객체지향적 코드를 작성할 수 있다는 장점

∙ 핵심 인터페이스

- 컬렉션 프레임워크에서는 크게 3가지 타입이 존재 (List, Set, Map)

- 이 중 List와 Set은 공통적인 특징이 있어 Collection을 추가로 정의

∙ 컬렉션 프레임워크의 핵심 인터페이스와 특징

- 인터페이스 특징 : List 순서가 있는 데이터의 집합, 데이터의 중복을 허용 (ex) 대기자 명단)

- 구현클래스 : ArrayList, LinkedList, Stack, Vector 등 Set 순서를 유지하지 않는 데이터의

집합, 데이터의 중복을 허용하지 않음

- 구현클래스 : HashSet, TreeSet 등 Map 키(key)와 값(value)의 쌍(pair)으로 이루어진

데이터의 집합

- 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용

(ex) 우편번호, 지역번호(전화번호))

- 구현클래스 : HashMap, TreeMap, HashTable, Property 등

□ 맵(map)

- 맵 인터페이스는 key와 value를 하나의 쌍으로 묶어서 저장하는 컬렉션 클래스를 구현하는

데 사용됨

- 키는 중복될 수 없지만 값은 중복을 허용

- 기존에 저장된 데이터와 중복된 키와 값을 저장하면 기존의 값은 없어지고, 마지막에

저장된 값이 남게 됨

- Map 인터페이스를 구현한 클래스로는 Hashtable, HashMap, TreeMap 등이 있음

- void clear( ) : map의 모든 객체를 삭제

- Object get(Object key) : 지정한 key객체에 대응하는 value객체를 찾아서 반환

- boolean isEmpty( ) : map이 비었는지 확인

- Object put(Object key, Object value) : map에 value객체를 key객체에 연결하여 저장

- Object remove(Object key) : 지정한 key객체와 일치하는 value객체를 삭제

- int size( ) : 저장된 개수를 반환

□ 추가 내용

- 변수 – 진법

- 연산자 - 시프트 연산자

- 조건문, 반복문 - Do-While

- 다차원 배열

- 재귀 호출

- 클래스 - Object클래스, Abstract클래스

- 예외처리(사용자 정의)

- 버퍼, 빌드

- 래퍼 클래스

- 정규식

- 형식화

- 심플데이트 포맷

- 스택 - 큐 - 힙

- enum (열거형)

- 쓰레드

- 람다식

- 입출력

- 직렬화

- 네트워킹