**학습점검목록**

**1일차**

1. 소프트웨어 유형 별 특징을 설명할 수 있는가? []

소프트웨어의 유형은 크게 시스템S/W 와 애플리케이션 S/W로 구분된다 시스템 소프트웨어는 os, driver, imbedded, IoT와 같이 H/W를 제어하는 소프트웨어이다. 애플리케이션 소프트웨어는 또 2가지로 나뉘는데 설치만 하면 인터넷 연결없이 단독적으로 실행할 수 있는 소프트웨어인 standalone 프로그램과 client/server가 있다. Client/server 프로그램은 로컬에 클라이언트를 설치하고 서버와 통신하는 클라이언트 서버가 있고, 웹브라우저를 통해 서버와 통신하여 네트웍을 하는 웹 애플케이션 소프트웨어가 있다.

2. 웹 애플리케이션 아키텍처와 구동 원리를 간단히 설명할 수 있는가? []

먼저 사용자가 웹브라우저에서 요청하면 웹 서버에서 프론트엔드에 해당하는 HTML, CSS, Javacript는 직접 바로 다운로드하고 java파일의 경우 실행해서 결과만 받아온다. 하지만 이 과정 중 웹서버가 java파일을 직접 관리하기가 어려워 서블릿 컨테이너에 이 역할을 위임하고 서블릿 컨테이너는 java파일을 더 효율적으로 관리하기 위한 프레임워크인 스프링 프레임워크를 통해 실행하고 결과를 받아 리턴하게 된다.

**2일차**

1. CPU와 기계어(instruction set)의 관계를 설명할 수 있는가? []

CPU는 기본적으로 인간이 하는 언어를 이해하지 못한다. 0과 1로 이루어진 전기 신호인 기계어만 이해할 수 있는데 이 기계어는 CPU마다 각각 다르다.

2. 동일한 CPU에 OS가 다를 때 프로그램이 호환되지 않는 이유를 설명할 수 있는가? []

운영체제마다 프로그램 실행 포맷이 다르기 때문이다. 윈도우는 PE 맥은 MACH-O 리눅스는 ELF라는 포맷이 존재한다.

3. 동일한 OS에 CPU 아키텍처가 다를 때 프로그램이 호환되지 않는 이유를 설명할 수 있는가? []

실행 포맷이 같더라도 CPU가 이해할 수 있는 기계어의 유형이 다르기 때문에 해당 CPU 기계어 기반으로 만든 프로그램이 아니라면 호환되지 않는다.

4. 어셈블리와 C 언어를 사용해 애플리케이션 만드는 과정을 설명할 수 있는가? []

개발자가 소스를 작성하여 실행하면 C 컴파일러를 통해 어셈블리어로 컴파일 되고 어셈블리어는 다시 어셈블리어 컴파일러를 통해 기계어로 컴파일되어 애플리케이션이 실행된다.

5. Low-level 언어와 High-level 언어의 의미를 설명할 수 있는가? []

Low-level 언어는 기계어와 흡사한 언어로 대표적으로 어셈블리어가 있으며 이 어셈블리어는 os에 종속적이고 프로세서에 대한 사전지식을 많이 요구하며 라인당 1라인에 명령어가 대응되어 코드가 길다.

High-level 언어는 인간에게 친숙한 언어이다. 즉 인간이 이해하기 쉬운 문법으로 구성되어 있다.

6. 자바 애플리케이션을 만드는 방법을 설명할 수 있는가? []

개발자가 개발도구로 자바 소스를 작성하면 자바 컴파일러를 통해 바이트코드라는 중간 형태의 명령이 생성된다. 이 바이트코드는 기계어와 유사하지만 진짜 기계어는 아니기에 cpu에 전달되어 실행되기 위해서는 JVM, 다른 말로 자바 플레이어를 통해 이를 해석하고 실행한다.

7. REPL과 jShell 을 설명할 수 있는가? []

REPL은 Read – Evaluation-Print-Loop의 약자로 개발자가 만든 명령을 테스트 하는 용도의 과정이다. JSHELL은 자바에서 간단히 사용되는 REPL 중 하나이다.

8. JVM, JRE, JDK, JavaSE, JavaEE, JavaME를 설명할 수 있는가? []

먼저 용도에 따라 구분이 Java SE, Java EE, Java ME로 나뉜다.

Java Se는 Standard Edition으로 개인 서버 애플리케이션에 제작에 필요한 도구를 제공하고, EE는 Enterprise Edition으로 기업 애플리케이션 서버, ME는 Micro Edition은 임베디드 애플리케이션 개발에 필요한 도구를 제공한다.

JVM은 바이트코드를 해석하고 실행하는 프로그램이며 JRE는 JVM을 포함한 실행관련 파일을 합친 것이다. JDK는 여기에 컴파일러와 디버거 프로파일러 등 개발도구를 모두 합친 개발도구를 의미한다.

9. JDK를 설치하고 설정할 수 있는가? []

첫번째, 설치설치

두번째,환경변수 설정

윈도우에서는 고급설정가서 환경변수 -> PATH에 jdk/bin까지 등록 -> JAVA\_HOME 새로만들기 한 후 폴더 JDK-버전까지 등록

맥은 해당 nano 편집기에 들어가서(nano .zshrc) export JAVA\_HOME=$(jdk위치) 등록

10. JAVA\_HOME, PATH 환경 변수를 설정하는 이유를 알고 있는가? []

자바 애플리케이션이 실행할 때 JDK가 있는 폴더를 찾기도 하는데 이때 JDK를 더 찾기 쉽도록 경로를 지정해주는 것이다.

11. VSCode를 설치하고 자바 애플리케이션 작성에 필요한 환경을 설정할 수 있는가? []

Codelens, whitespace, inlay hint none, side 미니맵 제거

12. [github.com](http://github.com/) 에 저장소를 생성할 수 있는가? []

Repository 생성, public에 Read.me 파일 체크해서 생성

3일차

1. 버전관리시스템의 유형 별 특징을 설명할 수 있는가?

버전관리 시스템은 소스관리와 변경관리를 처리하는 프로그램, 로컬 버전관리시스템은 local에서 생성/변경/삭제의 사항들을 타 서버가 아닌 로컬에서 각 버전으로 나누어서 직접 관리, 중앙 집중식 버전관리시스템은 서버에 소스파일과 변경사항을 기록한다. 소스파일으 가져오는 것이 checkout, 다시 기록하는 것을 checkin이라고 한다. 공유가 가능하고 서버 하나만 관리하면 되지만 서버가 날아가면 변경기록을 복구할 수가 없다. 분산버전관리 시스템은 서버에 관리하는 것은 동일하지만 해당 기록을 각 로컬에도 분산하여 관리하는 것이 특징

2. git client를 사용하여 git server로부터 저장소를 복제할 수 있는가?

-> $ git clone 주소

3. 깃 저장소의 .git 폴더와 working directory의 관계를 설명할 수 있는가?

.git 폴더가 변경기록을 담은 폴더이고 나머지는 working directory로 사용자의 작업 영역으로서 파일을 생성하거나 수정하는 등 작업중인 소스코드를 담고 있다.

4. 깃 저장소 working directory에 있는 파일의 변경 상태를 알아내는 명령을 실행할 수 있는가?

-> $ git status

5. 깃 저장소의 working directory에 있는 파일의 변경 상태 세가지를 설명할 수 있는가?

Untracked: 한번도 커밋되지 않은 새로운 파일

Unmodified: 커밋된 후 수정되지 않은 상태의 파일

Modified: 커밋된 후 수정된 상태의 파일

6. 깃 저장소의 staging area에 대해 설명할 수 있는가?

Git add 명령어로 커밋되기 전 인덱스 명단에 추가된 상태,

7. 깃 명령 add, commit, push, pull에 대해 설명할 수 있는가?

Git add: 백업할 대상을 백업 명단(인덱스 영역)에 추가하는 명령어

Git commit: 인덱스 영역의 내용을 바탕으로 새로운 버전의 파일을 생성하는 명령어, 백업의 개념으로도 본다. 즉 특정 시점에 존재하는 파일들에 대한 정보를 스냅샷을 찍어서 하나의 버전으로 저장해두겠다는 의미

Git push: 마지막으로 커밋된 내용을 그대로 서버 저장소로 이동시키는 명령어

8. zsh에서 깃 저장소에 커밋된 기록을 조회할 수 있는가?

조회: Git log or git log –oneline

9. 깃 저장소에서 다른 커밋 버전의 파일들을 working directory에 꺼낼 수 있는가?

다른 버전 파일 꺼내기

Git checkout 커밋id or main(마지막 커밋내용)

10. 로컬 깃 사용자의 이름과 이메일 정보를 조회하고 설정할 수 있는가?

Git config –list

등록은 뒤에 사용자이름 또는 이메일 값을 입력해주면 됨

11. github.com에서 원격 깃 저장소에 접근할 때 사용할 토큰을 발급할 수 있는가?

토큰 발급 방법 setting -> developer setting -> personal access tokens -> classic

4일차

1. 소프트웨어의 형상의 개념과 관리의 개념을 설명할 수 있는가?

소프트웨어에서 형상은 소프트웨어에서 관리되어야 항목을 의미한다. 관리는 이러한 항목에 대해 체계적으로 추적하고 통제하는 것.

2. 소프트웨어 형상관리와 버전관리시스템의 관계를 설명할 수 있는가?

형상관리의 항목 중 변경관리에 해당하는 tracking과 controlling을 담당하는 시스템이 버전관리 시스템이다.

3. git push를 수행할 때 충돌이 발생할 경우 그 이유를 설명할 수 있는가?

Git 서버에서는 파일을 작업한 후 한 로컬에서 커밋하고 푸시하면 새 버전으로 업데이트되는데 기존 동일한 작업 파일을 수정하고 푸시하면 git 서버에서 자체적으로 reject한다.

4. git push 과정에서 발생한 충돌을 처리할 수 있는가?

새 버전으로 업데이트 된 레파지토리 파일을 git pull 한 후 merge하는 명령어를 통해 병합, 병합된 내용을 편집 후 다시 커밋하면 새버전으로 다시 push가 가능해진다.

5. .gitignore 파일의 용도를 설명할 수 있는가?

add/commit하고싶지 않은 특정 파일을 지정하여 백업 명단에서 제외하는 파일

6. 텍스트파일과 바이너리 파일을 구분할 수 있는가?

텍스트 편집기로 편집이 가능한 파일. 플레인 텍스트 파일은 글자 데이터만 존재, MD, html과 같은 파일은 메타데이터, markup,tag 같은 콘텐트를 제어하는 명령을 포함한다. 바이너리 파일은 이진파일이라고도 하며 0과 1로 이루어진 파일로서 엑셀, 프리미어 등 전용 편집기로만 편집이 가능한 파일

7. 바이트코드의 컴파일 버전과 JVM 버전 사이의 실행관계를 설명할 수 있는가?

상위 JVM은 하위 버전의 컴파일러로 만들어진 바이트코드를 실행할 수 있지만 하위 JVM은 상위 버전의 컴파일러로 만들어진 바이트코드를 실행할 수 없다.

8. 인터프리터 방식, 컴파일 방식, 하이브리드 방식을 예를 통해 비교 설명할 수 있는가?

인터프리터 방식은 파이썬처럼 인터프리터를 사용해 소스파일을 직접 읽어서 바로 실행하는 방식, 실행할때마다 문법검사와 해석을 하기 때문에 컴파일방식에 비해 느리다. 또 코드가 실행되기 전에는 오류를 알 수 없다. 소스코드 자체가 실행파일이기 때문에 자산으로서 보호하기 어렵다.

컴파일 방식은 c언어와 같이 c컴파일러를 통해 소스코드를 변환한 후 결과파일이 기계어라 바로 실행이 가능한 방식. 컴파일 할때 문법검사를 실행하여 비교적 빠르다. 또 소스코드를 노출하지 않아 보호가 가능하다.

하이브리드 방식은 자바,C#과 같이 컴파일을 하면 중간 형태의 언어(bytecode,CIL)로 변환되고 해당 언어 인터프리터를 통해 실행하는 방식이다. 컴파일 방식이 이점을 그대로 취하고 특정 os에 종속되지 않아 호환성이 좋다. 다만 진짜 기계어가 아니기 때문에 컴파일 방식에 비해 상대적으로 느리다

9.Jit 컴파일 방식과 AOT 컴파일의 특징을 설명할 수 있는가?

Jit 컴파일 방식은 실행시점에 자주 실행하는 코드를 진짜 기계어로 바꿔놓고 그 기계어를 실행하여 실행속도를 높이는 방식, 이때 캐시에 보관된 기계어를 보관하는 곳을 cashe라고 하며 실행 완료시 제거된다.

AoT 컴파일 방식은 설치 과정에 미리 컴파일을 하여 cashe에 보관하고 실행 시 기계어를 실행하는 방식이다. 실행속도는 빠르지만 설치 시간이 길다.

10. 빌드가 무엇인지 설명할 수 있는가?

컴파일 – 테스트 – 보고서/문서 작성 – 패키징(jar,war) – 배포 과정, 즉 소프트웨어 제작 전체 과정을 빌드라고 한다.

11. 빌드 도구와 빌드 스크립트 파일의 관계를 설명할 수 있는가?

빌드 도구는 빌드를 수행하는 것을 자동화시켜주는 도구이고 빌드 스크립트 파일은 빌드에 사용한 정보, 빌드 순서, 사용할 도구를 지정하는 파일이다.

‘

12. Ant, Maven, Gradle 빌드 도구의 빌드 스크립트 파일을 구분할 수 있는가?

아파치 ant는 build.xml, maven은 pom.xml, Gradle은 build.gradle이다.

13. Ant 빌드 도구 이후에 Maven과 Gradle이 등장하게 된 이유를 설명할 수 있는가?

Ant는 개발자가 원하는 대로 빌드할 수 있도록 유연성이 높은 도구지만 xml, remote repository를 가져올 수 없다는 단점이 존재한다. Maven은 의존라이브러리 기능을 포함함으로서 ant의 단점을 보완하였고 Gradle의 경우 ant, Maven의 단점을 보완해 자바 기반의 빌드도구로서 groovy 로직을 이용해 보다 정밀하게 빌드과정을 제어할 수 있도록 하였다.

14. 소프트웨어 세계에서 라이브러리 파일이 무엇을 의미하는지 설명할 수 있는가?

사람의 라이브러리는 책, 소프트웨어의 라이브러리는 컴파일된 파일, 또는 날것의 파일을 담고 있다. 라이브러리는 다른 개발자가 만든 코드가 들어있는 파일을 의미한다.