



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE



# Esercitazione

Inheritance exercise

# Obiettivo

- Il progetto CLion fornito contiene classi e scheletri di classi relative al gioco in stile Rogue (<https://it.wikipedia.org/wiki/Roguelike>) della scorsa esercitazione.
- Scopo della presente esercitazione è:
  - Estendere le classi GameCharacter e Weapon, partendo da scheletri di classi derivate
    - Costruttori, metodi, attributi
  - Vedere come usare puntatori e riferimenti a classi base per usare oggetti di tipo derivato.

# Schema del codice

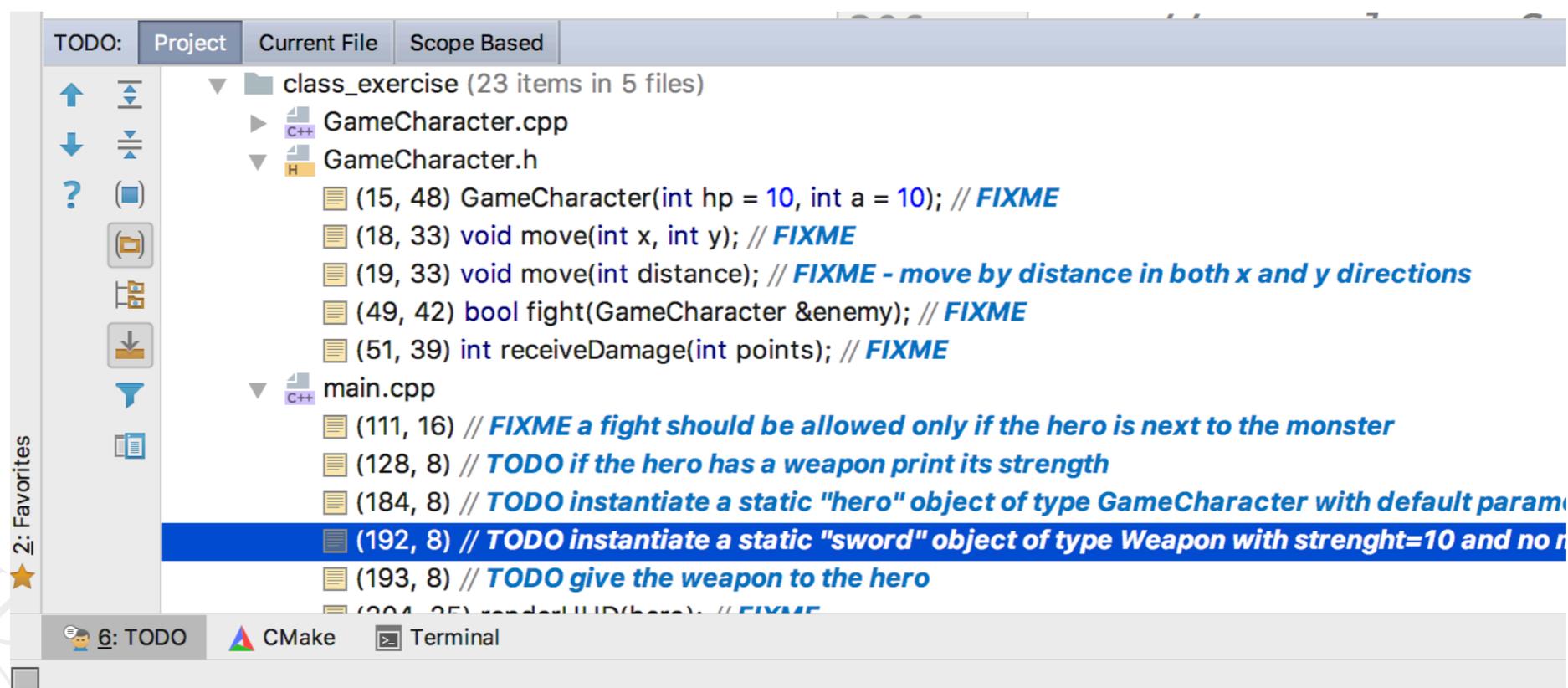
- Il programma è composto da 4 classi di partenza:
- Dungeon crea mappe casuali con stanze, corridoi, scale, porte, etc.
- Weapon rappresenta un arma con forza e magia
- GameCharacter rappresenta un personaggio del gioco, ed è composto con Weapon.
- Dice rappresenta un dado

# Schema del codice

- Lo schema del codice delle classi di base è lo stesso di quello della volta scorsa
- In questa esercitazione andremo ad estendere le classi `GameCharacter` e `Weapon` per aggiungere nuove funzionalità come due tipi di armi diverse (`Sword` e `Bow`), due tipi di personaggi per il giocatore (`Knit` e `Wizard`) e due mostri (`Orc` e `Skeleton`)
- Queste classi derivate hanno funzionalità diverse, modificando il comportamento di metodi di classi base, aggiungendo nuovi metodi ed attributi.

# Dove modificare il codice

- Le indicazioni precise sul codice da modificare sono fornite come commenti indicati con TODO e FIXME
- Per vedere tutti questi commenti selezionare la finestra TODO di CLion



# Dove modificare il codice

The screenshot shows the CLion IDE interface. The main editor displays the code in `main.cpp` with the following content:

```
185 // find a legal start position
186 int startX = 0;
187 int startY = 0;
188 setupCharacterCell(startX, startY, map);
189 hero.setPosX(startX);
190 hero.setPosY(startY);
191 // create a weapon and give it to hero
192 // TODO instantiate a static "sword" object of type Weapon with strenght=10 and
193 // TODO give the weapon to the hero
194 // create an enemy with a low grade armor
195 GameCharacter enemy(20, 2);
196 // find monster position not too far from hero position
197 startX += 5;
198 startY += 3;
199 setupCharacterCell(startX, startY, map);
200 enemy.setPosX(startX);
201 enemy.setPosY(startY);
202
203 // render
204 renderHUD(hero); // FIXME
205 renderGame(map, hero, enemy);
```

The TODO list at the bottom of the IDE shows the following items:

- (15, 48) `GameCharacter(int hp = 10, int a = 10);` // FIXME
- (18, 33) `void move(int x, int y);` // FIXME
- (19, 33) `void move(int distance);` // FIXME - move by distance in both x and y directions
- (49, 42) `bool fight(GameCharacter &enemy);` // FIXME
- (51, 39) `int receiveDamage(int points);` // FIXME
- (111, 16) // FIXME a fight should be allowed only if the hero is next to the monster
- (128, 8) // TODO if the hero has a weapon print its strength
- (184, 8) // TODO instantiate a static "hero" object of type GameCharacter with default parameters
- (192, 8) // TODO instantiate a static "sword" object of type Weapon with strenght=10 and no magic
- (193, 8) // TODO give the weapon to the hero
- (204, 25) `renderHUD(hero);` // FIXME

The status bar at the bottom indicates the current context is `class_exercise [D]`.

# Classe Weapon

- Questa è una classe base per Sword e Bow
- Il metodo più interessante è `use()` e ci si aspetta che debba essere implementato in modo diverso nelle classi derivate, per cui è stato marcato come `virtual`
- E' l'unico cambiamento. Si può usare come esempio per rendere classe base `GameCharacter`



# Classe Sword

- Estende Weapon, mostrando come fare override e come aggiungere attributi.
- Se la spada è in acciaio di Valiria fa più danni...
- Usarla come ispirazione per il lavoro sulla classe Bow



# Classe Bow

- Estende Weapon, rappresenta un arco, ovvero un'arma che ha un numero limitato di colpi (le frecce).
  - Dobbiamo inizializzare l'attributo del numero delle frecce e fare l'override del metodo use per fare in modo che si possa attaccare solo se si hanno frecce e si decrementi il numero di frecce ad ogni uso.
  - Si può riusare il metodo della classe base all'interno del metodo derivato. La sintassi per invocarlo è `Weapon::use()`

# Classe GameCharacter

- Questa è una classe base per Orc, Skeleton, Knight e Wizard
- Bisogna decidere quali metodi rendere metodi di cui fare override nelle classi derivate, per fare in modo che queste possano fornire varianti e usarli in modo polimorfico
  - Guardare i metodi selezionati come candidati.
  - Un metodo nuovo è `getCharacterSymbol()` che è usato nel rendering, usando una lettera diversa per ogni tipo di personaggio.

# Classe Orc e Knight

- Il lavoro da fare è molto simile.
- Quando si fa l'override di un metodo usare `override`, per essere sicuri di avere la signature corretta (es. non dimenticarsi un `const` di un metodo...)
- Per Orc l'implementazione di due metodi come `move` e `fight` è già fornita... basta fare la dichiarazione
- Ispirarsi a questo codice per l'implementazione dei metodi di Knight



# Classe Skeleton

- Il metodo `receiveDamage` è già implementato, basta dichiararlo
- Implementare solo il costruttore, gli altri metodi vengono semplicemente ereditati dalla classe base



# Classe Wizard

- Aggiunge un nuovo metodo (già implementato)
- Per uno dei metodi d cui fare override è già fornita l'implementazione, basta fare la dichiarazione



# Main

- Istanziare oggetti: armi, personaggi del giocatore e mostro.
- Notare come si usino puntatori e riferimenti a classe base per puntare/riferirsi a oggetti di tipo derivato
- Indipendentemente dal tipo di oggetto derivato a cui ci si riferisce/punta si invoca il metodo specifico della classe derivata.
- E' un esempio di **polimorfismo** !



# Classe Dice

- Questa classe non deve essere toccata





# Classe Dungeon

- Questa classe non deve essere toccata

