☞ **썬 마이크로시스템즈에서 개발하여 1996년에 발표한 객체지향 프로그래밍 언어.**

1. 역사

● 썬의 엔지니어들이 고안한 **오크(Oak)**라는 언어에서 시작
● **C++**의 장점을 도입하고 단점을 보안한 언어
● 처음에는 **가전제품이**나 PDA와 같은 **소형기기**에 탑재될 소프트웨어를 만들 때 사용될 목적
● **운영체제에 독립적인 성격** 덕분에 인터넷에 적합하도록 개발 방향을 바꾸면서 이름을 자바(JAVA)로 변경
● 서버 쪽 프로그래밍을 위한 **서블릿(Servelet), JSP(JAVA Server Pages)**, 구글의 스마트폰 운영체제인 **안드로이드**에서 사용됨
● 앞으로는 자바의 원래 목표였던 **소규모 가전제품과 대규모 기업환경은 위한 소프트웨어 개발 분야**에 활발히 사용될 것으로 기대됨

2. 특징

1) 운영체제에 독립적(write once, run anywhere)

자바 응용프로그램은 **자바가상머신(JVM)**하고만 통신하고 JVM이 전달 받은 명령을 해당 운영체제가 이해할 수 있도록 변환하여 운영체제와 하드웨어에 전달하기 때문에, 자바는 **운영체제/하드웨어 관계없이 실행이 가능**하다(독립적이다). 하지만 JVM은 운영체제에 종속적이어서 각 운영체제별로 다든 버전이 있다.

2) 객체지향언어(Object-Oriented Programming Langauge)

객체지향의 특징인 **상속, 캡슐화, 다형성**이 잘 적요된 순수한 객체지향언어.

3) 자동 메모리 관리(Garbage Collection)

가비지컬렉터(garbage collector)가 자동적으로 메모리 관리를 해 주어 프로그래머는 따로 메모리를 체크하고 반환하는 일을 수동적으로 처리하지 않아도 된다.

4) 네크워크와 분산처리를 지원

풍부하고 다양한 네트워크 프로그래밍 라이브러리(**Java API**)을 통해 네트워크 관련 프로그램을 쉽게 개발할 수 있도록 지원한다.

5) 멀티쓰레드(multi-thread) 지원

운영체제와는 관련없이 구현 가능하며, 관련된 라이브러리(Java API)가 제공되므로 구현이 쉽다. 여러 쓰레드에 대한 **스케줄링(scheduling)**을 **자바 인터프리터**가 담당하게 된다.

6) 동적 로딩(Dynamic Loading) 지원

실행 시에 모든 클래스가 로딩되지 않고 **필요한 시점에 클래스를 로딩**하여 사용할 수 있는 장점이 있다. 일부 클래스가 변경되어도 전체 애플리케이션을 **다시 컴파일하지 않아도 된다**.

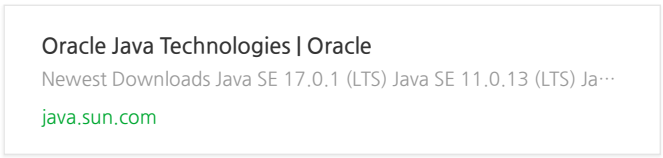
3. JVM (Java Virtual Machine)

● JVM은 **자바를 실행하기 위한 가상 기계 또는 소프트웨어로 구현된 하드웨어**, 즉, 컴퓨터 속의 컴퓨터.
● 일반 애플리케이션은 OS와 바로 맞붙어 있기 때문에 OS종속적이라 애플리케이션을 OS마다 그에 맞게 변경해 주어야 하지만, **Java 애플리케이션은 JVM이랑 맞붙어 있기 때문에 OS와 하드웨어에 독립적**이다.

● 다만 **JVM은 OS에 종속적**이라 해당 OS에서 실행가능한 JVM이 필요하다.
● 또한 **JVM은 한 번 더 거치기 때문**에 그리고 하**드웨어에 맞게 완전히 컴파일 된 상태가 아니고 실행 시에 해석(interpret)**되기 때문에 **속도가 느리다**는 단점을 가지고 있다.
● 하지만 **바이트코드(컴파일된 자바코드)를 하드웨어의 기계어로 바로 변환해주는 JIT컴파일러**와 향상된 최적화 기술이 적용되어서 이런 속도적인 단점이 많이 보안되었다.

4. Java로 프로그램 작성하기

● JDK (Java Developement Kit) 을 설치하면 JVM과 자바클래스 라이브러리 (Java API) 외에 자바를 개발하는 데 필요한 프로그램들이 설치된다.
● Java API문서: http://java.sun.com



● 자바에서 **모든 코드는 반드시 클래스 안**에 존재해야 하며, 서로 **관련된 코드들은 그룹으로 나누어 별도의 클래스**로 구성하게 되고, 이 클래스들이 모여 하나의 Java Application을 이룬다.

● Java 애플리케이션은 **main 메서드의 호출로 시작해서 main 메서드의 첫 문장부터 마지막 문장까지 수행을 마치면 종료**된다. **main 메서드를 포함한 클래스가 반드시 하나**는 있어야 하는데, Java Application의 **시작점**이므로 main메서드 없이는 Java 애플리케이션은 실행될 수 없기 때문이다.

● 하나의 **소스파일(.java)**에 하나의 클래스를 정의하는 것이 보통이지만 둘 이상의 클래스를 정의하는 것도 가능하다. **소스파일의 이름은 public class의 이름과 일치**해야 한다. 만약 소스파일 내에 **public class가 없다면**, 소스파일의 이름은 소스파일 내의 어떤 **클래스 이름**으로 해도 괜찮다. **하나의 소스 파일에 둔 이상의 public class가 존재하면 안 된다**. 각 클래스를 별도의 소스파일에 나눠서 저장하든가 아니면 둘 중의 한 클래스에 public을 붙이지 않아야 한다.

● **클래스파일(.class)**은 클래스마다 하나씩 만들어지므로 소스파일을 컴파일하면 클래스 갯수별로 클래스파일이 생성된다.

● 자주 발생하는 에러
1) cannot find symbol / cannot resolve symbol: 선언되지 않은 변수나 메서드를 사용, 혹은 변수 또는 메서드의 이름을 잘못 사용.
2) Exception in thread "main" java.lang.NoSuchMethodError: main: main메서드가 존재하지 않거나 선언부에 오타가 존재하는 경우.
3) Exception in thread "main" java.lang.NoSuchMethodError: main: Hello: 클래스 Hello의 철자, 대소문자 확인, 클래스파일이 생성되었는지 확인. 클래스패스(classpath)의 설정이 제대로 됐는지 확인.
4) illegal start of expression: 문법적 오류, 괄호를 닫지 않거나 수식이나 if문, for문에 문법적 오류가 있을 때, public이나 static같은 키워드를 잘못 사용한 경우.
5) class, interface, or enum expected: 괄호 갯수가 일치하지 않는 경우.

● 자바프로그램의 실행과정
1) 프로그램의 실행에 필요한 클래스(.class파일)을 로드
2) 클래스파일 검사(파일 형식, 악성코드 체크)
3) 지정된 클래스에서 main(String[] args) 호출
main메서드의 첫 줄부터 코드가 실행되기 시작해 마지막 코드까지 모두 실행되면 프로그램이 종료되고, 프로그램에서 사용했던 자원들은 모두 반환된다.

● 주석(Comment)

- 왜 코드를 이렇게 작성했는지
- 프로그램의 작성자, 작성일시, 버전과 그에 따든 변경이력 등의 정보
- // 한 줄 주석
- /* 범위 주석 */

컴파일러는 주석을 무시하고 건너뛰기 때문에 컴파일되지 않는다.

Resource: 자바의 정석