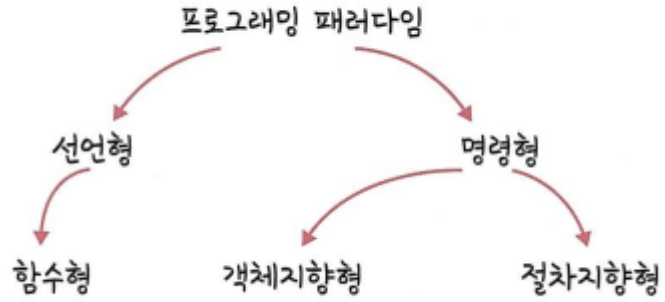
1-2

1.2 프로그래밍 패러다임

- 프로그래밍 패러다임은 프로그래머에게 프로그래밍의 관점을 갖게 해주는 역할을 하는 개발 방법론이다.
- 예를 들어 객체지향 프로그래밍은 프로그래머들이 프로그램을 상호 작용하는 객체들의 집합으로 볼 수 있게 하는 반면에, 함수형 프로그래밍은 상태 값을 지니지 않는 함수 값들의 연속으로 생각할 수 있게 해준다.
- 프로그래밍 패러다임은 크게 선언형, 명령형으로 나누며 선언형은 함수형이라는 하위 집합을 갖는다.
- 명령형은 다시 객체지향, 절차지향으로 나뉜다.

▼ 그림 1-27 프로그래밍 패러다임의 분류



1.2.1 선언형과 함수형 프로그래밍

- 선언형 프로그래밍이란 무엇을 풀어내는가에 집중하는 패러다임이며, 프로그램은 함수로 이루어진 것이다. 라는 명제가 담겨 있는 패러다임이다.
- 함수형 프로그래밍은 선언형 패러다임의 일종이다.
- 함수형 프로그래밍은 이와 같은 작은 '순수 함수'들을 블록처럼 쌓아 로직을 구현하고 '고차 함수'를 통해 재사용성을 높인 프로그래밍 패러다임이다.
- JS는 단순하고 유연한 언어이며, 함수가 일급 객체이기 때문에 객체지향 프로그래밍보다는 함수형 프로그래밍 방식이 선호된다.

순수함수

- 출력이 입력에만 의존하는 것을 의미

```
cosnt pure = (a, b) => {
  return a + b
}
```

- PURE함수는 들어오는 매개변수 a,b 에만 영향을 받는다.
- 만약 a, b 말고 다른 전역 변수 c등이 출력에 영향을 주면 순수 함수가 아니다.

고차 함수

- 고차 함수란 함수가 함수를 값처럼 매개변수로 받아 로직을 생성할 수 있는 것을 말한다.

일급 객체

- 고차 함수를 쓰기 위해서는 해당 언어가 일급 객체라는 특징을 가져야 하며 그 특징은 다음과 같다
  - 변수나 메서드에 함수를 할당할 수 있다.
  - 함수 안에 함수를 매개변수로 담을 수 있다.
  - 함수가 함수를 반환할 수 잇다.
- 함수형 프로그래밍은 이외에도 커링, 불변선등 많은 특징이 있다.

1.2.2 객체지향 프로그래밍

- 객체지향 프로그래밍(OOP)은 객체들의 집합으로 프로그램의 상호 작용을 표현하며 데이터를 객체로 취급하여 객체 내부에 선언된 메서드를 활용하는 방식을 말한다.
- 설계에 많은 시간이 소요되며 처리 속도가 다른 프로그래밍 패러다임에 비해 상대적으로 느리다.

객체지향 프로그래밍의 특징

- 객체지향 프로그래밍은 추상화, 캡슐화, 상속성, 다형성이라는 특징이 있다.

추상화

- 추상화(abstraction)란 복잡한 시스템으로부터 핵심적인 개념 또는 기능을 간추려내는 것을 의미.
- 여러 특징중에 일부분인 특징만 간추려서 나타냄

캡슐화

- 캡슐화(encapsulation)는 객체의 속성과 메서드를 하나로 묶고 일부를 외부에 감추어 은닉 하는 것을 말한다,

상속성

- 상속성(inheritance)는 상위 클래스의 특성을 하위 클래스가 이어받아서 재사용하거나 추가, 확장하는 것을 말한다.
- 코드의 재사용 측면, 계층적인 관계 생성, 유지 보수성 측면에서 중요

다형성

- 다형성(polymoprphsim)은 하나의 메서드나 클래스가 다양한 방법으로 동작하는 것을 말한다.
- 대표적으로 오버로딩, 오버라이딩이 있다.

오버로딩

- 오버로딩은 같은 이름을 가진 메서드를 여러 개 두는 것을 말한다.
- 메서드의 타입, 매개변수의 유형, 개수 등으로 여러 개를 둘 수 있으며 컴파일 중에 발생하는 ‘정적’ 다형성이다.

```
class Person {

    public void eat(String a) {
```

```
        System.out.println("I eat " + a);
    }

    public void eat(String a, String b) {
        System.out.println("I eat " + a + " and " + b);
    }
}

public class CalculateArea {

    public static void main(String[] args) {
        Person a = new Person();
        a.eat("apple");
        a.eat("tomato", "phodo");
    }
}
/*
I eat apple
I eat tomato and phodo
*/
```

오버라이딩

- 오버라이딩은 주로 메서드 오버라이딩을 말하며 상위 클래스로부터 상속받은 메서드를 하위 클래스가 재정의 하는 것을 의미.
- 이는 런타임 중에 발생하는 동적 다형성이다.

```
class Animal {
    public void bark() {
        System.out.println("mumu! mumu!");
    }
}

class Dog extends Animal {
    @Override
    public void bark() {
        System.out.println("wal!!! wal!!!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.bark();
    }
}
/*
wal!!! wal!!!
*/
```

설계 원칙

- 객체지향 프로그래밍을 설계할 때는 SOLID원칙을 지켜주어야 한다.
- S는 단일 책임 원칙
- O는 개방-폐쇄 원칙
- L은 리스코프 치환 원칙
- D는 의존성 역전 원칙을 의미한ㄷ.

단일 책임 원칙

- 단일 책임 원칙(SRP, Single Responsibility Principle)은 모든 클래스는 각각 하나의 책임만 가져야 하는 원칙이다.
- 예를 들어 A라는 로직이 존재한다면 어떠한 클래스는 A에 관한 클래스여야 하고 이를 수정한다고 했을 때도 A와 관련된 수정잉 ㅑ한다.

개방-폐쇄 원칙

- 개방-폐쇄 원칙(OCP, Open Closed Principle)은 유지 보수 사항이 생긴다면 코들르 쉽게 확장할 수 있도록 하고 수정할 떄는 닫혀 있어야 하는 원칙이다.
- 즉 기존의 코드는 잘 변경하지 않으면서도 확장은 쉽게 할 수 있어야 한다.

리스코프 치환원칙

- 리스코프 치환원칙(LSP)은 프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 하는 것을 의미.
- 클래스는 상속이 되기 마련이고 부모, 자식이라는 계층 관계가 만들어 진다. 이때 부모 객체에 자식 객체를 넣어도 시스템이 문제 없이 돌아가게 만드는 것을 말한다.

인터페이스 분리 원칙

- 인터페이스 분리 원칙(ISP, Interface Segregation Principle)은 하나의 일반적인 인터페이스보다 구체적인 여러 개의 인터페이스를 만들어야 하는 원칙을 말한다.

의존 역전 원칙

- 의존 역전 원칙(DIP, Dependency Inversion Principle)은 자신보다 변하기 쉬운 것에 의존하던것을 추상화된 언터페이스나 상위 클래스를 두어 변하기 쉬운 것의 변화에 영향받지 않게 하는 원칙
- 타이어를 갈아끼울 수 있는 틀을 만들어 놓은 후 다양한 타이어를 교체할 수 있어야 한다.
- 즉, 상위 계층은 하위 계층의 변화에 대한 구현으로부터 독립해야 한다.

1.2.3 절차형 프로그래밍

- 절차형 프로그래밍은 로직이 수행되어야할 연속적인 계산 과정으로 이루어져 있다.
- 일이 진행되는 방식으로 코드를 구현하기만 하면 되기 때문에 코드의 가독성이 좋으며 실행속도가 빠르다.
- 계산이 많은 작업에 많이 쓰이며 대표적으로 포트란을 이용한 대기 과학 관련 연산 작업, 머신 러닝의 배치 작업이 있다.
- 단점으로 모듈화하기 어렵고 유지 보수성이 떨어진다는 단점이 있다.

1.2.4 패러다임 혼합

- 어떠한 패러다임이 가장 좋다 ? 는 없다 !
- 비즈니스 로직이나 서비스의 특징을 고려해서 패러다임을 정하는것이 좋다.
- 하나의 패러다임을 기반으로 통일하여 서비스를 구축하는 것도 좋은 생각이지만 여러 패러다임을 조합하여 상황과 맥락에 따라 패러다임간에 장점만 취해 개발하는 것이 좋다.!