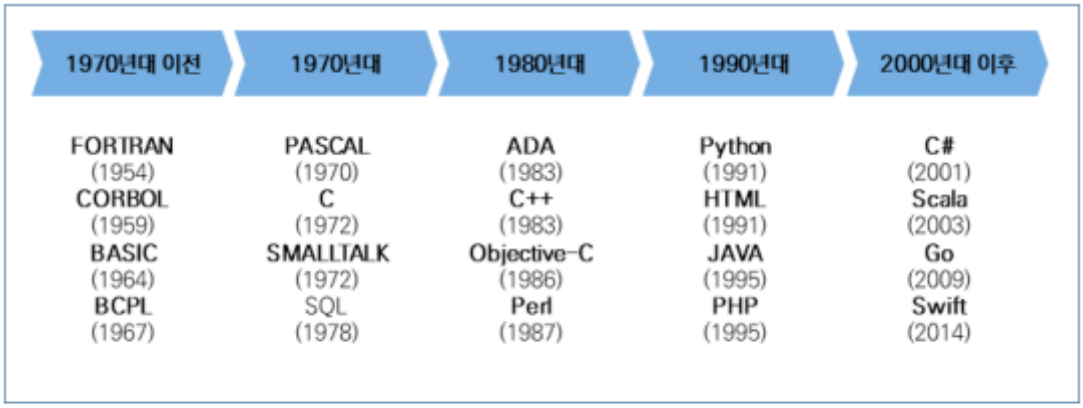
프로그래밍 언어 응용

1. 언어 특성 활용하기

1 - 1 언어별 특성 파악

① 프로그래밍 언어의 발전 과정과 유형 분류

1. 프로그래밍 언어의 발전 과정



출처: 집필진 제작(2021)
[그림 1-1] 프로그래밍 언어의 발전 과정

2. 프로그래밍 언어의 발전 과정

(1) 개발 편의성 측면에 따른 분류

(가) 저급언어(Low-Level Language):

컴퓨터가 직접 이해할 수 있는 언어로 실행속도는 빠르나 기계마다 기계어가 상이하여 호환성이 없고 유지관리가 어렵다.

(나) 고급언어(High-Level Language):

개발자가 이해할 수 있도록 소스코드를 작성할 수 있는 언어로, 실행을 위해서는 번역 과정이 필요하다.

(2) 실행 및 구현 방식에 따른 분류

(가) 명령형 언어(Imperative Language):

컴퓨터가 동작해야 할 알고리즘을 통해 프로그래밍의 상태를 변경시키는 구문에 중점을 둔 방식으로 FORTRAN, C 등이 속한다.

(나) 함수형 언어(Functional Language):

함수의 응용을 강조하면서 자료의 처리는 수학적인 함수의 연산으로 취급하고, 상태와 가변 데이터는 멀리하는 방식으로 LISP, Scala 등이 속한다.

(다) 논리형 언어(Logic Language):

논리 문장을 이용하여 프로그램을 표현하고 조건 이 만족되면 연관된 규칙이 실행되는 방식으로 PROLOG 등이 속한다

(라) 객체지향언어(Object-Oriented Language):

객체 간의 메시지 통신을 이용하여 동작하는 방식으로 JAVA, C++ 등이 속한다.

(3) 빌드(Build) 방식에 따른 분류

(가) 컴파일 언어(Compile Language):

소스코드가 기계어 실행 파일로 빌드되는 방식이다. C, C++ 등이 속하며 실행속도가 높은 특징이 있다.

(나) 인터프리터 언어(Interpreter Language):

소스코드를 한 줄씩 번역하여 실행하는 방식이다. Python 등이 속하고 실시간 실행 및 분석이 가능한 특징이 있다.

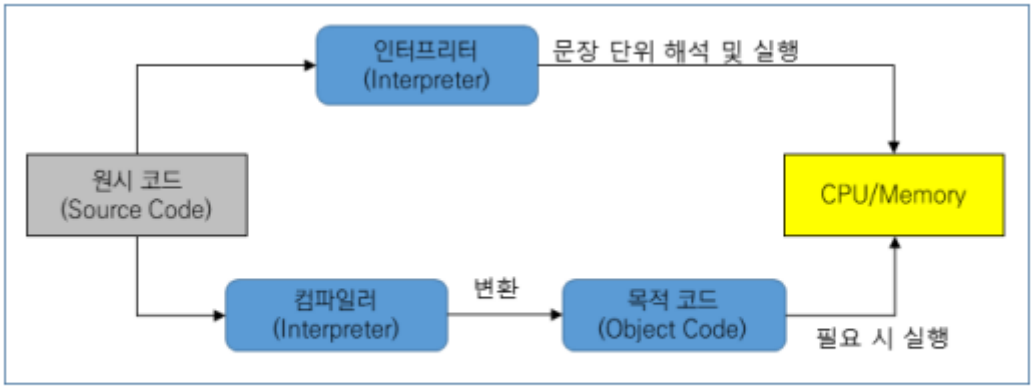
(다) 바이트 코드 언어(Byte Code Language):

컴파일을 통해 가상머신이 번역할 수 있는 Byte Code로 변환되며, 가상머신은 다시 Native OS가 이해할 수 있는 기계어로 번역하는 방식이다. JAVA, Scala 등이 속한다.

② 컴파일러(Compiler)와 인터프리터(Interpreter)

1. 컴파일러와 인터프리터 동작 방식

컴파일러의 경우 소스코드를 목적 코드로 변환하여 실행하는 방식이며,
인터프리터의 경우 문장 단위로 읽어 들여 해석을 하여 실행한다.



출처: 집필진 제작(2021)
[그림 1-2] 컴파일러와 인터프리터 동작 방식

2. 컴파일러와 인터프리터 비교

컴파일 방식의 경우 실행속도가 빠르고 보안적인 측면에서는 유리한 장점이 있으나
매번 빌드 작업을 거쳐야 하는 불편함이 존재하며, 인터프리터 방식의 경우 즉시 수정 및 실행이 가능한 장점은 있으나 처리속도가 느린 단점이 있다.

〈표 1-1〉 컴파일러 및 인터프리터 비교

구분	컴파일러	인터프리터
개발 편의성	코드를 수정하고 실행이 필요한 경우 재컴파일 필요	코드 수정 후 즉시 실행 가능
번역 단위	전체 소스코드	문장 단위
실행 파일 및 속도	실행 파일 생성 처리속도 빠름	실행 파일 미생성 처리속도 느림
메모리 할당	실행 파일 생성 시 사용	할당하지 않음
오류 확인 및 처리	전체 코드에 대한 컴파일 수행 시 발생한 오류 확인 가능	프로그램 실행 후 오류가 발생한 문장 이후의 코드는 실행하지 않음
파일 용량 및 보안	실행 파일 전체를 처리해야 하므로 용량이 크며, 원시 코드의 유출 가능성이 상대적으로 낮음	원시 코드만 처리하면 되므로 용량이 상대적으로 작고, 원시 코드의 유출 가능성이 높음
주요 언어	C, C++, JAVA	Python, Javascript, Ruby

1 - 2 애플리케이션 구현 및 최적화

① 코드 인스펙션(Code Inspection)

프로그램 소스코드를 실행하지 않고 코드상에서의 잠재적인 오류와 표준 미 준수 등의 결함을 사전에 식별하여 개선하는 공식적인 리뷰(Review) 기법

1. 인스펙션 수행 절차

〈표 1-12〉 인스펙션 수행 절차 및 수행 내용

단계	수행 내용
계획(Planning)	인스펙션 대상 산출물 선정 및 대상자를 구성. 인스펙션 대상 산출물을 사전에 배포하고 날짜와 시간 및 장소를 공지한다.
개관(Overview)	참여자를 대상으로 산출물에 대한 이해도를 높여 인스펙션의 효과성을 향상시킬 수 있으며 생략 가능하다.
준비(Preparation)	구성원이 개별적으로 산출물에 대해 숙지하고 체크리스트를 활용하여 결함 부분에 대해서 기록한다.
검토회의(Meeting)	산출물을 함께 검토하며 준비단계나 검토회의에서 식별된 결함 부분에 대해 집중한다.
재작업(Rework)	검토회의에서 발견된 결함에 대해 수정한다.
추적(Follow-up)	결함이 정상적으로 수정되었는지 최종 확인하고 인스펙션을 종료한다.

2. 코드리뷰 기법 비교

〈표 1-13〉 코드리뷰 기법 비교

단계	수행 내용
동료검토(Peer Review)	별도의 절차 없이 비공식으로 계획 없이 임의적으로 실시되며, 개발자가 동료와 코드 및 산출물의 결함을 식별하는 기법
워크스루(Walk Through)	개발자가 리뷰의 주제와 시간을 정해서 발표를 하고 동료들로부터 의견이나 아이디어를 듣는 시간을 가질 수 있으며, 사례에 대한 정보 공유나 아이디어 수집을 위해서 사용될 수 있다.
Inspection	역할과 절차, 체크리스트를 기준으로 결함을 식별하는 공식적인 리뷰 기법

② 리팩토링(Refactoring)

SW의 원래 기능은 유지하면서 소스코드의 내부 구조를 수정 및 보완하여
가독성, 성능 향상 및 로직을 개선하는 기법이다.

1. 주요 리팩토링 대상 및 방법

리팩토링 대상인 나쁜 냄새(Bad Smell)는 오류는 아니지만 문제를 유발할 수 있거나 설계 결함을 가지고 있는 부분으로 잠재적인 오류 예방과 성능 향상, 불필요한 코드 제거, 유지보수 비용 절감, 재사용성 향상 등을 위해 개선한다.

〈표 1-14〉 리팩토링 대상 및 적용 방법

리팩토링 대상	리팩토링 방법
중복된 코드(Duplicated Code)	중복을 제거해야 한다.
긴 메소드(Long Method)	메소드를 적정 수준의 크기로 나누어야 한다.
긴 파라미터 리스트(Long Parameter List)	파라미터 개수를 줄여야 한다.
게으른 클래스(Lazy Class)	자식 클래스와 부모 클래스의 차이가 없으면 합친다.
주석(Comment)	주석이 없어도 코드를 이해할 수 있도록 소스코드를 변경한다.
메시지 체인(Message Chain)	특정 객체를 얻기 위한 다수 객체는 간소화한다.
미들 맨(Middle Man)	다른 클래스로 위임하는 역할만 담당하는 클래스 제거를 검토한다.
불완전 라이브러리(Incomplete Library)	불완전 시 필요 부분 추가 구성한다.
스위치 명령문(Switch Statements)	지나치게 많은 case를 사용하는 Switch 문장은 코드 중복의 신호이다.

③ OWASP TOP 10

OWASP(The Open Web Application Security Project)는 웹 애플리케이션에 대한 보안 프로젝트이다. OWASP TOP 10은 웹 애플리케이션 취약점 중에서 빈도가 많이 발생하고 보안상 영향을 크게 줄 수 있는 10가지를 선정하여 발표한다

〈표 1-15〉 OWASP TOP 10 항목 및 내용(2017년 발표 기준)

항목	상세 내용
Injection(인젝션)	웹 애플리케이션에 비정상적인 명령어나 Query 등을 보내 공격자가 시스템에 불법적으로 접근할 수 있는 취약점
Broken Authentication(취약한 인증)	잘못 구현된 인증이나 Session 관리 기능으로 인해 일시적 혹은 영구적으로 공격자가 다른 사용자의 권한을 획득할 수 있는 취약점
Sensitive Data Exposure(민감한 데이터 노출)	개인 식별 정보나 신용 정보 등의 민감 데이터 저장 및 전송 시 노출 취약점
XML External Entities(XXE)(XML 외부 개체)	XML 문서에서 External Entity를 이용하여 공격자가 의도하는 외부 URL을 실행하여 서버의 로컬파일 정보를 출력하거나 서비스 거부 공격을 수행할 수 있는 취약점
Broken Access control (취약한 접근 통제)	다른 사용자의 계정 접근, 중요 데이터 열람 및 수정, 액세스 권한 변경 등의 악의적인 행위가 가능한 취약점
Security misconfigurations (잘못된 보안 구성)	안전하지 않은 구성, 잘못된 구성으로 HTTP Header나 민감 정보가 포함된 에러 메시지 등으로 인한 취약점
Cross Site Scripting (XSS)(크로스 사이트 스크립팅)	웹 페이지에 악성 스크립트를 삽입할 수 있는 취약점으로 사용자의 정보(쿠키, 세션 등)를 탈취하거나 악의적인 사이트로 이동할 수 있는 취약점
Insecure Deserialization(안전하지 않은 역직렬화)	데이터를 역직렬화하는 과정에서 원격코드 실행이나 권한 상승 등이 가능한 취약점
Using Components with Known Vulnerabilities(알려진 취약점이 있는 구성요소 사용)	취약한 컴포넌트가 악용되는 경우 서버가 장악되거나 심각한 데이터 손실을 발생할 수 있는 취약점
Insufficient logging and monitoring (불충분한 로깅 및 모니터링)	충분하지 않은 로깅과 모니터링을 통해 시스템을 추가로 공격하고 데이터를 변조하거나 추출, 파괴 가능한 취약점

출처: OWASP Top Ten. https://owasp.org/www-project-top-ten/에서 2021. 08. 16. 검색.

④ 정적 분석과 PMD(Programming Mistake Detector)

1. 정적 분석(Static Analysis)

동적 분석(Dynamic Analysis)과 상대되는 개념으로 소프트웨어를 실행하지 않고 코드 레벨에서 분석하는 방법으로 소스코드의 실행 없이 코드의 의미를 분석해 결함을 찾아내는 코드 분석 기법

*** 정적분석도구로 PMD 프로그램이있음**

2. PMD(Programming Mistake Detector)

응용프로그램 코드의 문제를 사전에 확인할 수 있는 정적 분석을 위한 오픈 소스코드 분석 도구이며, 기본 제공 규칙 세트가 포함되어 있고 사용자 지정 규칙을 작성할 수 있다. 단독 형태로도 사용할 수 있고 이클립스 등과 같은 IDE에 플러그인 형 태로 배포되어 누구나 사용할 수 있다.

*** 자바에서 노란 점선 같은 의미**

2 - 1 라이브러리 선정

① 라이브러리(Library)

라이브러리는 프로그램을 효율적으로 개발할 수 있도록 필요한 기능이 구현된 프로그램을 모은 집합체이며, 일반적으로 설치 파일과 도움말, 예시 코드 등을 제공한다.

1. 라이브러리 개념과 목적

코드의 재사용 및 부품화, 기술 유출 방지를 위하여 하나 이상의 Subroutine 혹은 Function들의 집합. 일반적으로 컴파일되어 실행이 가능한 Object Code 형태로 제공된다.

② 라이브러리의 유형

1. 별도 설치 여부에 따른 유형

(1) 표준 라이브러리(Standard Library)

프로그래밍 언어와 함께 제공되는 라이브러리로 표준 라이브러리를 이용하면 별도의 파일 설치 없이 다양한 기능을 이용할 수 있다. 각 프로그래밍 언어의 표준 라이브러리는 기본 기능 이외에 날짜와 시간 등의 다양한 추가 기능을 이용할 수 있는 API를 제공한다.

(2) 외부 라이브러리(3rd Party Library)

별도의 파일을 설치하여 사용할 수 있는 라이브러리를 의미한다. 누구나 개발하여 사용하거나 공유할 수 있고, 관련 커뮤니티나 인터넷 검색 등을 이용하여 공유된 라이브러리를 사용할 수 있다.

2. 코드 결합 방식에 따른 유형

(1) 정적 라이브러리(Static Library)

프로그램을 빌드할 때 라이브러리가 제공하는 코드를 실행 파일에 넣는 방식으로 컴파일러가 원시 소스를 컴파일할 때 참조되는 모듈이다. 동작 코드가 실행 바이너리에 포함되어 별도의 추가 작업 없이 독립적으로 라이브러리 함수 사용이 가능하다

(2) 동적 라이브러리(Dynamic Library)

공통적으로 필요한 기능들을 프로그램과 분리하여 필요시에만 호출하여 사용할 수 있도록 만든 라이브러리를 의미한다. 필요할 때만 해당 라이브러리를 메모리로 불러들일 수 있어 실행 파일의 크기를 줄일 수 있고, 사용이 끝나면 메모리에 서 제거되어 효율적인 메모리 사용이 가능하다. 윈도에서는 주로 .dll 확장자, 리눅스에서는 주로 .so 확장자를 가진다.

③ 라이브러리, 프레임워크, 아키텍처, 플랫폼 비교

라이브러리는 프로그램을 효율적으로 개발할 수 있도록 필요한 기능이 구현된 프로그램을 모은 집합체이며, 일반적으로 설치 파일과 도움말, 예시 코드 등을 제공한다.

〈표 2-1〉 라이브러리와 유사 용어 설명

항목	설명	예시
Library	코드 재사용 및 부품화를 위하여 필요한 기능에서 호출하여 사용할 수 있도록 제공되는 모듈의 집합	jQuery, DLL, Class, Jar 등
Framework	응용프로그램 표준 구조를 구현하기 위한 클래스와 라이브러리 모임	Spring, Django 등
Archtecture	여러 가지 컴퓨터 구성요소들에 대한 전반적인 기계적 구조와 이를 설계하는 방법	모노리틱 아키텍처, MSA 등
Platform	여러 가지 기능을 제공해주는 프로그램 실행이 가능한 공통 실행환경으로 플랫폼 위에 다른 플랫폼이 존재할 수 있음	Window, Linux, JVM 등

2 - 2 라이브러리 구성 및 적용

① 모듈과 패키지

라이브러리는 모듈과 패키지를 모두 포함하며, 모듈의 경우 개별 파일을 지칭하고 패키지의 경우 여러 개의 파일을 모아 놓은 폴더로 생각할 수 있다

〈표 2-9〉 모듈과 패키지 비교

구분	설명	사용법
Module	한 개의 파일에서 기능을 제공한다.	import (모듈명)
Package	여러 개의 모듈을 한 개의 폴더에 묶어서 기능을 제공한다. 패키지명과 모듈명을 import하여 불러올 수 있다.	import (패키지명).(모듈명)

② PyPI(Python Package Index)

Python Package Index는 파이썬 패키지들이 모여 있는 저장소를 의미하고, Python 개발자라면 누구에게나 오픈되어 있다. PyPI에서 권하는 Python라이브러리 설치 툴이 pip이며,
파이썬 3.4 이후의 버전에는 기본으로 포함되어 있다

③ Maven

아파치 메이븐(Apache Maven)은 자바용 프로젝트 관리 도구로 아파치 앤트(Ant)의 대안으로 만들어졌으며 아파치 라이선스로 배포되는 오픈소스 소프트웨어이다. Maven은 필요한 라이브러리를 특정 문서(pom.xml)에 정의해 놓으면 내가 사용할 라이브러리뿐만 아니라 해당 라이브러리가 작동하는 데에 필요한 다른 라이브러리들까지 관리하여 네트워크를 통해서 자동으로 다운받도록 해준다

3. 빌드 도구 비교

〈표 2-11〉 빌드 도구 비교

구분	특징
Ant	- XML 기반의 빌드 스크립트 작성, 자유롭게 빌드 단위 지정 - 간단하고 사용하기 쉬우나 큰 프로젝트의 경우 스크립트 관리나 빌드 과정이 복잡 - 생명주기(Lifecycle)을 갖지 않아 각각의 결과물에 대한 의존관계 등을 정의
Maven	- XML 기반으로 작성되며 생명주기와 POM(Project Object Model) 개념 도입 - pom.xml 파일을 이용해 dependency를 추가하고 삭제하여 라이브러리를 관리
Gradle	- Ant와 Maven의 장점을 모아 작성되었으며 의존성 관리를 위한 다양한 방법 제공 - 빌드 스크립트를 XML이 아닌 JVM에서 실행되는 스크립트 언어인 그루비(Groovy) 사용