

Zadanie F - Igrzyska

Punktów do uzyskania: **10**

Zadanie polega na implementacji zestawu klas obsługujących walki na arenie amfiteatru w czasach Cesarstwa Rzymskiego.

Ogólne klasy

- Bazową klasą implementującą graczy jest `PLAYER_CLASS` spełniająca warunki:
 - Jest klasa abstrakcyjną.
 - Przechowuje poniższe informacje mieszczące się w zakresie typu **`unsigned int`**:
 - Maksymalne zdrowie.
 - Aktualne zdrowie.
 - Atak.
 - Zwinność.
 - Wymaga implementacji poniższych metod w szczegółach opisanych dalej.
 - `unsigned int`** `getRemainingHealth ()` - publiczna, zwracającą pozostałe zdrowie w rozumieniu ilorazu aktualnego zdrowia i maksymalnego zdrowia zaokrągloną w dół.
 - `unsigned int`** `getDamage ()` – publiczna, zwracając obrażenia zadawane wrogowi.
 - `unsigned int`** `getAgility ()` – publiczna, zwraca zwinność.
 - `void`** `takeDamage (unsigned int)` – publiczna, zmniejsza aktualne zdrowie o wartość daną argumentem uwzględniając specyfikę graczy.
 - `void`** `applyWinnerReward ()` – publiczna, aktualizuje dane gracza po wygranej walce.
 - `void`** `cure ()` – publiczna, przywraca aktualne zdrowie do maksymalnego zdrowia.
 - `void`** `printStats ()` wypisuje na standardowe wyjście parametry gracza.
 - `void`** `die ()` – niepubliczna, uśmiercająca gracza poprzez wyzerowanie aktualnego zdrowia.
- Decydująca rolę w walkach pełni cezars, implementowany w klasie `CAESAR_CLASS` spełniającej warunki:
 - Nie może dziedziczyć po klasie `PLAYER_CLASS`.
 - Ma dostęp do wszystkich pól klasy `PLAYER_CLASS`.
 - Implementuje wyłącznie metodę **`void`** `judgeDeathOfLife (PLAYER_CLASS*)` rozstrzygającą śmierć lub życie gracza przekazanego w argumentcie. Skazuje na śmierć każdego trzeciego podsądnego o ile ilość ataków toczonej walki była parzysta. Reguły podlegania osądowi opisane są dalej.
- Walki toczą się na arenie zaimplementowanej w klasie `ARENA_CLASS` spełniającej warunki:
 - Posiada prywatne pole typu `CAESAR_CLASS`.
 - Jedyny dopuszczalny konstruktor w argumentcie przyjmuje wskaźnik do obiektu klasy `CAESAR_CLASS`.
 - Jedyną metodą (poza konstruktorem) jest publiczna metoda **`void`** `fight (PLAYER_CLASS*, PLAYER_CLASS*)` implementująca opisane dalej walki.

Reguły walki

- Do walki nie dochodzi, jeżeli jeden z graczy jest martwy.
- Walkę zaczyna gracz z większą zwinnością uzyskaną metodą `getAbility`, zaś w przypadku równych zwinności gracz podany w pierwszym argumentcie. W dalszym opisie zaczynający gracz nazywany będzie *pierwszym graczem*, zaś przeciwnik nazywany będzie *drugim graczem*.
- Przed pierwszym atakiem wywoływane są metody `printStats` pierwszego gracza, a następnie drugiego gracza.
- Walka polega na przemiennej atakach, w następujących krokach:
 - Jeżeli pierwszy gracz żyje, to:
 - Atakuje drugiego gracza zadając obrażenia zgodnie z własną metodą `getDamage` stosowaną w argumentcie metody `takeDamage` drugiego gracza.

- Zastosowanie skutków ataku pierwszego gracza na zdrowie drugiego gracza.
- Wywołanie metody `printStats` drugiego gracza.
- Jeżeli drugi gracz żyje, to:
 - Atakuje pierwszego gracza zadając obrażenia zgodnie z własną metodą `getDamage` stosowaną w argumentcie metody `takeDamage` pierwszego gracza.
 - Zastosowanie skutków ataku drugiego gracza na zdrowie pierwszego gracza.
 - Wywołanie metody `printStats` pierwszego gracza.
- Jeżeli skutek ataku pozostałe zdrowie spadnie do wartości 0, atakowany gracz umiera z użyciem metody `die`.
- Walka jest przerywana, jeżeli:
 - U jednego z graczy pozostałe zdrowie spadnie poniżej 10.
 - W walce wykonano 40 ataków.
- Po zakończeniu ataków każdy żyjący gracz:
 - Podlega osądowi cezarsa.
 - Wyświetla swoje parametry.
- Zwycięzcą walki jest każdy gracz pozostający przy życiu po osądzie cezarsa.
- Każdy zwycięzca zwiększa swój atak i zwinność o 2 z użyciem metody `applyWinnerReward`.
- Każdy zwycięzca jest przywracany do pełni aktualnego zdrowia z użyciem metody `cure`.
- Na koniec walki wyświetlane są parametry najpierw pierwszego, a następnie drugiego gracza.

Klasy walczących

- Człowiek, implementowany w klasie `HUMAN_CLASS` spełniającej warunki:
 - Dziedziczy po klasie `PLAYER_CLASS`.
 - Pamięta:
 - Identyfikator typu `string` zadany w konstruktorze.
 - Maksymalne zdrowie wynoszące 200.
 - Aktualne zdrowie na początku wynoszące 200.
 - Atak na początku wynoszący 30.
 - Zwinność na początku istnienia wynoszącą 10.
 - Obronę typu **`unsigned int`** wynoszącą 10.
 - Metody używane w walce:
 - `getAgility` – zwraca aktualną zwinność.
 - `getDamage` – zwraca aktualny atak.
 - `takeDamage` – obniża aktualne życie o wartość argument pomniejszoną o sumę wartości obrony i wartości zwinności.
 - Metoda `printStats` wypisuje w jednej linii:
 - Dla żywego gracza oddzielone znakiem dwukropka:
 - Identyfikator
 - Maksymalne zdrowie.
 - Aktualne zdrowie.
 - Pozostałe zdrowie z następującym znakiem %.
 - Aktualny atak.
 - Aktualną zwinność.
 - Opór.
 - Dla martwego gracza identyfikator i po znaku dwukropka napis `R.I.P.`
- Bestia, implementowana w klasie `BEAST_CLASS` spełniającej warunki:
 - Dziedziczy po klasie `PLAYER_CLASS`.
 - Pamięta:
 - Identyfikator typu `string` zadany w konstruktorze.
 - Maksymalne zdrowie wynoszące 150.
 - Aktualne zdrowie na początku wynoszące 150.
 - Atak na początku wynoszący 40.
 - Zwinność na początku wynoszącą 20.
 - Metody używane w walce:
 - `getAgility` – zwraca aktualną zwinność
 - `getDamage` – zwraca aktualny atak, ale gdy pozostałe zdrowie spadnie poniżej 25 zwraca podwojony aktualny atak.
 - `takeDamage` obniża aktualne życie o wartość argumentu pomniejszoną o połowę zwinności zaokrągloną w dół.

- Metoda `printStats` wypisuje w jednej linii:
 - Dla żywego gracza oddzielone znakiem dwukropka:
 - Identyfikator.
 - Maksymalne zdrowie.
 - Aktualne zdrowie.
 - Pozostałe zdrowie z następującym znakiem %.
 - Aktualny atak.
 - Aktualną zwinność.
 - Dla martwego gracza identyfikator i po znaku dwukropka napis `R.I.P.`
- Berserker, implementowany w klasie `BERSERKER_CLASS` spełniającej warunki:
 - Dziedziczy zarówno po klasie `HUMAN_CLASS` oraz po klasie `BEAST_CLASS`.
 - W konstruktorze przewiduje dwa identyfikatory, kolejno dla składowych `HUMAN_CLASS` oraz `BEAST_CLASS`.
 - Pamięta:
 - Maksymalne zdrowie wynoszące 200.
 - Aktualne zdrowie początkowo wynoszące 200.
 - Atak początkowo wynoszący 35.
 - Zwinność początkowo wynoszącą 5.
 - Obronę wynoszącą 15.
 - Rozpoczynając walkę (metody `takeDamage`, `getDamage`, `getAbility`, `printStats`) zachowuje się jak człowiek (stosuje metody składowej `HUMAN_CLASS`), ale gdy pozostałe zdrowie spadanie poniżej 25 walczy jak bestia (z użyciem metod składowej `BEAST_CLASS`). Umierając wraca do postaci człowieka.
- Zespół, implementowany w klasie `SQUAD_CLASS` spełniającej warunki:
 - Dziedziczy wyłącznie po klasie `PLAYER_CLASS`.
 - Ma dostęp do każdej składowej klasy `PLAYER_CLASS`.
 - Przed walką tworzy zespół złożony z niepustej ilości różnych oraz żywych graczy wcześniej opisanych klas walczących, dodawanych wymagającą implementacji metodą:**`void`** `addPlayer (PLAYER_CLASS*)` z argumentem wskaźnika do obiektu dodawanego gracza.
 - Pamięta identyfikator typu `string` zadawany w konstruktorze.
 - Metody i parametry używane w walce:
 - `getAbility` zwraca najmniejszą zwinność spośród graczy zespołu.
 - `getDamage` zwraca sumę ataków członków zespołu
 - `takeDamage` powoduje dla każdego członka zespołu doznanie uszkodzenia równego ilorazowi ataku przeciwnik i ilości członków własnego zespołu zaokrągloną w dół i pomniejszoną o indywidualne wartości zwinności i obrony.
 - Pozostałe zdrowie zespołu jest równe największemu pozostałemu zdrowiu członków.
 - Ponadto:
 - Ginący członek jest z usuwany z zespołu.
 - Uśmiercenie przez cezarsa zabija wszystkich członków zespołu.
 - Metoda `printStats`:
 - Dla niepustego zespołu wypisuje w jednej linii oddzielone znakiem dwukropka:
 - Identyfikator.
 - Liczbę członków zespołu.
 - Pozostałe zdrowie z następującym znakiem %.
 - Aktualny atak.
 - Aktualną zwinność.
 - Następnie wywołuje metodę `printStats` dla każdego członka zespołu w kolejności wywołania metod `addPlayer`.
 - Dla pustego zespołu wypisuje identyfikator oraz po dwukropku napis `nemo`.

Dodatkowe wymagania

- Pierwsza linia kodu rozwiązania zawiera komentarz z imieniem i nazwiskiem autora.
- Jedynym dozwolonym do włączenia plikiem zewnętrznym jest `iostream`.
- Zabronione jest używanie własnych identyfikatorów rozpoczynających się znakiem podkreślenia.