



UNIVERSITÀ
DEGLI STUDI
FIRENZE



Esercitazione

Adapter exercise

Obiettivo

- Il progetto CLion fornito contiene classi e scheletri di classi relative ad una componente di rendering grafico di un gioco.
- Scopo della presente esercitazione è:
 - Implementare un design pattern Adapter nelle versioni class ed object
 - Scrivere un'iterazione sul container di oggetti e sfruttare il polimorfismo
 - Riscrivere il codice per passare da raw pointer a smart pointer

Schema del codice

- Il programma è composto da 4 classi di partenza:
- Shape rappresenta una forma; è una classe astratta che fornisce l'interfaccia per disegnare e scalare oggetti grafici.
- Sprite e Tile, derivate da Shape, rappresentano elementi grafici del gioco come i personaggi da disegnare e le caselle della mappa.
- Text rappresenta del testo, non eredita da Shape, non ha il metodo draw di Shape e il resize ha argomenti diversi.

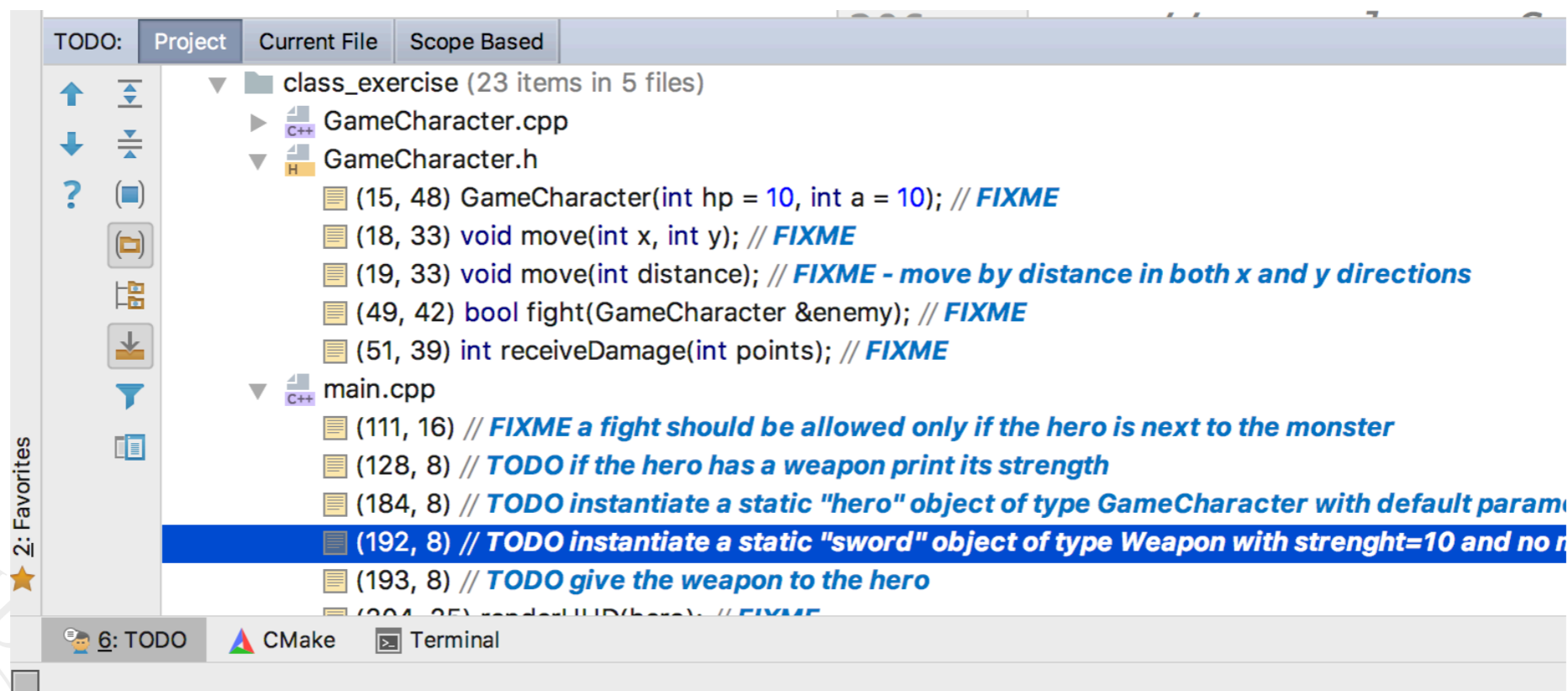
Schema del codice

- In questa esercitazione andremo a creare un `TextShapeAdapter` così da poter inserire scritte nel vettore di oggetti grafici in `main`
- Scriveremo un'iterazione sul container di oggetti e sfrutteremo il polimorfismo per ridimensionare e disegnare tutti gli oggetti
- Una volta completate le due parti precedenti riscriveremo il codice per usare smart pointer a scelta



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 TextShapeAdapter

- Implementare sia un Class che un Object Adapter che consenta di inserire oggetti che racchiudano testo nel vettore di puntatori a shape





Main

- Iterare su tutti gli elementi grafici, ridimensionarli di 1.2 volte e disegnarli.
- Sostituire i raw pointer con smart pointer.
Trasformare il vettore in vettore di smart pointer



unique_ptr e shared_ptr

- Uno `unique_ptr` non può essere copiato. Per inserirlo in un container STL si può:
 - Usare `emplace_back` di un oggetto allocato dinamicamente (verrà gestito da uno `unique_ptr` creato automaticamente)
 - Usare `push_back` di uno smart pointer creato al momento della chiamata del `push_back` (ovvero senza nome)
 - Usare il `push_back` passando uno `unique_ptr` con nome tramite `std::move`:

```
std::vector<std::unique_ptr<Obj>> myVector;  
std::unique_ptr<Obj> fooPtr(new Obj);  
myVector.push_back(std::move(fooPtr));
```

- Uno `shared_ptr` può essere copiato per cui l'inserimento in un container STL non richiede nessun particolare accorgimento.