〈자바 정리본〉
소스 파일
: 프로그래밍 언어로 작성된 텍스트 파일
컴파일
소스파일을 컴퓨터가 이해할 수 있는 기계어로 만드는 과정

〈자바의 역사〉
1991년 그린 프로젝트
썬마이크로시스템즈의 제임스 고슬링에 의해 시작
(가전제품에 들어갈 소프트웨어를 위해 개발)
1995년에 자바 발표

목적
플랫폼 호환성 문제 해결
: 기존 언어로 작성된 프로그램은 PC, 유닉스 메인 프레임 등
플랫폼 간에 호환성이 없음
: 다시 컴파일하거나 소스 코드를 재작성해야 하는 단점

플랫폼 독립적인 언어 개발
: 모든 플랫폼에서 호환성을 갖는 프로그래밍 언어 필요
: 네트워크, 특히 웹에 최적화된 프로그래밍 언어 필요성 대두

메모리 용량이 적고 다양한 플랫폼을 가지는 가전 제품에 적용
: 내장형 시스템 요구 충족

자바 초기 이름 : 오크
: 웹 브라우저 Netscape에서 실행
현재 오라클에서 자바를 지원
: 2009년 썬마이크로시스템즈를 오라클에서 인수

〈기존 언어의 플랫폼 종속성〉
플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

프로그램의 플랫폼 호환성 없는 이유
: 기계어가 CPU마다 다름
: 운영체제마다 API(클래스 라이브러리) 다름
: 운영체제마다 실행파일 형식 다름

〈자바의 플랫폼 독립성 (WORA)〉

WORA (Write Once Run Anywhere)

: 한 번 작성된 코드는 모든 플랫폼에서 실행

: C/C++ 등 기존 프로그래밍 언어가 가진 플랫폼 종속성 극복

(OS, H/W에 상관없이 자바 프로그램이 동일하게 실행)

: 네트워크에 연결된 어느 클라이언트에서나 실행

(웹 브라우저, 분산 환경 지원)

WORA를 가능하게 하는 자바의 특징

- 바이트 코드

: 자바 소스를 컴파일한 목적 코드 -> CPU에 종속적이지 않은 중립적인 코드

- JVM (Java Virtual Machine)

: 자바 바이트 코드를 실행하는 자바 가상머신 (소프트웨어)

: 다양한 플랫폼에 대해 각각 적절한 JVM이 제공됨

〈자바의 실행 환경〉

바이트 코드

: 클래스 파일(.class)에 저장

: 자바 가상 기계에서 실행 가능한 바이너리 코드

- 바이트 코드는 컴퓨터 CPU에 의해 직접 실행되지 않음
- 자바 가상머신(JVM)이 인터프리터 방식으로 바이트 코드를 해석하여 실행

자바 가상머신(JVM)

: 자바 가상머신 자체는 설치되는 하드웨어 및 운영체제 플랫폼에 종속적이지만,
그를 통해 제공되는 자바 실행 환경은 어디서나 동일

: 자바 가상머신은 자바 개발사인 오라클 외 IBM, MS 등 다양한 회사에서 공급
(바이트 코드를 즉석으로 변환하지만, 컴퓨터마다 상이)

JDK(Java Development Kit)

: 자바 응용 개발 환경으로 개발에 필요한 도구 포함

: 컴파일러, JRE, 클래스 라이브러리(API), 샘플 등 포함

JRE(Java Runtime Environment)

: 자바 실행 환경으로 JVM을 포함 (JRE만 따로 설치 가능)

JDK의 bin 디렉터리에 포함된 주요 개발 도구

javac	자바 소스를 바이트 코드로 변환하는 컴파일러
java	JRE의 bin디렉터리에도 있는 자바 응용프로그램 실행기
javadoc	자바 소스로부터 HTML 형식의 API 도큐먼트 생성
jar	자바 아카이브 파일(JAR)(압축)의 생성 및 관리하는 유틸리티
jmod	자바의 모듈 파일(.jmod)생성 및 모듈 파일의 내용 출력
jlink	응용프로그램에 맞춤 맞춤형(custom) JRE 제공
jdb	자바 응용프로그램의 실행 중 오류를 찾는 데 사용하는 디버거
javap	클래스 파일의 바이트 코드를 소스와 함께 보여주는 디어셈블러

JDK 디렉터리 구조

bin 자바 개발, 실행에 필요한 도구와 유틸리티 명령
conf 여러 종류의 배치 파일
include 네이티브 코드 프로그래밍에 필요한 C언어 헤더 파일
jmods 컴파일된 모듈 파일들
legal 각 모듈에 대한 저작권과 라이선스 파일
lib 실행 시간에 필요한 라이브러리 클래스들

〈자바의 배포판 종류〉

Java SE

: 자바 표준 배포판
: 데스크탑과 서버 응용 개발 플랫폼

Java ME

: 마이크로 배포판
: 휴대폰, 셋톱박스 등 제한된 리소스를 갖는 하드웨어를
위한 응용 개발 플랫폼
: Java SE의 서브셋 +임베디드 및 가전 제품을 위한 API정의

Java EE

: 기업용 배포판
: 다중 사용자, 기업용 응용 개발을 위한 플랫폼
: Java SE + 인터넷 기반의 서버 사이드 컴퓨팅 관련 API추가

〈자바의 모듈 프로그래밍〉
모듈화 : Java 9부터 등장한 새로운 기능 (17년 9월)
모듈 : 자바 패키지와 이미지, XML 파일 등 리소스를 묶은 단위
모듈 프로그래밍
: 자바 응용 프로그램을 마치 직소 퍼즐을 연결하듯이
필요한 모듈을 연결하는 방식으로 작성

자바 플랫폼의 모듈화
: 실행 시 사용되는 자바 API의 모든 클래스들을 모듈들로 분할

모듈화의 목적
: 세밀한 모듈화 실현, 자바 응용 프로그램이 실행되는데
필요하지 않은 모듈이 포함되는 것을 배제
: 하드웨어가 열악한 소형 IoT 장치 기원 (가벼운 실행 환경)
: 하지만 모듈방식이 아닌 기존 방식대로 프로그래밍해도 무관
(응용 프로그램 작성 시 꼭 모듈 방식으로 작성할 필요 없음)

〈자바 API〉
API(Application Programming Interface)
: JDK에 포함된 클래스 라이브러리
(주요한 기능들을 미리 구현한 클래스 라이브러리의 집합)
: 개발자는 API를 이용하여 쉽고 빠르게 자바 프로그램 개발
: API에서 정의한 구격에 따라 클래스 사용

자바 패키지(package)
: 서로 관련된 클래스들을 분류하여 묶어 놓은 것 -〉 계층 구조
(클래스의 이름에 패키지 이름도 포함
다른 패키지에 동일한 이름의 클래스 존재 가능)
: 자바 API(클래스 라이브러리)는 JDK에 패키지 형태로 제공
(필요한 클래스가 속한 패키지만 import하여 사용)
: 개발자 자신의 패키지 생성 가능

〈자바 IDE - 이클립스〉
IDE(Integrated Development Environment)
: 통합 개발 환경
: 편집, 컴파일, 디버깅을 한번에 할 수 있는 통합된
개발 환경

이클립스
: 자바 응용 개발을 위한 통합 개발 환경
: IBM에 의해 개발된 오픈 소스 프로젝트

〈자바 응용의 종류〉
데스크탑
: 가장 전형적인 자바 응용프로그램
: JRE가 설치된 어떤 환경에서나 실행
: 다른 응용프로그램의 도움 없이 단독으로 실행

애플릿
: 웹 브라우저에 의해 구동되고 실행이 제어되는
자바 프로그램
: 애플릿은 사용할 수 있는 자원 접근에 제약 있음

서블릿
: 애플릿과 반대로 서버에서 실행되는 자바 프로그램
(서버 클라이언트 모델에서는 서블릿과 애플릿이
각각 통신하면서 실행)
: 데이터베이스 서버 등과 연동하는
복잡한 기능 구현 시 사용
: 사용자 인터페이스가 필요 없는 응용
: 웹 서버에 의해 실행 통제 받음

안드로이드
: 구글의 주도로 여러 모바일 회사가 모여 구성한
OHA(Open Handest Alliance)에서 만든 무료 모바일 플랫폼
: 안드로이드에서 JVM에 해당하는
ART(안드로이드 런타임)는 기존 자바 바이트 코드와
호환성이 없어 변환 필요

〈자바의 주요 특징〉

플랫폼 독립성

: 자바 가상기계가 바이트 코드 실행

: 플랫폼(하드웨어, 운영체제)에 종속성을 갖지 않음

클래스로 캡슐화

: 자바의 모든 변수나 함수는 클래스 내에 선언

: 클래스 안에서 클래스(내부 클래스) 작성 가능

객체지향

: 클래스 계층 구조, 상속, 다형성, 캡슐화 등 지원

소스(.java)와 클래스(.class) 파일

: 하나의 소스 파일에 여러 클래스를 작성 가능
(하나의 public 클래스만 가능)

: 소스 파일 이름과 public으로 선언된 클래스 이름은 같아야 함

: 클래스 파일에는 단 하나만의 클래스만 존재
(다수의 클래스를 가진 자바 소스를 컴파일하면 클래스마다
별도 클래스 파일이 생성됨)

실행 코드 배포

: 한 개의 class 또는 다수의 class 파일로 구성

: 여러 폴더에 걸쳐 다수의 클래스 파일로 구성된 경우
jar 파일 형태로 배포

main() 메소드

: 자바 응용프로그램의 실행은 main() 메소드에서 시작

: 하나의 클래스 파일에 하나 이상의 main() 메소드가 있을 수 없음
(각 클래스 파일이 main() 메소드를 갖는 것은 가능)

패키지

: 서로 관련 있는 여러 클래스를 패키지로 묶어 관리

: 패키지는 폴더 개념
예) java.lang.System.은 java₩lang 디렉터리의 System.class 파일

멀티스레드
: 여러 스레드의 동시 수행 환경 지원
: 자바는 운영체제의 도움 없이 자체적으로 멀티스레드 지원
: C/C++ 프로그램은 멀티스레드를 위해 운영체제 API를 호출

가비지 컬렉션
: 자바 언어는 메모리 할당 기능은 있지만, 반환 기능이 없음
: 사용하지 않는 메모리는 JVM에 의해 자동 반환됨
= 가비지 컬렉션

실시간 응용그램에 부적합
: 실행 도중 예측할 수 없는 시점에 가비지컬렉션 실행 때문
: 응용프로그램의 일시적 중단 발생

자바 프로그램은 안전
: 타입 체크 엄격
: 물리적 주소를 사용하는 포인터 개념 없음

프로그램 작성 쉬움
: 포인터 개념이 없음
: 동적 메모리 반환하지 않음: 다양한 라이브러리 지원
실행 속도 개선을 위한 JIT 컴파일러 사용
: 자바는 바이트 코드를 인터프리터 방식으로 실행
(기계어가 실행되는 것보단 느림)
: JIT 컴파일 기법으로 실행 속도 개선

JIT 컴파일
: 실행 중에 바이트 코드를 기계어 코드로 컴파일하여
기계어를 실행하는 기법

〈자바 프로그램 기본 구조〉
클래스
: class 키워드로 선언
: public으로 선언하면 다른 클래스에서 접근 가능

주석문
// 한 라인 주석
/* 여러 행 주석 */

main() 메소드
: public static void로 선언
: String[] args로 실행 인자를 전달 받음

필드
: 클래스의 멤버 변수

메소드
: C/C++에서의 함수를 메소드로 지칭
: 클래스 바깥에 작성할 수 없음

변수
지역 변수 : 메소드 내에서 선언된 변수
: 메소드 실행이 끝나면 저장 공간 반환

〈식별자〉
: 클래스, 변수, 상수, 메소드 등에 붙이는 이름

식별자의 원칙
: @, #, ! 와 같은 특수 문자, 공백 또는 탭은 식별자로 사용할 수 없으나
_, $ 는 사용 가능
: 유니코드 문자 사용 가능 (한글 가능)
: 자바 언어의 키워드는 사용 불가
: 첫 번째 문자로 숫자는 사용 불가
: 불린(true, false)과 널(null) 등의 리터럴은 식별자로 사용 불가
: 길이 제한 없음

〈자바의 데이터 타입〉
레퍼런스 타입 : 1개 (3가지 용도)
: 클래스에 대한 레퍼런스
: 인터페이스에 대한 레퍼런스
: 배열에 대한 레퍼런스

문자열
: 문자열은 기본 타입이 아님
: String 클래스로 문자열 표현
리터널
: 프로그램에서 직접 표현한 값
: 정수, 실수, 문자, 논리, 문자열 리터럴이 있음

〈기본 타입 외의 리터럴〉
null 리터럴
: 레퍼런스에 대입하여 사용 가능
예) String str = null;

문자열 리터럴
: 자바에서 문자열은 객체이므로 기본 타입이 아님
: String 객체로 자동 처리됨
예) String str = “Good”;

〈변수의 타입〉
상수
: final 키워드 사용
: 선언 시 초기값 지정

var 키워드 사용
: 타입을 생략하고 변수 선언 가능
: 컴파일러가 추론해 변수 타입 결정
(초기값이 없다면 컴파일 오류)
: var은 지역변수 선언만 한정

〈키 입력〉
System.in
: 키보드와 연결된 자바의 표준 입력 스트림
: 입력되는 키를 바이트(문자 X)로 리턴하는 저수준 스트림
: 문자나 숫자 변환은 어려움

〈Scanner 클래스와 객체〉

Scanner 클래스

: 읽은 바이트를 문자, 정수, 실수, 불린, 문자열 등 다양한 타입으로 변환하여 리턴

예) java.util.Scanner

: 객체 생성 가능

: 키보드에 연결된 System.in에게 키를 읽게 하고, 원하는 타입으로 변환하여 리턴

: 입력되는 키 값을 공백으로 구분되는 토큰 단위로 읽음

Scanner 클래스의 주요 메소드

String next( )	다음 토큰을 문자열로 리턴
String nextLine( )	'₩n'을 포함하는 한 라인을 읽고 '₩n'을 버린 나머지만 리턴
void close( )	Scanner의 사용 종료
Boolean hasNext( )	현재 입력된 토큰이 있으면 true, 아니면 새로운 입력이 들어올 때까지 무한정 기다려서, 새로운 입력이 들어오면 그 때 true 리턴. ctrl-z 키가 입력되면 끝이므로 false 리턴

연산의 종류

증감	++ --
산술	+ - * / %
시프트	〉〉 〈〈 〉〉〉
비교	〉 〈 〉= 〈= == !=
비트	& | ^ ~
논리	&& || ! ^
조건	? :
대입	= *= /= -= &= ^= |= 〈〈= 〉〉= 〉〉〉=

대입 연산자

a & b	a와 b의 각 비트들의 AND연산. 모두 1일 때만 1이며 나머지는 0
a | b	a와 b의 각 비트들의 OR연산. 모두 0일 때만 0이며 나머지는 1
a ^ b	a와 b의 각 비트들의 XOR연산. 서로 다르면 1, 같으면 0
~ a	단항연산자로서 a의 각 비트들에 NOT 연산. 1을 0으로 0을 1로
a 〈〈 b	a의 비트를 왼쪽으로 b번 시프트. 최하위 비트 빈자리는 0으로
a 〉〉 b	오른쪽으로 b번 시프트. 최상의 비트 빈자리 전 최상위 비트로 채움
a 〉〉〉 b	오른쪽으로 b번 시프트. 최상위 비트 빈자리는 0으로 채움

〈〈 : 1비트 왼쪽으로 시프트할 때마다 2로 곱하는 결과
  : 음수(최상위 비트가 1)는 시프트 결과 최상위 비트가 0인 양수가 되는 오류 발생 주의
〉〉 : 1비트 오른쪽으로 시프트할 때마다 2로 나누기하는 결과
〉〉〉 : 시프트 시 최상위 비트에 항상 0이 삽입
   : 나누기의 산술적 효과가 나타나지 않음 (논리적 오른쪽 시프트)

* 객체, 메소드, 필드 사용법
예) if(text.equals("exit"))
    (객체.메소드(필드))

〈배열 선언과 생성〉
(1) 배열에 대한 레퍼런스 intArray 선언
int intArray [ ];
- 2차원 배열
int intArray [ ][ ];

(2) 배열 생성
intArray = new int [5];
- 2차원 배열
intArray = new int [2][5];

- 선언과 동시에 배열을 생성하는 방법

```
int intArray[ ] = new int [5];
int intArray[ ] = new int [2][5];    // 2차원 배열
```

- 배열 생성과 값 초기화

```
int intArrya[ ] = { 4, 3, 2, 1, 0 };
// 5개의 정수 배열 생성 및 초기화
```

〈레퍼런스 치환과 배열 공유〉

```
int intArray[ ] = new int[5];
int myArray[ ] = intArray;
```

=〉 레퍼런스 치환으로 배열 공유

〈배열의 크기, length 필드〉

자바의 배열은 객체로 처리

: 배열의 크기는 배열 객체의 length 필드에 저장

〈for-each문〉

: 배열이나 나열의 각 원소를 순차적으로 접근하는데 유용한 for문

```
int [ ] n = { 1,2,3,4,5 };
int sum = 0;
for (int k : n) {
    sum += k;
}
```

= for문으로 구성하면 이와 같다.

```
for(int i=0; i<n.length; i++) {
    int k=n[i];
    sum += k;
}
```

〈메소드에서 배열 리턴〉

배열의 레퍼런스만 리턴

: 배열 전체가 리턴되는 것이 아님

메소드의 리턴 타입

: 리턴하는 배열 타입과 리턴받는 배열 타입이 일치

: 리턴 타입에 배열의 크기를 지정하지 않음

```
int[ ] makeArray( ) {
        int temp[ ] = new int[4];
        return temp;
}

int intArray[ ];
intArray = makeArray;      // 메소드가 리턴한 배열 참조 (temp 치환)
```

〈자바의 예외 처리〉

예외(Exception)

: 실행 중 오동작이나 결과에 악영향을 미치는 예상치 못한 상황

(자바에서는 실행 중 발생하는 에러를 예외로 처리)

: 실행 중 예외가 발생 -〉 응용프로그램이 예외를 처리하도록 호출

: 응용프로그램이 예외를 처리하지 않을 경우 프로그램을 강제 종료 시킴

: try-catch-finally 문을 사용해 예외에 대응

예외 클래스

예외 타입(예외 클래스)	예외 발생 경우	패키지
ArithmeticException	정수를 0으로 나눌 때 발생	java.lang
NullPointException	null 레퍼런스를 참조할 때 발생	java.lang
ClassCastException	변환할 수 없는 타입으로 객체를 변환할 때 발생	java.lang
OutOfMemoryError	메모리가 부족한 경우 발생	java.lang
ArrayIndexOutOfBoundException	배열의 범위를 벗어난 접근 시 발생	java.lang
IllegalArgumentException	잘못된 인자 전달 시 발생	java.lang
IOException	입출력 동작 실패 또는 인터럽트 시 발생	java.io
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생	java.lang
InputMismatchException	Scanner 클래스의 nextInt( )를 호출하여 정수로 입력받고자 했지만, 사용자가 문자를 입력한 경우	java.util

〈객체〉

실세계 객체의 특징

: 객체마다 고유한 특성(state)와 행동(behavior)를 가짐

〈객체 지향의 특성〉

캡슐화

: 객체를 캡슐로 싸서 내부를 볼 수 없게 하는 것

: 객체의 가장 본질적인 특징 (외부의 접근으로부터 객체 보호)

자바의 캡슐화

클래스(class)

: 객체 모양을 선언한 틀 (캡슐화하는 틀)

: 클래스 내에 메소드(하는 일)와 필드(특징, 아는 것) 구현

: 객체를 만들기 위함 -〉 객체의 설계도 혹은 틀

예) 클래스: 소나타 자동차 객체: 출고된 실제 소나타 100대

객체

: 생성된 실체(instance)

: 캡슐화를 통해 외부로부터의 접근을 제한

상속 (extends 키워드 사용)

: 상위(조상) 개체의 속성이 하위 개체에 물려짐

: 하위(후손) 개체가 상위 개체의 속성을 모두 가지는 관계

자바의 상속

: 상위 클래스의 멤버(필드, 메소드)를 하위 클래스가 물려받음

: 상위 클래스 (= 슈퍼 클래스)

: 하위 클래스 (= 서브 클래스)

  수퍼 클래스 코드의 재사용, 새로운 특성 추가 가능 (확장)

다형성

: 같은 이름의 메소드가 클래스 혹은 객체에 따라 다르게 구현됨

메소드 오버로딩

: 한 클래스 내에서 같은 이름이지만 다르게 작동하는 여러 메소드

메소드 오버라이딩
: 슈퍼 클래스의 메소드를 동일한 이름으로 서브 클래스마다 다르게 구현

〈객체 지향 언어의 목적〉
소프트웨어의 생산성 향상
: 상속, 다형성, 객체, 캡슐화 등 소프트웨어 재사용을 위한 여러 장치 내장

실세계에 대한 쉬운 모델링
: 절차나 과정보다 실제 일과 관련된 물체(객체)들의
상호 작용으로 묘사하는 것이 용이

절차 지향 프로그래밍
: 작업 순서를 표현하는 컴퓨터 명령 집합
: 함수들의 집합으로 프로그램 작성
객체 지향 프로그래밍
: 수행할 작업을 객체들 간의 상호 작용으로 표현
: 클래스 혹은 객체들의 집합으로 프로그램 작성

〈클래스와 객체〉
클래스
: 객체의 속성(state)와 행위(behavior) 선언
: 객체를 만들기 위함 -〉 객체의 설계도 혹은 틀
객체 : 클래스의 틀로 찍어낸 실체(instance)

〈생성자〉
생성자
: 객체가 생성될 때 초기화 목적으로 실행되는 메소드
: 객체가 생성되는 순간에 자동 호출

특징
생성자 이름은 클래스 이름과 동일
생성자는 여러 개 작성 가능(생성자 중복)
- 꼭 달라야 하는 것 : 타입, 인자 개수
생성자는 객체 생성시 한 번만 호출
생성자는 리턴 타입을 지정할 수 없음

기본 생성자(default constructor)
: 매개 변수 없고, 아무 작업 없이 단순 리턴하는 생성자
: 디폴트 생성자

기본 생성자가 자동 생성되는 경우
- 클래스에 생성자가 하나도 선언되어 있지 않을 때
  컴파일러에 의해 자동 생성

기본 생성자가 자동 생성되지 않는 경우
- 클래스에 생성자가 하나도 선언되어 있을 때
  컴파일러에 의해 자동 생성해 주지 않음

this 레퍼런스
: 객체 자신에 대한 레퍼런스
: 컴파일러에 의해 자동 관리, 개발자는 사용하기만 하면 됨
: this.멤버 형태로 멤버를 접근할 때 사용
this( )로 다른 생성자 호출
this( )
: 같은 클래스의 다른 생성자 호출
: 생성자 내에서만 사용 가능 (일반 메소드에서는 사용 불가)
: 반드시 생성자 코드의 제일 처음에 수행

〈객체 배열〉
자바의 객체 배열
: 객체에 대한 레퍼런스 배열

자바의 객체 배열 만들기 3단계
- 배열 레퍼런스 변수 선언
- 레퍼런스 배열 생성
- 배열의 각 원소 객체 생성