Programming Language

컴퓨터개론

(Introduction to Computer Systems)

GEN1030

프로그래밍 언어 개요

프로그래밍 언어

- 사람과 컴퓨터가 의사교환을 하기 위한 언어
  - 사람이 컴퓨터에게 지시할 명령어를 기술

- Low level language
- High level language

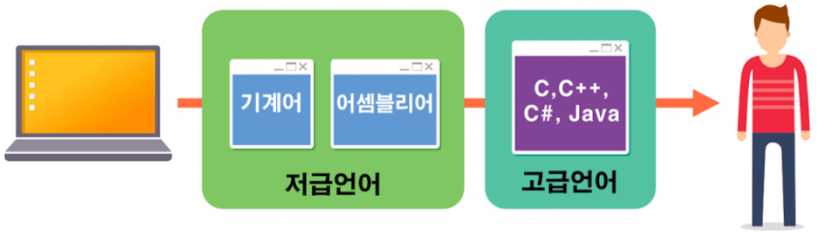
Low Level Language

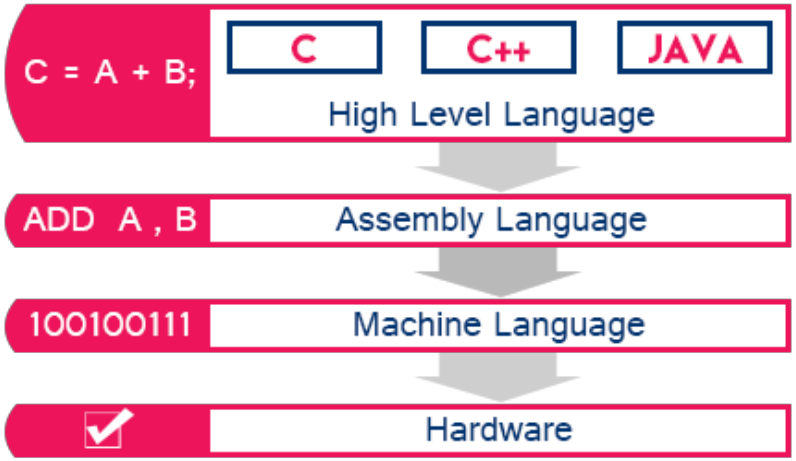
- 주기억장치, 레지스터 등의 하드웨어를 직접 통제 가능
- 기계어(Machine language)
  - 0과 1로 표현
  - 컴퓨터가 직접 이해 가능한 언어
  - Operation code와 Operand로 구성
- 어셈블리어(Assembly language)
  - 기계어를 기호화(symbolize) 함 –연산코드와 피연산자를 기호형태로 일대일 대응
  - 연산코드를 기호화 한 것은 mnemonic이라고 부름 (ex. LDA, ADD)

순서	기계어	어셈블리어	의미
명령어1	0101000000000100	LDA A	메모리 A의 내용을 누산 레지스터(AC)에 저장
명령어2	0111000000000110	ADD B	메모리 B의 내용과 누산 레지스터(AC)의 값을 더하여 누산 레지스터(AC)에 다시 저장
명령어3	0100000000000111	STA C	누산 레지스터(AC)의 값을 메모리 C에 저장
명령어4	0011000000000000	HLT	프로그램 종료

High Level Language

- 인간이 이해하기 (비교적) 쉬운 언어
  - 일상적인 언어나 기호 등을 사용
  - 종류
    - FORTRAN, COBOL
    - C, C++, Java, Python 등





세대별 분류

세대	시기	기능
1세대	1945년	컴퓨터가 이해하는 유일한 언어인 기계어만을 이용한 세대이며 현재에도 기계어는 이용됨
2세대	1950년 중반	어셈블리어와 어셈블러가 개발되어 프로그램 개발의 생산성이 높아진 세대이나 시스템마다 어셈블리어는 다르므로 시스템 호환 문제가 계속 남아 있었던 세대임
3세대	1960년 초반	포트란, 알골, 베이직, 파스칼 같은 고급 언어와 컴파일러가 개발되었고 시스템에 독립적인 프로그램을 개발하여 프로그램 개발의 생산성이 매우 높아진 세대임
4세대	1970년 초반 이후	비절차 중심의 언어로 보고서 생성기와 데이터베이스 질의 언어(query language) 또는 비주얼 베이직과 같은 비주얼 프로그래밍 언어임
5세대	현재와 미래	영어, 한국어와 같은 진정한 의미의 자연 언어는 없으며, 컴퓨터에 대한 기초지식이 없는 일반인도 코드 없이 프로그램을 만들 수 있는 블록 비주얼 프로그래밍 언어가 소개되어 교육용 프로그래밍 언어로 많이 활용되고 있음

프로그램 구현

프로그래밍

- 프로그램(Program)
  - 컴퓨터에서 특정 목적의 작업을 수행하기 위해 관련된 명령어와 데이터를 모아 놓은 것
  - 컴퓨터에게 지시할 일련의 처리 작업 내용을 담고 있음
  - 프로그래밍(Programming): 프로그래밍 언어를 이용해 실행 프로그램을 만드는 것

- 프로그램 개발 과정
  1. 프로그래밍 언어 및 개발 도구(Development tools) 선정
  2. 소스 코드 작성
  3. 컴파일
  4. 링크
  5. 실행
  6. 디버깅

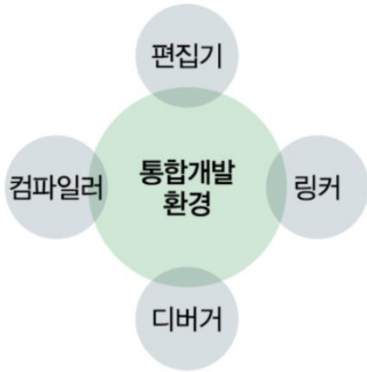
프로그램 개발 환경

- 프로그램 언어로 만들어진 프로그램 소스(source)를 실행 파일로 생성해주는 개발 도구가 필요

- 편집기(Editor)
  - 프로그래밍 언어를 작성하고 편집
- 컴파일러(Compiler)
  - (High level) 프로그래밍 언어로 작성한 프로그램을 컴퓨터가 이해할 수 있는 기계어로 변환
- 링커(Linker)
  - 여러 목적 파일을 하나의 실행 파일로 만들어 주는 기능
- 디버거(Debugger)
  - 작성한 프로그램에서 발생하는 프로그램 오류를 쉽게 찾아 수정할 수 있도록 도와주는 프로그램

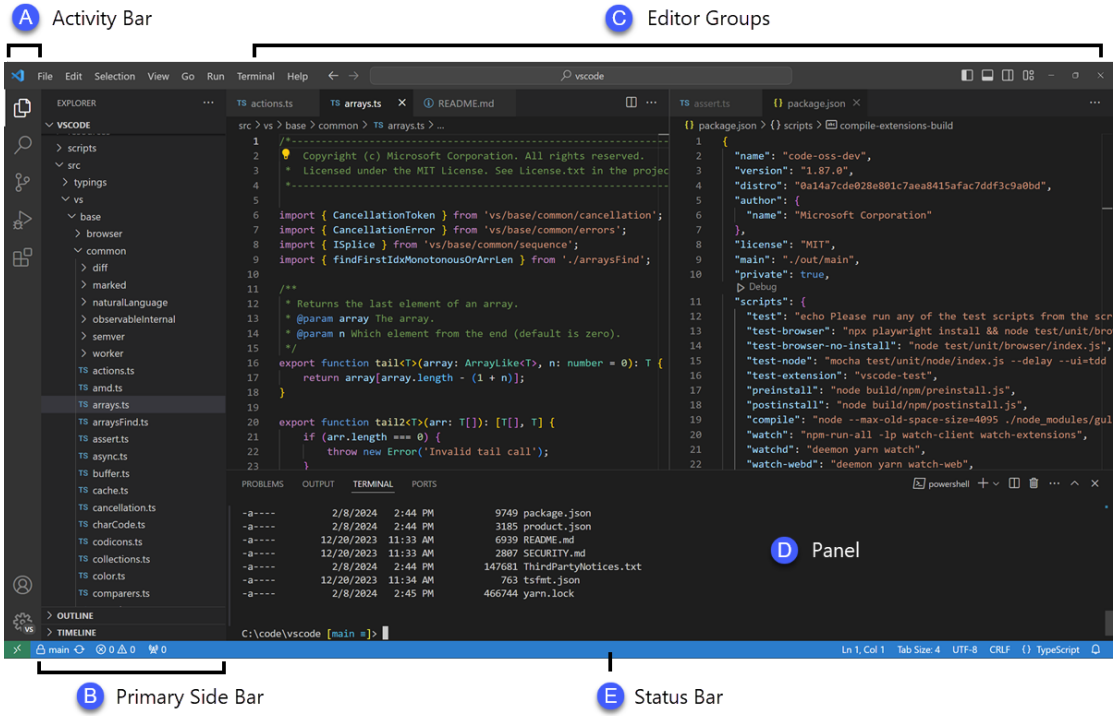
통합개발환경

- Integrated Development Environments (IDE)
  - 프로그램 개발에 필요한 기능들을 통합적으로 제공하는 개발 환경
  - 프로젝트 관리, 에디터, 컴파일러, 디버거, 링커 등



- 종류
  - Visual Studio
  - Visual Studio Code
  - IntelliJ IDEA
  - Eclipse

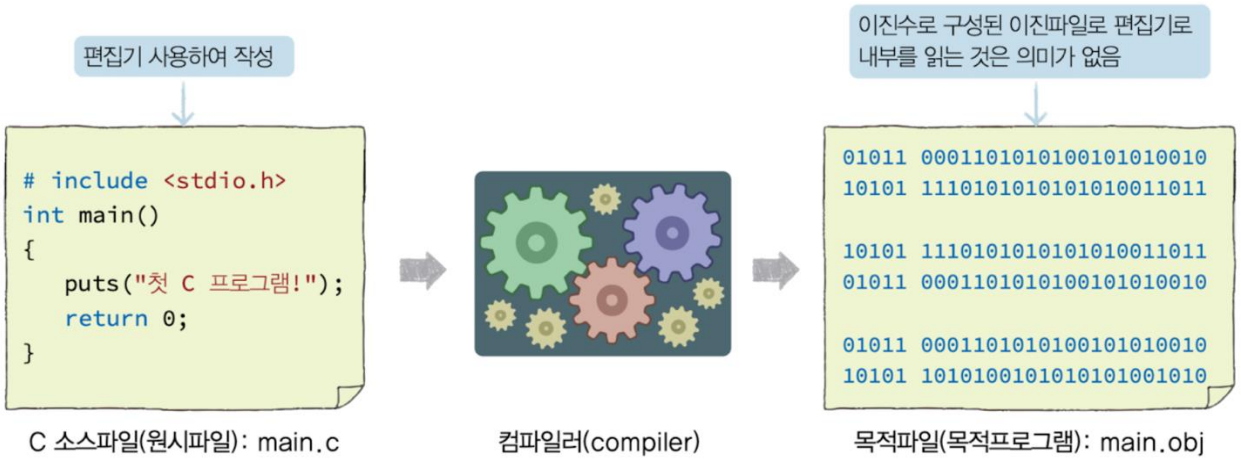
< Visual Studio Code >



소스 코드 작성, 컴파일

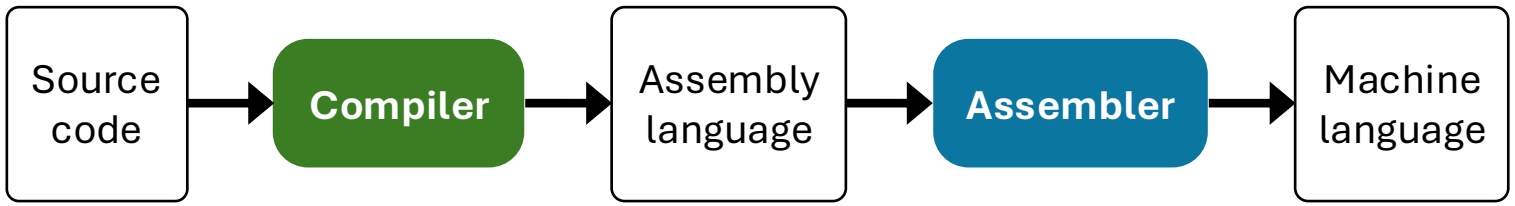
- 소스 코드 작성
  - 프로그래밍 언어를 이용해 작업을 기술한 소스 코드(Source code) 작성 및 소스 파일(Source file) 생성

- 컴파일러(Compiler)
  - 컴파일 해주는 프로그램
  - 컴파일(Compile)
    - High level language로 작성된 프로그램을 기계어로 변환
    - 소스 파일을 목적 파일(Object file)로 변환
      - 목적 파일: 컴파일 과정에 의해 기계어로 번역된 파일



소스 코드 작성, 컴파일

- 소스 코드 작성
  - 프로그래밍 언어를 이용해 작업을 기술한 소스 코드(Source code) 작성 및 소스 파일(Source file) 생성
- 컴파일러(Compiler)
  - 컴파일 해주는 프로그램
  - 컴파일(Compile)
    - High level language로 작성된 프로그램을 기계어로 변환
    - 소스 파일을 목적 파일(Object file)로 변환
      - 목적 파일: 컴파일 과정에 의해 기계어로 번역된 파일
- 어셈블러(Assembler)
  - 어셈블리어로 작성한 프로그램을 기계어로 바꿔주는 프로그램

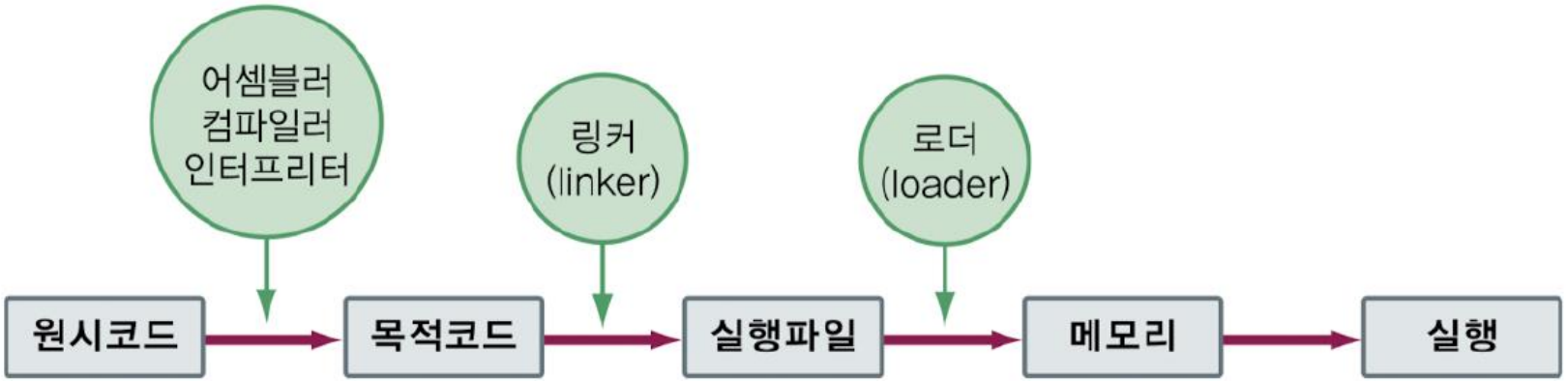


링크, 실행

- 링커(Linker)
  - 목적 파일을 실행 가능한 실행 파일(execute file)로 만들어주는 프로그램
    - 실행 파일? 링커에 의해 생성되는 프로그램 (ex. ".exe" 파일)
  - 여러 개의 목적 파일들을 라이브러리(library) 함수와 연결해서 하나의 파일로 합치는 작업 수행
    - 라이브러리? 자주 사용하는 기능/함수를 미리 만들어 놓은 코드
  - 이러한 과정을 링크(link)라고 부름

- 로더(loader)
  - 실행 파일을 컴퓨터의 주기억장치에 로드(load)하여 실행 가능하게 하는 프로그램

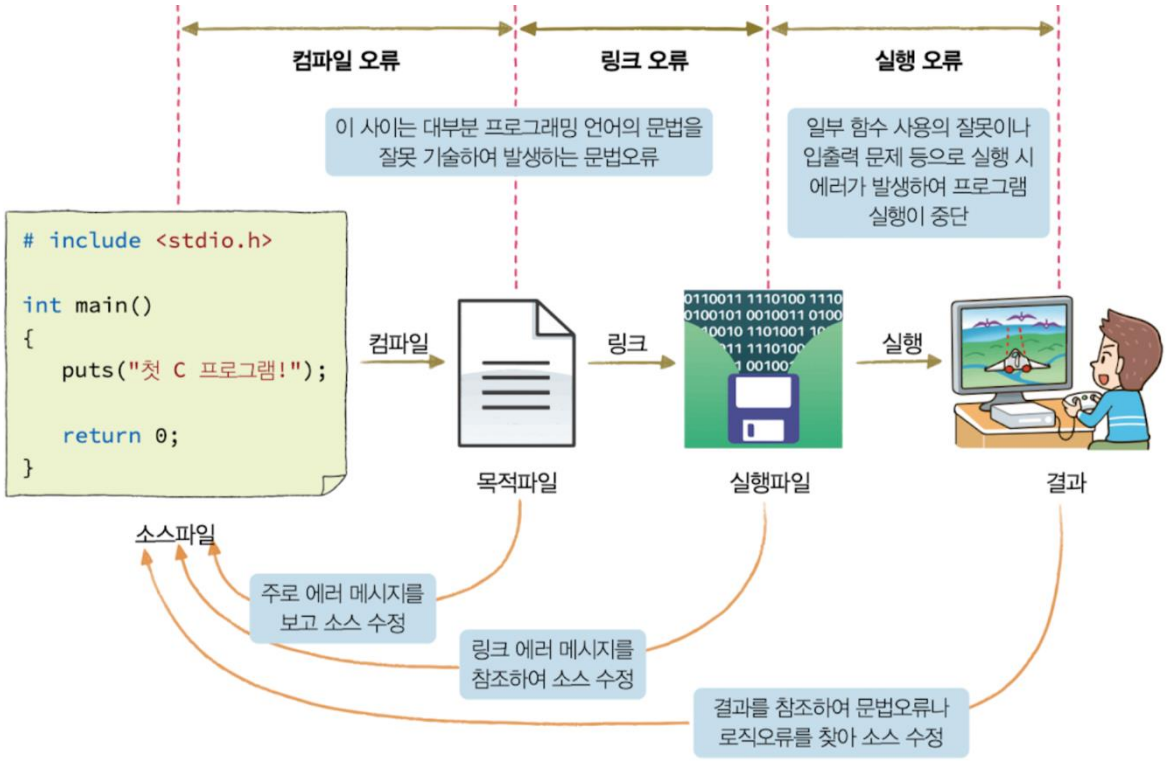
프로그램 구현 및 실행 과정



디버깅

- 디버거(Debugger)
  - 프로그램의 상태를 보여주거나 오류(Bug)를 찾고 분석하는 도구
  - 디버깅(Debuggin): 오류를 수정하는 과정

  - 컴파일 오류
  - 링크 오류
  - 실행 오류

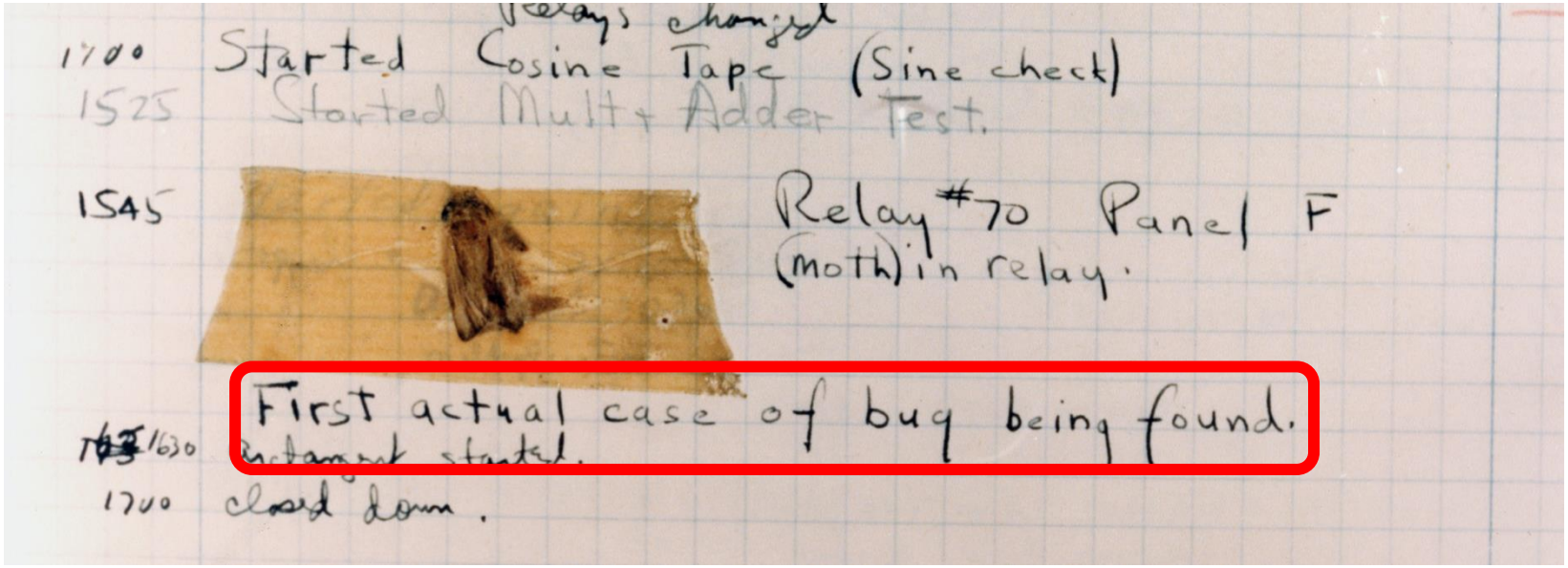


디버깅

- 디버거(Debugger)
  - 프로그램의 상태를 보여주거나 오류(Bug)를 찾고 분석하는 도구
  - 디버깅(Debuggin): 오류를 수정하는 과정

  - Debug: 벌레(bug)를 제거하다
  - 실제 벌레(bug)에 의해 컴퓨터가 오작동 된 이유에서 유래

<벌레 발견 정황을 기록한 작업 일지>



인터프리터

- 인터프리터(Interpreter)
  - High level language를 기계어로 번역
  - 코드 전체를 미리 컴파일 하지 않고 실행
    - 코드 한 줄 씩 읽으면서 실행
  - cf) 컴파일러: 소스 코드 전체를 읽은 후 기계어로 번역

특징 \ 방식	컴파일러	인터프리터
번역 방법	프로그램 전체 번역	실행되는 줄(라인) 단위 번역
장점	한 번 컴파일한 후에는 매번 빠른 시간 내에 전체 실행 가능	번역 과정이 비교적 간단하고 대화형 언어에 편리함
단점	프로그램의 일부를 수정하는 경우에도 전체 프로그램을 다시 컴파일해야 함	실행할 때마다 매번 기계어로 바꾸는 과정을 다시 수행해야 하므로 항상 인터프리터가 필요함
출력물	목적 코드	즉시 실행
언어 종류	FORTRAN, COBOL, C 등	Basic, Python 등

컴파일러와 인터프리터 중간 방식

- 컴파일러와 인터프리터의 중간 방식 언어
  - 컴파일 된 파일을 실행할 때는 인터프리터 필요
  - "시스템에 독립적인 프로그램 언어 개발이 목적"
  - 컴파일러가 존재하여 컴파일 과정이 필요

- Java
  - Write One, Run Anywhere (WORA)

프로그래밍 언어 분류

절차 지향

- Procedural programming
  - 프로그램을 순차적인 실행 순서를 중심으로 설계하는 프로그래밍
  - 순서대로 실행되는 함수 중심
    - 함수(function): 전체 과정을 나누어 처리하는 단위
  - 데이터와 함수가 분리
  - 코드가 커지면 관리 어려움

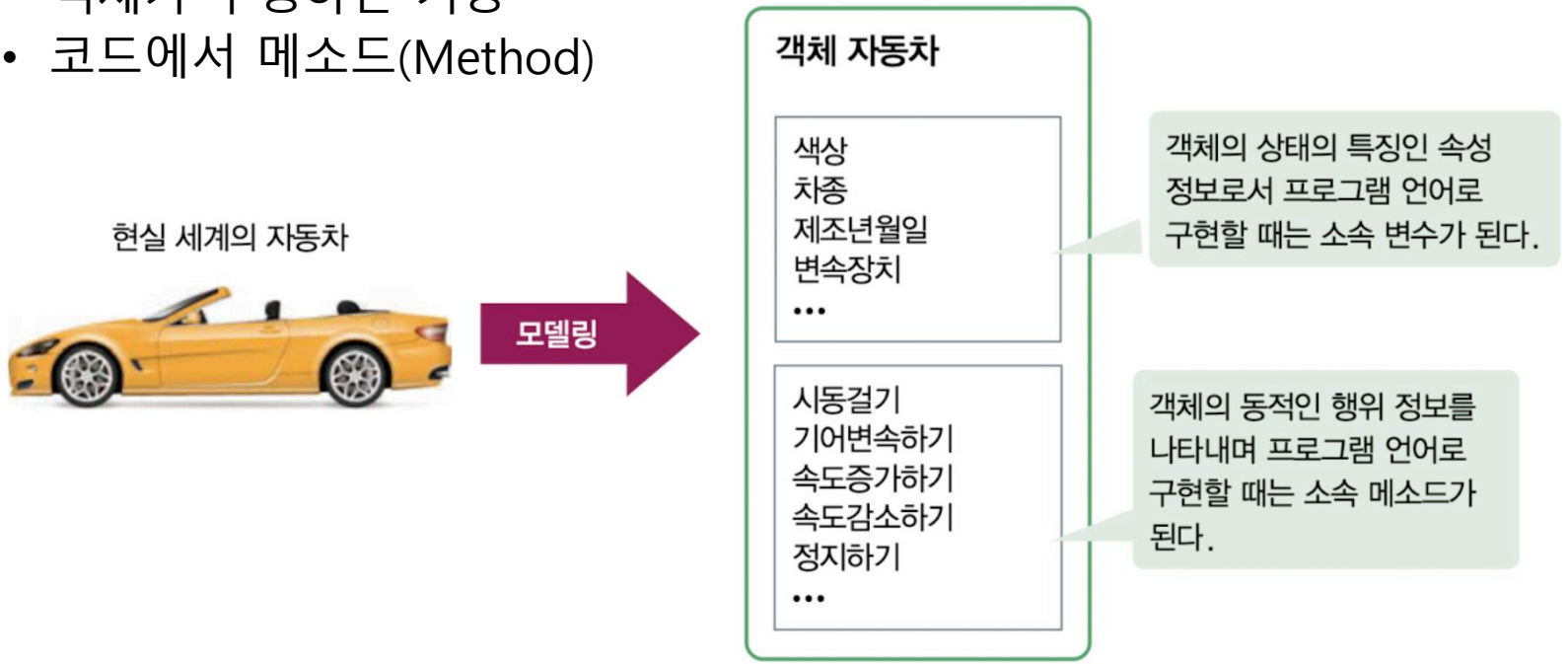
  - FORTRAN, BASIC, COBOL, C 등

객체 지향

- Object-oriented programming
  - 프로그램을 독립된 단위인 '객체(object)'들의 집합으로 파악
    - 현실 세계의 대상을 모델링
    - 객체는 데이터와 기능을 함께 가짐
    - 프로그램: 객체라는 기본 단위 + 이들의 상호 작용
  - 코드 재사용성, 유지보수성 향상
  - Java 등

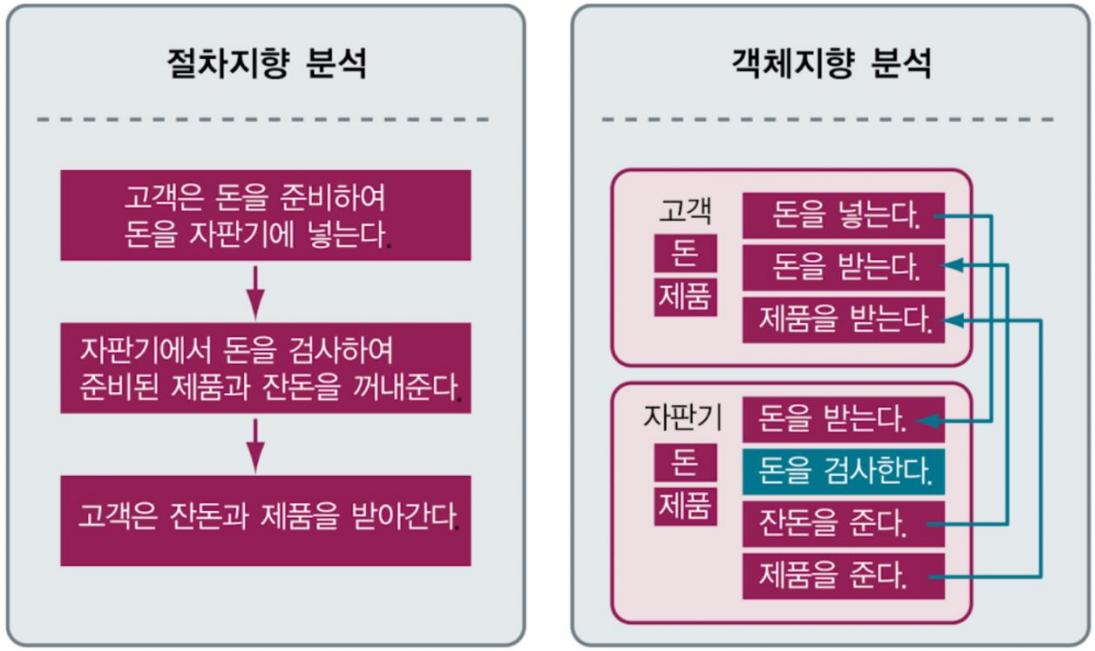
객체 지향 – 객체 구성

- 속성(attribute)
  - 객체의 특성을 표현하는 정적인 성질, 데이터
  - 코드에서 필드(field)
- 행동(behavior)
  - 동적인 일을 처리하는 단위
  - 객체가 수행하는 기능
  - 코드에서 메소드(Method)



절차 지향 vs. 객체 지향

Ex) 자동판매기



- 절차 지향
  - 동사 중심, 함수 (해야 할 업무를 처리하는 단위)
- 객체 지향
  - 객체: 고객, 자판기
  - 구매 과정에서 필요한 내용을 속성과 행동으로 나눔

객체 지향 원칙

- 추상화(Abstraction)
- 캡슐화(Encapsulation)
- 상속성(Inheritance)
- 다형성(Polymorphism)

객체 지향 원칙

- 추상화(Abstraction)
  - 불필요한 세부 구현 숨기고 필요한 기능만 노출
  - **단순화가 목적**
    - 복잡도 줄여 사용자가 핵심 기능에만 집중하게
  - ex) 사용자는 '돈을 넣는다', '버튼 누른다' 같은 기능만 사용하면 됨. 내부에서 '돈 검사' 등의 복잡한 기능 수행

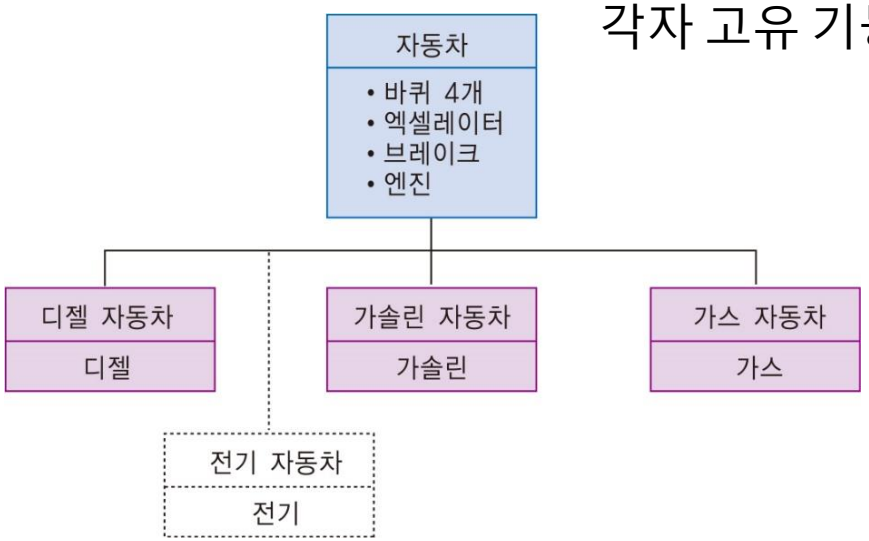
- 캡슐화(Encapsulation)
  - 데이터(상태)와 데이터를 처리하는 함수(행동)을 하나로 묶고, 외부에서 직접 접근하지 못하게 함
  - **외부의 접근을 제한(보호) 하는 것이 목적**
  - ex) 고객은 자판기 내부 상태(잔액, 재고) 건드리지 못함, 반드시 돈 넣기, 제품 선택 같은 메소드 통해서만 접근

객체 지향 원칙

- 상속성(Inheritance)
  - 기존 코드를 재사용
  - 이미 잘 만들어진 클래스(부모)의 특징을 그대로 이어받아 새로운 클래스(자식) 만들 수 있음
    - 클래스: 객체 생성하기 위한 틀, 설계도
    - 기존 클래스의 정의를 물려 받을 수 있음
  - ex) 자판기 종류 확장 가능: 기본 자판기 → 캔음료 자판기

→ 과자 자판기

각자 고유 기능만 추가



객체 지향 원칙

- 다형성(Polymorphism)
  - 같은 메소드 호출에 대해 객체에 따라 다른 동작이 수행 됨
  - 외부에서는 하나의 메소드로 보이지만 내부에서는 다르게 구현됨

  - 재생(Play) 버튼
    - 동영상 앱: 동영상 재생
    - 음악 앱: 음악 재생

객체 지향 원칙

- 다형성(Polymorphism)
  - 같은 메소드 호출에 대해 객체에 따라 다른 동작이 수행 됨
  - 외부에서는 하나의 메소드로 보이지만 내부에서는 다르게 구현됨

상속

다형성

```
class Animal {
   void sound() { System.out.println("Sound"); }
}
class Dog extends Animal {
   void sound() { System.out.println("Bark"); }
}
class Cat extends Animal {
   void sound() { System.out.println("Meow"); }
}
public class Main {
   public static void main(String[] args) {
      Animal a1 = new Dog();
      Animal a2 = new Cat();
      a1.sound();
      a2.sound();
   }
}
```

Summary

- 프로그래밍 언어
  - Low level language
  - High level language
- 프로그래밍 과정
  - 언어 및 개발도구 선정
  - 소스 코드 작성
  - 컴파일, 링크, 실행
- 프로그래밍 언어 분류
  - 절차 지향 vs. 객체 지향