

트하기 위해 프로그램의 main 함수는 사용자가 Viewer와 Owner 객체로부터 메뉴를 호출하도록 한다.

9. 환상적인 롤플레이 게임을 만든다고 가정하자. 이 게임에는 서로 다른 네 가지 종족이 있다 Humans, Cyberdemons, Balrogs, Elves. 이 종족 중 하나를 나타내기 위해 다음과 같이 클래스 Creature를 정의한다.

```
class Creature
{
    private:
        int type; //0 human, 1 cyberdemon, 2 balrog, 3 elf
        int strength; //가할 수 있는 손상의 양
        int hitpoints; //버틸 수 있는 손상의 양
        string getSpecies(); //종족의 종류를 반환

    public:
        Creature();
        //human, 10 strength, 10 hit points로 초기화한다.

        Creature(int newType, int newStrength, int newHit);
        //종족을 new type, strength, hit points로 초기화한다.

        //또한 type, strength, hit points에 대해서
        //적절한 accessor와 mutator 함수를 추가한다.

        int getDamage();
        //한 라운드의 전투에서 가해지는
        //종족들의 손상 값을 반환한다.
};
```

여기에 함수 getSpecies()의 구현이 있다.

```
string Creature::getSpecies()
{
    switch(type)
    {
        case 0: return "Human";
        case 1: return "Cyberdemon";
        case 2: return "Balrog";
        case 3: return "Elf";
    }
    return "Unknown";
}
```

함수 `getDamage()`는 한 라운드의 전투에서 종족들이 가하는 손상의 양을 반환하고 출력 한다. 손상을 계산하는 규칙은 다음과 같다.

- 각 종족들이 가하는 손상은 $0 < r \leq strength$ 사이의 임의의 수 r 이다.
- Demon은 추가로 50 손상 점수를 주는 악령 공격을 할 수 있는 5%의 기회를 가진다.

Balrogs와 Cyberdemons는 demon이다.

- Elves는 일반 손상 양의 2배인 마법 공격을 할 수 있는 10%의 기회를 가진다.
- Balrogs는 매우 빠르기 때문에 두 번 공격한다.

`getDamage()`의 구현은 다음과 같다.

```
int Creature::getDamage()
{
    int damage;
    //모든 종족이 가하는 손상은
    //strength 내의 임의의 수이다.
    damage = (rand() % strength) + 1;
    cout << getSpecies() << "attacks for"
        damage << " points!" << endl;
    //Demon은 5%의 기회로 50의 손상을 가할 수 있다.
    if((type == 2) || (type == 1))
        if((rand() % 100) < 5)
    {
        damage = damage + 50;
        cout << "Demonic attack inflicts 50"
            << " additional damage points!" << endl;
    }
    //Elves는 10%의 기회로 2배의 마법 손상을 가할 수 있다.
    if(type == 3)
    {
        if((rand() % 10) == 0)
    {
        cout << "Magical attack inflicts " << damage <<
            " additional damage points!" << endl;
        damage = damage * 2;
    }
    }
    //Balrogs는 빠르므로 2번 공격한다.
    if(type == 2)
    {
        int damage2 = (rand() % strength) + 1;
```

```

cout << "Balrog speed attack inflicts "<< damage2 <<
    " additional damage points!" << endl;
damage = damage + damage2;
}

return damage;
}

```

이 구현에서 한 가지 문제는 새로운 종족을 추가하는 것이 불편하다는 것이다. 변수형에 대한 필요를 제거하도록 상속을 사용하여 클래스를 재작성하라. 클래스 `Creature`는 기반 클래스가 되어야 한다. 클래스 `Demon`, `Elf`, `Human`은 `Creature`로부터 파생되어야 한다. 클래스 `Cyberdemon`과 `Balrog`은 `Demon`으로부터 파생되어야 한다. 각 클래스에 적절하게 맞게 함수 `getSpecies()`와 `getDamage()`를 다시 작성할 필요가 있다.

예를 들어, 함수 `getDamage()`는 각 클래스에서 오직 객체에 대한 적절한 손상을 계산해야 한다. 총손상은 상속 계층의 각 단계에서 `getDamage()`의 결과를 결합하여 계산한다. 예를 들어, `Balrog` 객체에 대해 `getDamage()` 호출은 `Demon` 객체에 대한 `getDamage()`를 호출하고, 이것은 `Creature` 객체의 `getDamage()`를 호출한다. 모든 종족이 가하는 기본 손상을 계산하고, `Demon`이 가하는 임의의 5% 손상을 계산하고, `Balrog`가 가하는 2배의 손상을 계산한다.

또한 `private` 변수에 대한 `mutator`와 `accessor` 함수도 포함하라. 클래스들을 테스트할 수 있는 드라이버를 포함한 `main` 함수를 작성하라. 이 프로그램은 각 종족에 대한 객체를 생성하고 반복적으로 `getDamage()`의 결과를 출력해야 한다.

10. 애완동물의 이름, 나이, 무게를 저장하는 클래스 `Pet`을 정의하라. 적절한 생성자, `accessor` 함수, `mutator` 함수를 추가하라. 또한 “unknown lifespan” 값을 가진 문자열을 반환하는 함수 `getLifespan`을 정의하라.

다음으로 `Pet`으로부터 파생된 클래스 `Dog`을 정의하라. 클래스 `Dog`은 강아지의 품종을 저장하는 `breed`란 `private` 멤버 변수를 가져야 한다. 변수 `breed`와 적절한 생성자에 대한 `mutator`와 `accessor` 함수를 추가하라. 강아지의 무게가 100파운드를 넘으면 “약 7년”을, 강아지의 무게가 100파운드를 넘지 않으면 “약 13년”을 출력하는 함수 `getLifespan`을 재정의하라.

다음으로, `Pet`으로부터 파생된 클래스 `Rock`을 정의한다. “Thousands of years”를 반환하는 함수 `getLifespan`을 재정의한다.

마지막으로 상속과 재정의 함수를 연습하기 위한 `pet rocks`와 `pet dogs`의 인스턴스를 생성하는 테스트 프로그램을 작성하라.

조가 필요하다.

5마리 개미귀신과 100마리 개미를 가진 세계를 초기화하라. 각 단계 후에 사용자에게 다음 단계로 이동하기 위해 Enter 키를 누르도록 유도하라. 비록 임의의 변화가 하나 또는 두 종을 제거할 수도 있지만 포식자와 먹이 개체군 사이의 주기적인 패턴을 보아야 한다.

4. 이 프로그래밍 프로젝트를 하기 전에 우선 14장의 프로그래밍 프로젝트 14.9를 해야 한다.

```

class Creature
{
    private:
        int type; //0 human, 1 cyberdemon, 2 balrog, 3 elf
        int strength; //가할 수 있는 손상의 양
        int hitpoints; //버틸 수 있는 손상의 양
        string getSpecies(); //종족의 종류를 반환

    public:
        Creature();
        //human, 10 strength, 10 hit points으로 초기화한다.

        Creature(int newType, int newStrength, int newHit);
        //종족을 newType, strength, hit points로 초기화한다.

        //또한 type, strength, hit points에 대해서
        //적절한 accessor와 mutator 함수를 추가한다.

        int getDamage();
        //한 라운드의 전투에서 가해지는
        //종족들의 손상 값을 반환한다.

};

```

여기에는 함수 getSpecies()의 구현이 있다.

```

string Creature::getSpecies()
{
    switch(type)
    {
        case 0: return "Human";
        case 1: return "Cyberdemon";
        case 2: return "Balrog";
        case 3: return "Elf";
    }
    return "Unknown";
}

```

함수 `getDamage()`는 한 라운드의 전투에서 종족들이 가하는 손상의 양을 반환하고 출력한다. 손상을 계산하는 규칙은 다음과 같다.

- 각 종족들이 가하는 손상은 $0 < r \leq strength$ 사이의 임의의 수 r 이다.
- Demon은 추가로 50 손상 점수를 주는 악령 공격을 가하는 5%의 기회를 가진다. Balrogs 와 Cyberdemons는 Demon이다.
- Elves는 일반 손상 양의 2배인 마법 공격을 가하는 10%의 기회를 가진다.
- Balrogs는 매우 빠르기 때문에 두 번 공격한다.

`getDamage()`의 구현은 다음과 같다.

```
int Creature::getDamage()
{
    int damage;
    //모든 종족이 가하는 손상은
    //strength 내의 임의의 수이다.
    damage = (rand() % strength) + 1;
    cout << getSpecies() << "attacks for"
        damage << " points!" << endl;

    //Demon은 5%의 기회로 50의 손상을 가할 수 있다.
    if ((type == 2) || (type == 1))
        if ((rand() % 100) < 5)
    {
        damage = damage + 50;
        cout << "Demonic attack inflicts 50:"
            << " additional damage points!" << endl;
    }

    //Elves는 10%의 기회로 2배의 마법 손상을 가할 수 있다.
    if (type == 3)
    {
        if ((rand() % 10) == 0)
        {
            cout << "Magical attack inflicts " << damage <<
                " additional damage points!" << endl;
            damage = damage * 2;
        }
    }

    //Balrogs는 빠르므로 두 번 공격한다.
    if (type == 2)
```

```

    {
        int damage2 = (rand() % strength) + 1;
        cout << "Balrog speed attack inflicts " << damage2 <<
            "additional damage points!" << endl;
        damage = damage + damage2;
    }
    return damage;
}

```

이 구현에서 한 가지 문제는 새로운 종족을 추가하는 것이 불편하다는 것이다. 변수형에 대한 필요를 제거하도록 상속을 사용하여 클래스를 재작성하라. 클래스 Creature는 기반 클래스가 되어야 한다. 클래스 Demon, Elf, Human은 Creature로부터 파생되어야 한다. 클래스 Cyberdemon과 Balrog는 Demon으로부터 파생되어야 한다. 각 클래스에 적절하게 맞게 함수 getSpecies()와 getDamage()를 다시 작성할 필요가 있다.

예를 들어, 함수 getDamage()는 각 클래스에서 오직 객체에 대한 적절한 손상을 계산해야 한다. 총손상은 상속 계층의 각 단계에서 getDamage()의 결과를 결합하여 계산한다. 예를 들어, Balrog 객체에 대해 getDamage() 호출은 Demon 객체에 대한 getDamage()를 호출하고, 이것은 Creature 객체의 getDamage()를 호출한다. 모든 종족이 가하는 기본 손상을 계산하고, Demon이 가하는 임의의 5% 손상을 계산하고, Balrog가 가하는 2배의 손상을 계산한다.

또한 private 변수에 대한 mutator와 accessor 함수도 포함하라. 클래스들을 테스트할 수 있는 드라이버를 포함한 main 함수를 작성하라. 이 프로그램은 각 종족에 대한 객체를 생성하고 반복적으로 getDamage()의 결과를 출력해야 한다. 첫 번째로 getDamage() 함수를 가상으로 만들라. 그리고 메인 프로그램에 입력으로 2개의 Creature 객체를 받는 함수 battleArena(Creature &creature1, Creature &creature2)를 만들라. 이 함수는 creature1에 의한 손상을 계산하여 creature2의 hit points에서 그 값을 빼야 하고 그 반대 경우도 있다. 두 종족 모두 hit points가 0이거나 0보다 작으면 그 전투는 비기게 된다. 그렇지 않으면 라운드 끝에 한 종족의 hit points가 양이고 다른 종족은 그렇지 않다면 그 전투는 끝난다. 이 함수는 전투를 비기거나 끝날 때까지 반복한다. 함수 getDamage()가 가상이기 때문에 특정한 종족을 위해 정의된 함수 getDamage()를 호출해야 한다. 다른 종족들을 포함하는 여러 전투로 프로그램을 테스트하라.

5. 다음은 두 선수가 숫자를 추측으로 맞추려고 시도하는 추측 게임을 하는 코드이다. 과제는 사람 선수와 컴퓨터 선수를 표현하는 객체를 가진 프로그램으로 확장하는 것이다.

```

bool checkForWin(int guess, int answer)
{
    if (answer == guess)

```