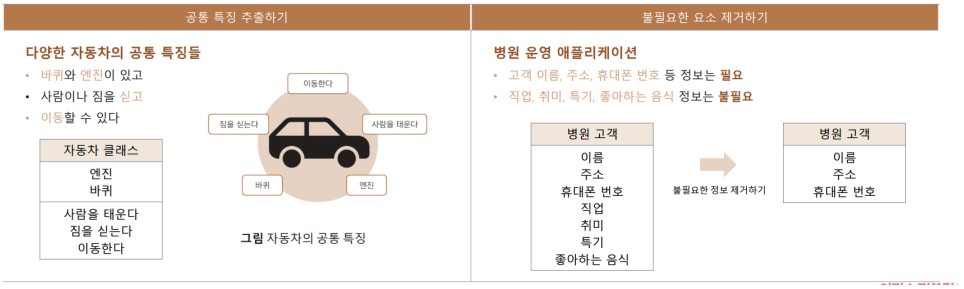
Chapter 07 - 추상화와 캡슐화

생성일	@2025년 9월 29일 오후 6:06
태그	

01) 추상화와 캡슐화

추상화

- 공통 특징 추출, 불필요한 부분은 제거하여 멤버 변수, 멤버 함수를 규정하는 것



캡슐화

- 변수와 함수를 클래스로 감싸서 외부에서 직접 접근할 수 없도록 은닉화 하는 것
- 하나의 기능이 여러 함수로 이루어질 경우, 대표 함수만 노출
  - 정보를 클래스화 하면

    → 클래스 내부의 세부 로직을 알 필요가 없음 → 클래스 변경이 외부 로직에 영향을 주지 않음

    → 복잡도는 낮아지고, 재사용성은 높아짐

- 은닉화
  - 아무나 접근할 수 없도록 숨기는 것

- **접근 지정자**

- public으로 선언한 멤버는 다른 클래스 어디서든 접근할 수 있음
- private 멤버는 같은 클래스 안의 멤버 함수에서만 접근할 수 있음
- 클래스의 멤버를 선언할 때는 접근 지정자를 지정해야 하며, 만약 지정하지 않으면 private로 선언된다.

접근 지정자 적용하기

```
#include <iostream>
using namespace std;

class bank {
    private:
        int safe;   // 금고

    public :
        bank();
        void use_counter(int _in, int _out);    // 입출금
창구 함수
};

bank::bank() {
    safe = 1000;    // 은행 초기 금액 설정
    cout << "최초 금고 : " << safe << endl;
    cout << endl;
}

void bank::use_counter(int _in, int _out) {
    safe += _in;    // 입금
    safe -= _out;   // 출금

    cout << "입금 : " << _in << endl;
    cout << "출금 : " << _out << endl;
    cout << "금고 : " << safe << endl;
    cout << endl;
}

int main()
```

```
{
    bank my_bank;   // my_bank 인스턴스 생성

    my_bank.use_counter(0, 20); // 출금 20
    my_bank.use_counter(50, 0); // 입금 50
    my_bank.use_counter(100, 50); // 입금 100, 출금 50

    return 0;
}
```

몬스터 클래스 적용하기

```
#include <iostream>
using namespace std;

// 캐릭터 클래스
class character {
public:
    character() : hp(100), power(100) {};

protected:
    int hp;
    int power;
};

// 플레이어 클래스
class player : public character {
public:
    player() {};
};

// 기본 몬스터 클래스
class monster {
public:
    monster() {};
    void get_damage(int _damage) {};
    void attack(player target_player) {};
    void attack_special(player target_player);
```

```
};

void monster::attack_special(player target_player) {
    cout << "기본 공격 : 데미지 - 10 hp" << endl;
}

// 기본 몬스터 클래스 상속
class monster_a : public monster, character {
public:
    // 상속받은 함수 오버라이딩
    void attack_special(player target_player);
};

void monster_a::attack_special(player target_player) {
    cout << "인텡글 공격 : 데미지 - 15 hp" << endl;
}

// 기본 몬스터 클래스 상속
class monster_b : public monster, character {
public :
    // 상속받은 함수 오버라이딩
    void attack_special(player target_player);
};

void monster_b::attack_special(player target_player) {
    cout << "가상 공격 : 데미지 - 0 hp" << endl;
}

// 기본 몬스터 클래스 상속
class monster_c : public monster, character {
public:
    // 상속받은 함수 오버라이딩
    void attack_special(player target_player);
};

void monster_c::attack_special(player target_player) {
    cout << "강력 뇌전 공격 : 데미지 - 100 hp" << endl;
}
```

```
int main()
{
    player player_1;

    monster_a forest_monster;
    monster_b tutorial_monster;
    monster_c boss_monster;

    cout << "몬스터 총 공격" << endl;
    forest_monster.attack_special(player_1);
    tutorial_monster.attack_special(player_1);
    boss_monster.attack_special(player_1);

    return 0;
}
```

02) 상속성과 다형성