Zadania na ogarnięcie polimorfizmu

- Zakładamy, że budujemy grę RPG (czysto hipotetycznie, wszystko nadal odbywa się w konsoli).
- Zależy mi na uwzględnieniu nie tylko „gramatyki" języka, ale również sposobu myślenia związanego z projektowaniem programu.
- Wszystkie zadania sugeruję wykonywać dzieląc kod na pliki źródłowe i nagłówkowe.
- Nie wszystko musi być powiedziane wprost. Testujcie każdy fragment, kombinujcie. Wszystko można rozbudowywać według własnych koncepcji.

Zadanie 1

Napisz interfejs Character. Interfejs to klasa, która nie posiada żadnych pól i zawiera wyłącznie metody abstrakcyjne (czysto wirtualne – **wirtualne bez definicji**). Będzie to klasa nadrzędna dla wszystkich postaci występujących w grze (zarówno naszego bohatera, jak i przeciwników).

Interfejs powinien zawierać następujące metody:

- int healthPercent() // zwraca procent życia postaci (od 0 do 100)
- void takeDamage(int damage) // postać przyjmuje obrażenia (jej życie spada)
- void attack(Character* enemy) // postać atakuje inną postać

Zadanie 2

Napisz klasę Warrior, realizującą interfejs Character (dziedziczącą po klasie Character i implementującą metody abstrakcyjne). Będzie to klasa postaci występującej w grze (jedna z kilku dostępnych).

Klasa powinna zawierać prywatne pola:

- int maxHealth // maksymalny poziom punktów życia
- int health // bieżący poziom punktów życia
- int strength // poziom siły (ataku) bohatera

Napisz konstruktor inicjalizujący oba pola związane ze zdrowiem wartością 120. Pole strength powinno zostać zainicjalizowane wartością 10.

Zaimplementuj metody z interfejsu Character w następujący sposób:

- metoda healthPercent() powinna obliczać i zwracać procent, jaki stanowi bieżący poziom punktów życia w stosunku do maksymalnego poziomu (jeżeli postać jest całkowicie zdrowa, powinna zwracać 100; jeżeli martwa, powinna zwracać 0)
- metoda takeDamage() przyjmuje jako argument liczbę punktów obrażeń, jaką postać otrzymuje podczas ataku przeciwnika – należy odjąć tę wartość od bieżącego zdrowia postaci (dbając przy tym, aby zdrowie nie przyjmowało wartości poniżej 0)
- metoda attack() realizuje atak na postać przeciwnika – należy wywołać metodę takeDamage() na obiekcie przeciwnika otrzymanym w argumencie (tym samym zabierając mu część życia); liczba zadanych punktów obrażeń powinna być równa sile naszej postaci

Metody takeDamage() oraz attack() mogą wypisywać komunikaty tekstowe, obrazujące co dzieje się wewnątrz programu (np. „wojownik otrzymuje X punktów obrażeń").

Zadanie 3

Przetestuj kod.

- w funkcji main() stwórz dynamicznie obiekt klasy Warrior
- sprawdź wartość zwracaną przez healthPercent() - nowa postać powinna być całkowicie zdrowa, więc powinno być to 100
- zadaj postaci 12 punktów obrażeń, wywołując metodę takeDamage()
- ponownie sprawdź poprawność wartości healthPercent()
- każ postaci zaatakować samą siebie (odpowiednio wywołując metodę attack())
- sprawdź rezultat metodą healthPercent()
- zadaj postaci obrażenia większe, niż jej bieżąca liczba punktów życia
- upewnij się, że healthPercent() zwraca 0 dla martwej postaci (i nie spada poniżej zera)

Po zakończeniu testów można usunąć kod testujący.

Zadanie 4

Stwórz kolejny rodzaj postaci. Zaimplementuj klasę Archer, również realizującą interfejs Character.

Klasa powinna zawierać cztery pola:

- int maxHealth // maksymalny poziom punktów życia
- int health // bieżący poziom punktów życia
- int strength // poziom siły (ataku) bohatera
- int dodgeChance // procentowa szansa uniknięcia ataku przeciwnika
- int criticalChance // procentowa szansa zadania obrażeń krytycznych

Konstruktor powinien inicjalizować maksymalne i bieżące życie wartością 90, siłę wartością 7, szansę uniknięcia ataku wartością 20, a szansę zadania obrażeń krytycznych wartością 25.

Metody powinny zostać zaimplementowane w następujący sposób:

- metoda healthPercent() - analogicznie, jak w klasie Warrior
- metoda takeDamage() powinna zawierać element losowy (należy wykorzystać funkcję rand()) - na podstawie wartości dodgeChance podejmowana jest decyzja, czy postać przyjmuje obrażenia, czy wykonuje unik (w przypadku uniku wartość bieżącego życia nie zmienia się)
- metoda attack() również powinna zawierać element losowy – na podstawie wartości criticalChance podejmowana jest decyzja o zadaniu obrażeń równych sile postaci (standardowy atak), lub równych trzykrotności tej siły (obrażenia krytyczne)

Można rozszerzyć to zadanie i stworzyć więcej podobnych klas. Niektóre postacie mogą np. posiadać pancerz, który redukuje przyjmowane obrażenia o 20%. Inne mogą odzyskiwać część własnego życia raz na trzy zadane ataki itd. - możliwości jest mnóstwo.

Zadanie 5

Przetestuj nową klasę / klasy. Można tymczasowo zmienić domyślne wartości pól dodgeChange oraz criticalChance (przy wartości 0 zdarzenie losowe nie powinno zachodzić, przy wartości 100 powinno zachodzić zawsze). Pamiętaj o użyciu funkcji srand() na początku main().

Zadanie 6

W funkcji main() stwórz wskaźnik na obiekt klasy Character – nazwij go player.

Zaimplementuj proste menu wyboru – pozwól użytkownikowi wybrać, jaką klasą postaci chce kierować. Na podstawie jego decyzji stwórz dynamicznie obiekt odpowiedniej klasy i przypisz go do przygotowanego wskaźnika.

Stwórz kolejny wskaźnik na obiekt klasy Character – nazwij go enemy. Losowo (używając funkcji rand()) wybierz klasę postaci przeciwnika. Dynamicznie stwórz odpowiedni obiekt i przypisz go do wskaźnika.

Zadanie 7

Zaimplementuj pętlę (w funkcji main()), w której przeciwnicy będą się wzajemnie atakować.

Pętla powinna sprowadzać się do odpowiedniego wywoływania metody attack() na obu obiektach (zauważ, że nie trzeba „ręcznie” wywoływać metody takeDamage() w main() - powinna być wywoływana przez „atakujące” obiekty na celach ich ataku).

Po każdym obrocie pętli należy wyświetlić informację o stanie życia obu postaci (używając metody healthPercent()). Pętla powinna obracać się tak długo, aż życie jednego z bohaterów spadnie do zera.

Symulacja powinna kończyć się informacją o tym, czy użytkownik wygrał, czy przegrał walkę.