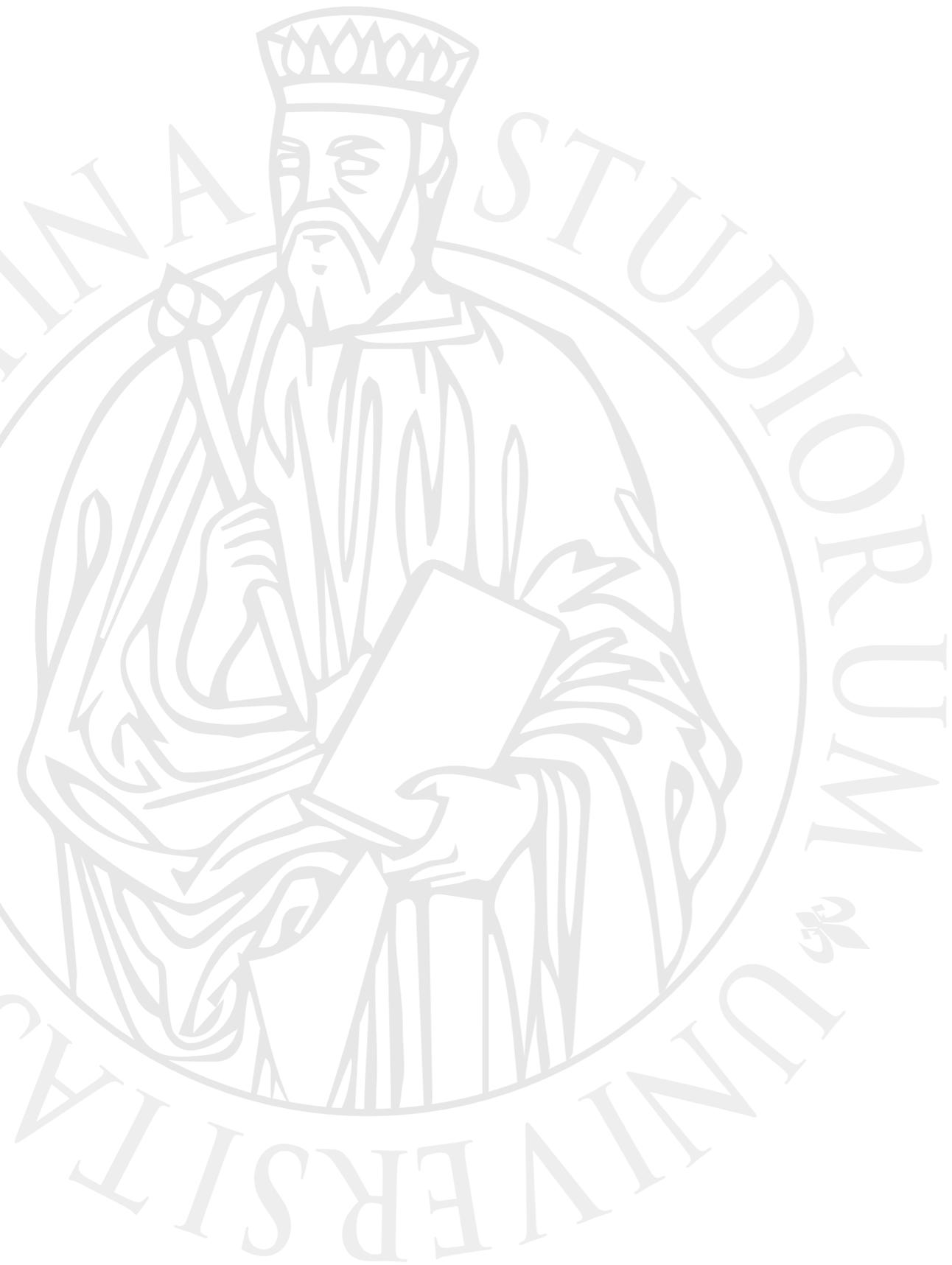




Esercitazione

Factory method/Abstract factory
exercise



E11

Factory Method

Obiettivo

- Il progetto CLion fornito contiene classi e scheletri di classi relative ad un gioco (personaggi).
- Scopo della presente esercitazione è:
 - Implementare un design pattern Factory Method per la creazione dei personaggi del gioco
 - Ovvero incapsulare il problema della creazione dei personaggi così da facilitare l'introduzione di nuovi tipi di personaggi

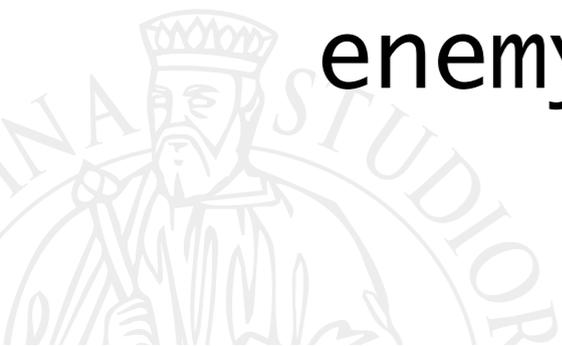
Schema del codice

- Il programma è composto da tre classi principali:
 - `GameCharacter` rappresenta un personaggio del gioco, con due metodi principali. È una classe astratta che fornisce l'interfaccia dei personaggi.
 - `Knight` è una classe concreta che estende `GameCharacter`
 - `Wizard` è una classe concreta che estende `GameCharacter`



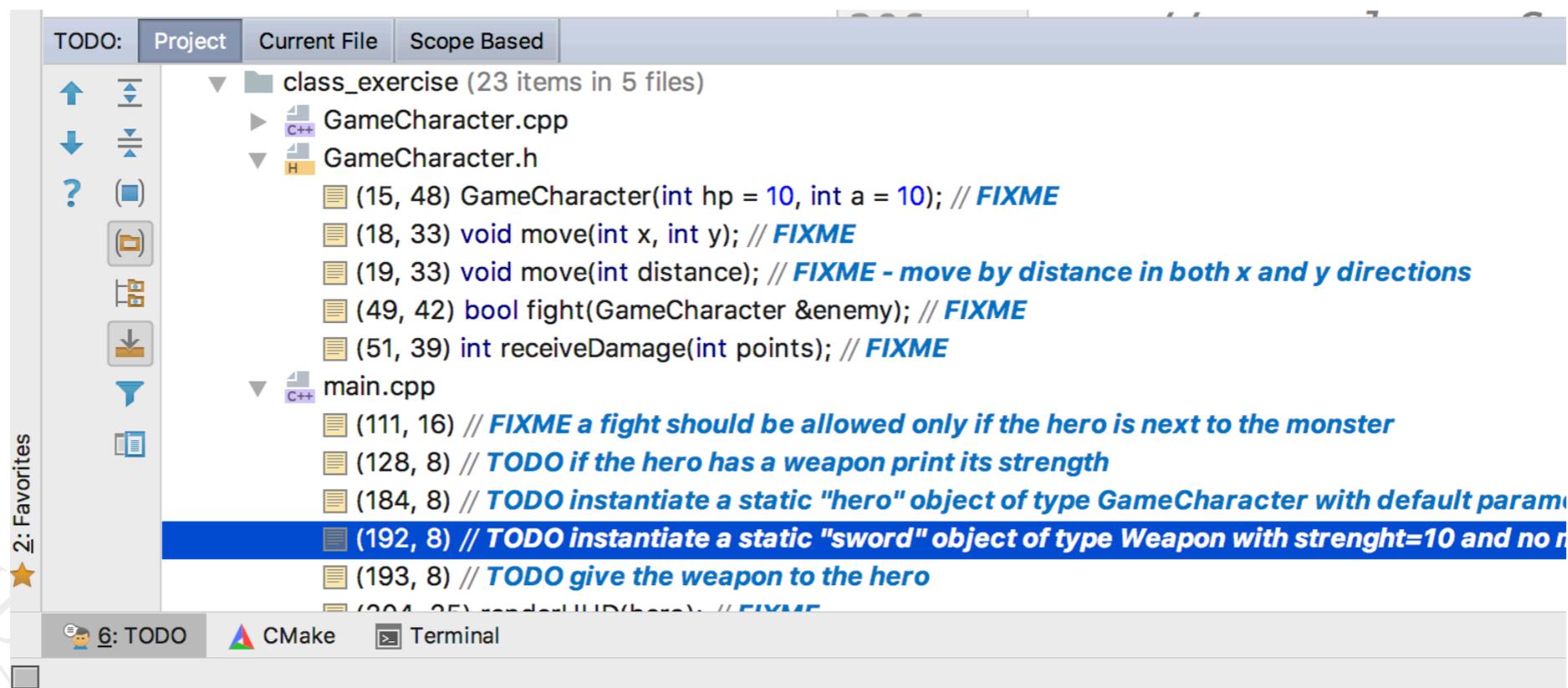
Schema del codice

- C'è una quarta classe - `CharacterFactory` - in cui è necessario implementare un metodo per la creazione di personaggi concreti sulla base di un parametro.
- Dopo aver creato un personaggio del gioco è necessario impostare la bitmap relativa, per poterlo disegnare.
- `Main` dove creare due personaggi (hero ed enemy).



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]`.

CharacterFactory

- Implementare tutte le modifiche necessarie per fare in modo che fornisca Knight e Wizard sulla base di un parametro
 - Si può usare il tipo enumerato presente nell header
 - La factory ha bisogno di essere inizializzata: nel costruttore si simula la necessità di caricare un file bitmap con le immagini da associare ad ogni personaggio
 - Un apposito metodo rende la bitmap di un personaggio, per poterla impostare nell'oggetto creato



CharacterFactory

- Implementare tutte le modifiche in modo che fornisca Knight e un parametro
- Si può usare il tipo enumerato
- La factory ha bisogno di un costruttore si simula la ne bitmap con le immagini del personaggio
- Un apposito metodo rende la bitmap di un personaggio, per poterla impostare nell'oggetto creato



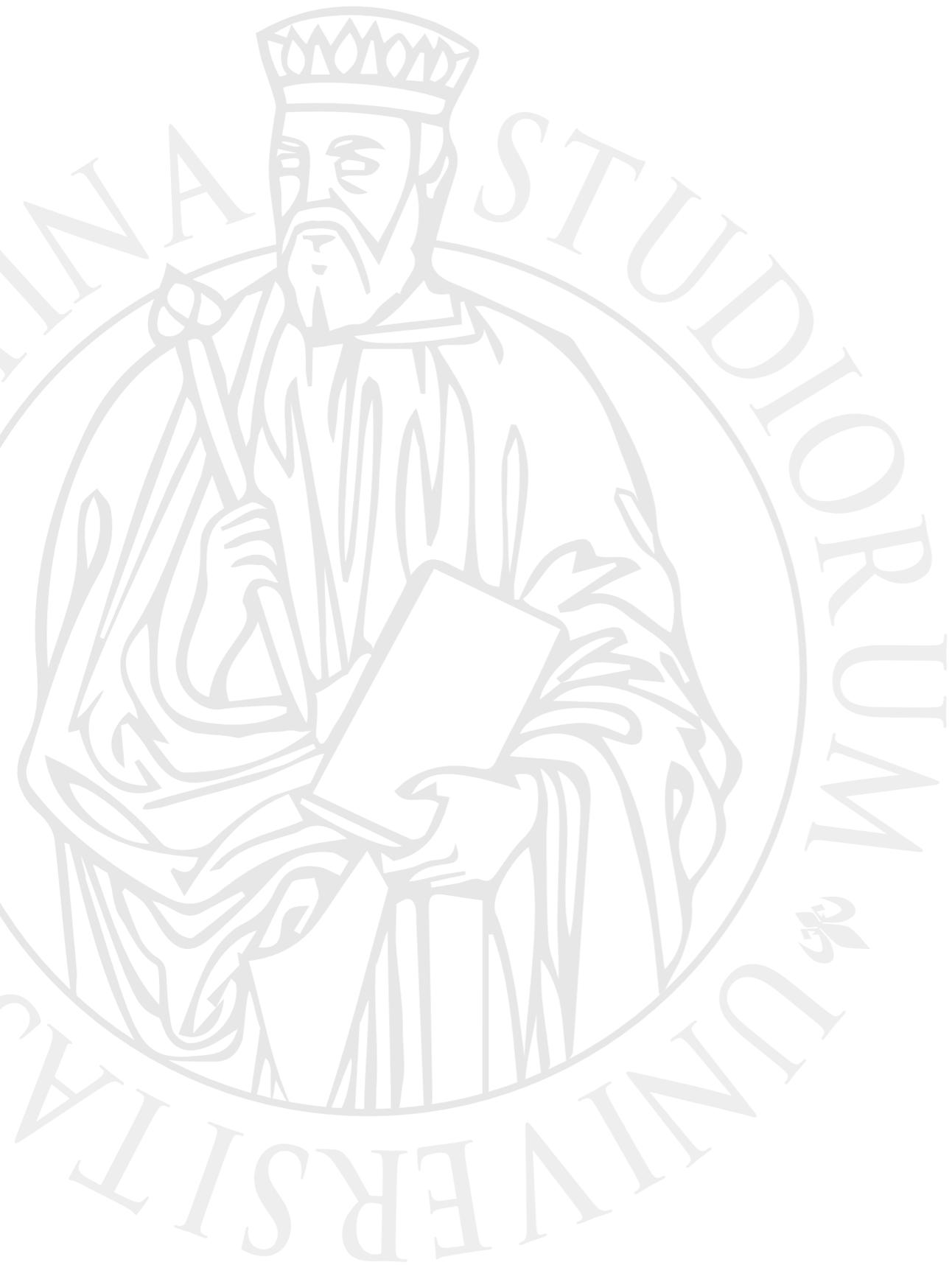
Smart pointer

- Implementare il sistema per usare smart pointers
 - Si deve implementare sia nella factory che nel main dove si usano i prodotti della factory





UNIVERSITÀ
DEGLI STUDI
FIRENZE



E12

Abstract

Factory

Obiettivo

- Il progetto CLion fornito contiene classi e scheletri di classi relative ad una GUI di un programma multiplatforma (e.g. iOS e Windows).
- Scopo della presente esercitazione è:
 - Implementare un design pattern Abstract Factory per la creazione di widget grafici del programma
 - Ovvero incapsulare il problema della creazione di widget tra loro coordinati (bottone e finestra) così da facilitare l'introduzione di nuovi tipi (es. altri S.O.)

Schema del codice

- Il programma è composto da diverse classi principali:
 - `Widget` rappresenta un widget grafico con un metodo principale che lo disegna. È una classe base astratta da cui discendono tutti i widget.
 - `Button` è una classe astratta che estende `Widget` e aggiunge un metodo specifico dei bottoni
 - `Window` è una classe astratta che estende `Widget` e aggiunge un metodo specifico delle finestre

Schema del codice

- Da `Button` e `Widget` si estendono classi specifiche per i sistemi operativi `iOS` e `Windows`
- `WidgetFactory` è la classe dove implementare l'interfaccia dell'`Abstract Factory`
- `main` è il corpo del programma principale dove dobbiamo istanziare un bottone ed una finestra sulla base di un booleano.
 - Dobbiamo poi eseguire metodi di `Widget` e delle classi derivate su questi oggetti



Factory concrete

- Implementare le factory concrete necessarie
- Implementare le factory per usare smart pointer al posto dei raw pointer

