

HOMEWORK B - 2025

Polimorfismo ed
Estensione di Classi

Esercizio 0

- ✓ Chi non avesse concluso la scrittura dei test per il precedente homework, lo faccia in questo homework, prima di fare le modifiche al codice, raggiungendo una situazione iniziale in cui numerosi test di unità hanno successo e confermano il corretto funzionamento a tempo di esecuzione del codice sviluppato sinora

Esercizio 1

- Implementare tutte le ristrutturazioni discusse nella dispensa sul polimorfismo
- Scrivere quindi i test della classe **ComandoVai**
- Implementare le classi corrispondenti a tutti i comandi previsti sino ad ora nel gioco
 - «aiuto», «fine», «prendi», «posa»; «guarda»
 - aggiungere la classe **ComandoGuarda**: «guarda» stampa le informazioni sulla stanza corrente e sullo stato della partita
 - aggiungere la classe **ComandoNonValido**
 - scrivere i test per le classi **ComandoPosa**, **ComandoPrendi**
- N.B. bisogna rifattorizzare *anche* i vecchi test mantenendo sempre i test ed il codice principale allineati

Esercizio 2

- Introdurre l'interfaccia ***FabbricaDiComandi*** e la classe **FabbricaDiComandiFisarmonica**
- Scrivere i test di unità su questa classe concreta ma limitarsi alla sola verifica del corretto riconoscimento dei comandi
 - *Suggerimento:* per scrivere questi test, aggiungere i metodi **getNome()** e **getParametro()** all'interfaccia ***Comando***
 - ✓ N.B.: evitare invece di soffermarsi sull'istanziamento della corretta classe concreta associata a ciascun comando perché ancora non abbiamo gli strumenti opportuni

Esercizio 2 (continua)

- Il progetto sta crescendo: riorganizziamo meglio le classi introducendo anche il package
 - **`it.uniroma3.diadia.comandi`**
ove collocare i comandi e la fabbrica

Esercizio 3

- In HW1 (esercizio 5) abbiamo rifattorizzato il codice affinché l'uso diretto di **System.out** e **System.in** fosse prima evitato eppoi “centralizzato” in **IOConsole**
- ✓ Assicurarsi di aver svolto l'esercizio 5 del precedente homework

Esercizio 3 (continua)

- Completiamo il processo di disaccoppiamento dall'I/O tramite l'introduzione di un apposita interfaccia denominata **IO** (presentata nella prossima slide>>) che astragga **IOConsole**:
 - Creare l'interfaccia e posizionarla nel package **it.uniroma3.diadia**
 - Cambiare **IOConsole** affinché la implementi
 - Cambiare tutto il codice affinché ogni riferimento ad **IOConsole** sia rimpiazzato da un (meno vincolante) riferimento tipato **IO**

Esercizio 3: Interfaccia IO

```
package it.uniroma3.diadia;  
  
public interface IO {  
    public void mostraMessaggio(String messaggio);  
    public String leggiRiga();  
}
```

- ✓ L'unica istanza (della sua implementazione) deve essere creata dal metodo **DiaDia.main()**

```
public class DiaDia {  
    ...  
    public static void main(String[] argc) {  
        IO io = new IOConsole();  
        DiaDia gioco = new DiaDia(io);  
        gioco.gioca();  
    }  
    ...}
```


Esercizio 3: Vantaggi

- ✓ In futuro sarà più facile cambiare l'implementazione dell'interfaccia **IO** creata nel metodo **main()** ed ipotizzare forme di interazione diverse da quelle sinora usate. Ad esempio:
 - una GUI (>>): dotando il gioco di una vera e propria parte grafica
 - un sistema automatico che simuli delle partite (>> vedi esercizio 9)

Esercizio 4

- Le recenti modifiche cambiano l'implementazione del gioco senza modificarne affatto il comportamento
- Subito dopo averle effettuate, verificare mediante i test sviluppati in questo e negli homework precedenti la correttezza del codice per confermare che non si siano introdotti nuovi errori
 - ✓ ovvero che non ci sia *regressione*
- In presenza di test che cominciano a fallire solo ora, mentre in precedenza avevano successo, utilizzare i fallimenti e la diagnostica per correggere gli errori
 - ✓ **Cominciando sempre dai test più semplici**
 - ✓ Se non si riesce subito a trovare i bug, aggiungere altri test-case sino a renderne palesi le cause

Esercizio 5

- Implementare ed introdurre nel gioco la «stanza magica», come descritto nelle dispense sull'estensione di classi
- Realizzare due distinte versioni di **Stanza**
 - **Stanza**
 - versione con campi privati: rispetta il principio dell'*information hiding* facendo utilizzare da parte delle classi estese solo la parte pubblica della classe base
 - **StanzaProtected**: campi protetti
- Corrispondentemente, realizzare anche due versioni della classe derivata:
 - **StanzaMagica**: estende **Stanza**
 - **StanzaMagicaProtected**: estende **StanzaProtected**

TDD (Facoltativo)

- ✓ N.B. È perfettamente lecito e consigliabile fare l'esercizio 8 anche prima degli esercizi 6&7

Esercizi 6-7

- Vogliamo introdurre nel gioco due ulteriori stanze particolari
 - La «stanza buia»: se nella stanza non è presente un attrezzo con un nome particolare (ad esempio "lanterna") il metodo **getDescrizione()** di una stanza buia ritorna la stringa *"qui c'è un buio pesto"*
 - La «stanza bloccata»: una delle direzioni della stanza non può essere seguita a meno che nella stanza non sia presente un oggetto con un nome particolare (ad esempio "passepartout")
- Creare le classi **StanzaBuia** e **StanzaBloccata** come estensioni della classe **Stanza**

Esercizio 6: Stanza Buia

- La classe **StanzaBuia** deve avere una variabile di istanza di tipo **String**: memorizza il nome dell'attrezzo che consente di avere la descrizione completa della stanza
- Il metodo **getDescrizione()** va sovrascritto affinché produca la descrizione usuale o la stringa "**qui c'è buio pesto**" a seconda che nella stanza ci sia o meno l'attrezzo richiesto per "vedere"
- Il nome dell'attrezzo necessario viene impostato attraverso il costruttore

Esercizio 7: Stanza Bloccata

- La classe **StanzaBloccata** deve avere due variabili di istanza di tipo **String** per memorizzare:
 - il nome della direzione bloccata
 - il nome dell'attrezzo che consente di sbloccare la direzione bloccata
- Il metodo **getStanzaAdiacente(String dir)** va riscritto (override)
 - se nella stanza non è presente l'attrezzo sbloccante, il metodo ritorna un riferimento alla stanza corrente
 - altrimenti ha l'usuale comportamento (ritorna la stanza corrispondente all'uscita specificata)

Esercizio 7: Stanza Bloccata (continua)

- Dentro la classe **StanzaBloccata** riscrivere anche il metodo **getDescrizione()** affinché produca una descrizione opportuna
- Anche in questo caso il nome dell'attrezzo sbloccante (ad es. 'piedediporco') e il nome della direzione bloccata vanno impostati attraverso il costruttore

Esercizio 8

- Scrivere i test per le classi **StanzaBuia** e **StanzaBloccata** implementate secondo le indicazioni espresse nelle trasparenze precedenti
- ✓ N.B. È perfettamente lecito e consigliabile fare questo esercizio anche prima degli esercizi 6&7
- Suggerimenti: cercare di mantenere i test «unitari»
 - ✓ non scomodare intere «Partite» solo per testare la logica di particolari stanze
 - ✓ i labirinti minimali per questi test sono «monocali» e/o «bilocali»?

Controlli (Manuali)

Prima della Consegna

- Assicurarsi che l'esercizio 3 di questo homework e l'esercizio 5 del precedente siano stati svolti correttamente. Nello specifico verificare che:
 - Non ci siano chiamate di metodo tramite **System.in** o **System.out** al di fuori del corpo della classe **IOConsole**
 - **IO** e **IOConsole** si trovino nel package **it.uniroma3.diadia**
 - **IOConsole** implementi **IO**
 - **IOConsole** sia correttamente istanziato una sola volta ed iniettato dal metodo **main()** nel costruttore di **DiaDia**
 - ✓ In tutto il codice principale i riferimenti siano tipati **IO** e *mai* **IOConsole**
 - ✓ Se si svolge anche l'esercizio 9 (>>) è normale avere riferimenti tipati **IOSimulator** nei corrispondenti test

Esercizio 9

- Scrivere una nuova classe **IOSimulator** (nel package **it.uniroma3.diadia**) che implementi l'interfaccia **IO**
- I metodi **mostraMessaggio()** e **leggiRiga()** dovranno rispettivamente scrivere / leggere i messaggi che sono scritti a video / letti da tastiera
 - Possono essere letti/conservati in array definiti come variabili di istanza
 - Il metodo **leggiRiga()** consentirà di “iniettare” le righe che desideriamo far figurare come istruzioni (di solito immesse dall'utente)
 - Il metodo **mostraMessaggio()** consentirà di conoscere i messaggi stampati durante la partita (a supporto di eventuali asserzioni)

Esercizio 9 (continua)

- Sviluppare dei test automatici che sappiano simulare *inter* partite senza bisogno di input manuale dell'utente, troppo oneroso da ottenere
 - i test devono “scrivere” i comandi e “leggere” i messaggi stampati rimpiazzando l'interazione “manuale” con i servizi offerti da **IOSimulator**
 - ideale per ampliare gli scenari del testing automatico sino a coprire *inter* partite
 - ✓ N.B. Non si tratta più di unit-test
- Ma per cominciare a capirne il funzionamento li scriviamo con JUnit
 - esistono framework dedicati alla scrittura di questa tipologia di test, ovvero: test di accettazione (>>)
- Lo unit-testing va esattamente nella direzione opposta: si verifica il corretto funzionamento di piccoli frammenti di codice, NON di *inter* partite!

Esercizio 9 (Test di Accettazione)

- ✓ Una nuova *tipologia di Test*
 - Chiamati *test di Accettazione*
 - “Accoppiati” all’I/O
 - alle istruzioni immesse
 - alle stampe effettuate
- ✓ NON sono test di unità

TERMINI E MODALITA' DI CONSEGNA

- ATTENZIONE:
 - Senza l'invio di questo homework, *non* sarà possibile continuare il percorso HQ di questo anno accademico
- Consegna tramite GitHub descritta di seguito (>>)
 - Creare un repository Github
 - Caricare il sorgente sviluppato
 - Creare una release di nome **versione.B**
 - Inviare una email contenente un link alla release del codice

MODALITA' DI CONSEGNA

- Per la consegna utilizzare poo.roma3@gmail.com
 - Per consegnare usare questo indirizzo di email!
Gestita automaticamente.
- Non inviare mail di richiesta di conferma di avvenuta ricezione, non saranno gestite
 - Eventuali problematiche saranno gestite successivamente, quando e se *veramente* serve
- Consegne multiple da evitare per quanto possibile
 - ✓ Inutile: fa fede la data della release non dell'email
 - ✓ Ma solo le email verranno gestite automaticamente per raccogliere le matricole di chi ha consegnato
- ATTENZIONE:
 - NON allegare direttamente archivi .jar, .zip, .tar.gz
 - ✓ Per motivi di sicurezza l'email non verrebbe consegnata
 - Aggiungere nel corpo del messaggio:
 - Il link al repository github (>>)
 - Descrizione degli eventuali malfunzionamenti noti, ma non risolti

TERMINI E MODALITA' DI CONSEGNA

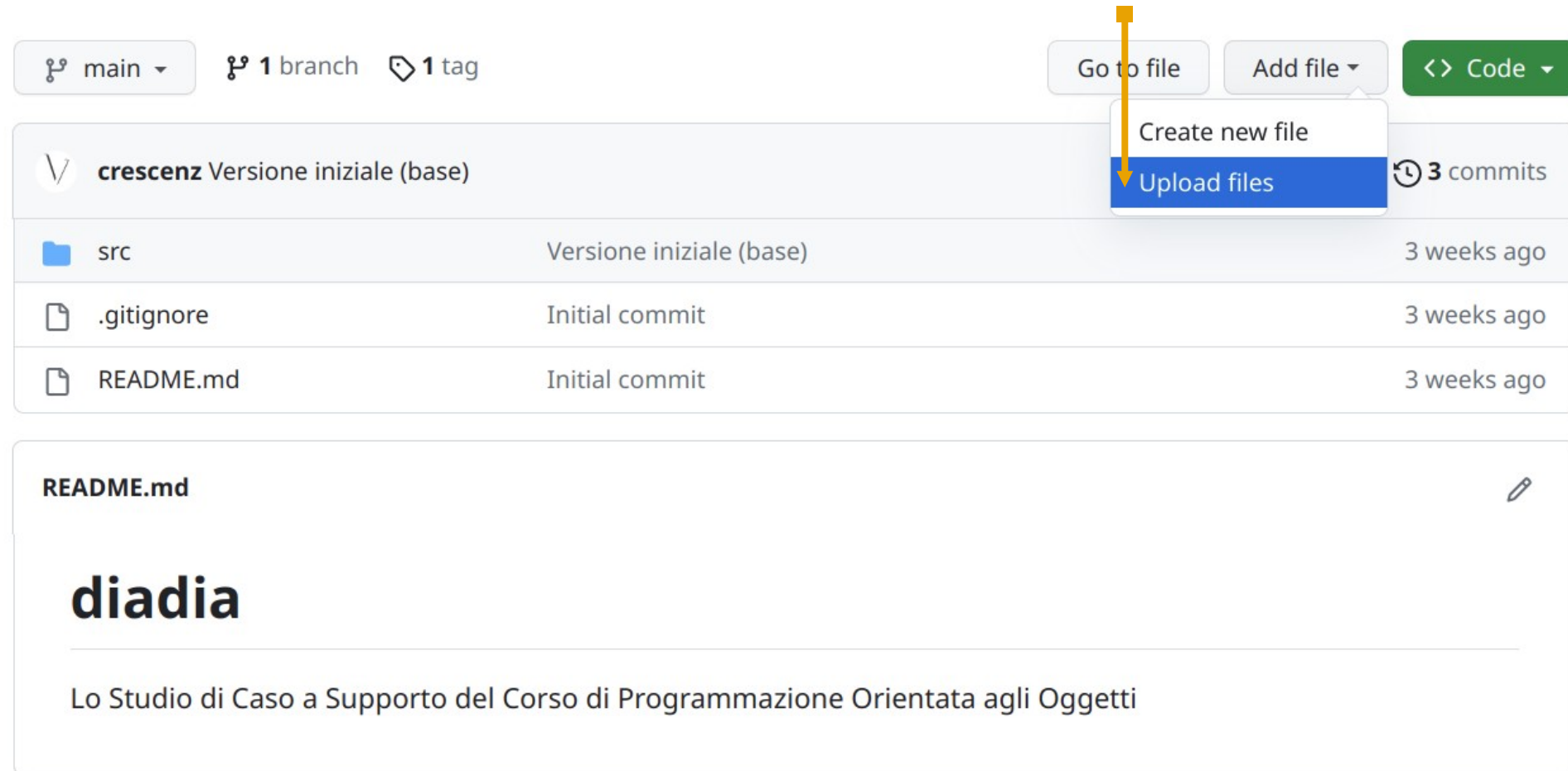
La soluzione deve essere inviata al docente entro le 21:00 di **domenica 11 maggio 2025** come segue:

- Svolgere in gruppi di esattamente 2 persone
- L'oggetto (subject) *DEVE* iniziare con la stringa **[2025-HOMEWORKB]** seguita dalle matricole

Ad es.: **[2025-HOMEWORKB] 612345 654321**

Consegna con GitHub (1)

- Nella stessa pagina della repository github già creata per la consegna del primo homework utilizzare: upload files



The screenshot shows a GitHub repository page for a user named 'crescenz'. The repository is on the 'main' branch and has 1 branch and 1 tag. The repository name is 'crescenz' and the description is 'Versione iniziale (base)'. There are 3 commits. The file list shows three files: 'src' (Versione iniziale (base), 3 weeks ago), '.gitignore' (Initial commit, 3 weeks ago), and 'README.md' (Initial commit, 3 weeks ago). The 'Add file' dropdown menu is open, showing 'Create new file' and 'Upload files' (highlighted with a blue bar and a yellow arrow pointing to it). Below the file list, the 'README.md' file is open, showing the text 'diadia' and 'Lo Studio di Caso a Supporto del Corso di Programmazione Orientata agli Oggetti'.

main 1 branch 1 tag

Go to file Add file > Code >

Create new file
Upload files

crescenz Versione iniziale (base) 3 commits

| | | |
|------------|--------------------------|-------------|
| src | Versione iniziale (base) | 3 weeks ago |
| .gitignore | Initial commit | 3 weeks ago |
| README.md | Initial commit | 3 weeks ago |

README.md

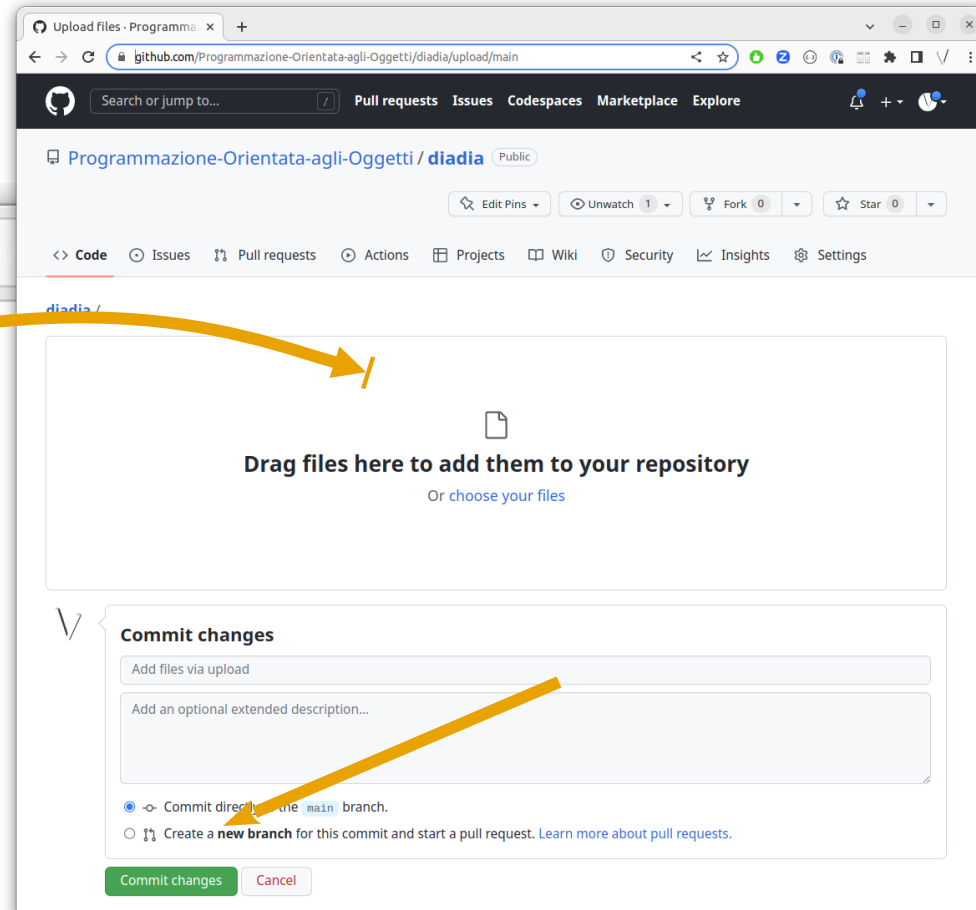
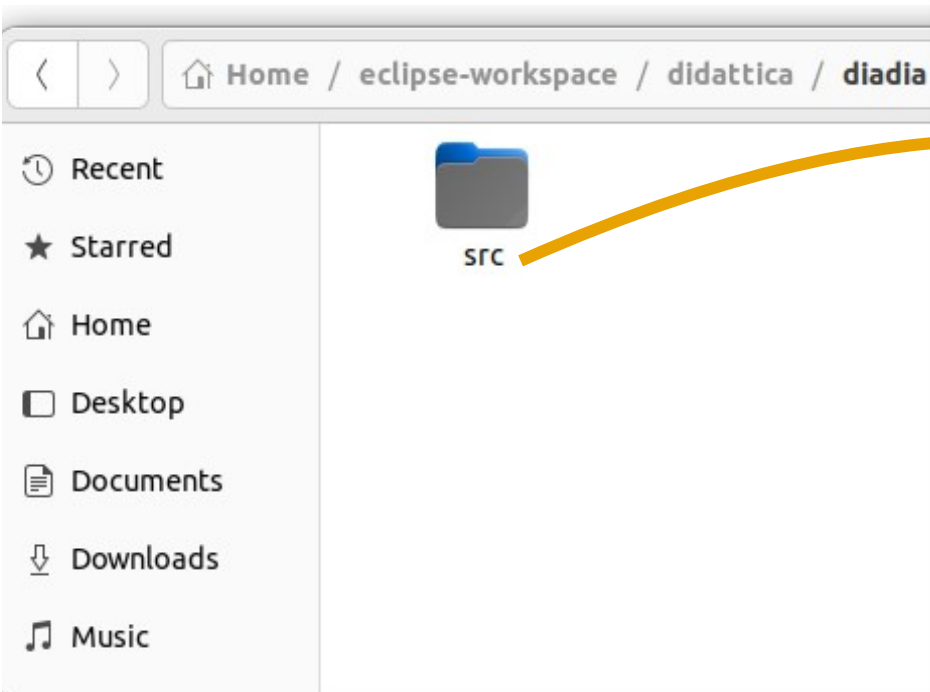
diadia

Lo Studio di Caso a Supporto del Corso di Programmazione Orientata agli Oggetti

2025 - HOMEWORK - 612345 - 654321

Consegna con GitHub (2)

- Navigare fino alla directory del vostro progetto locale
 - ad es. `/eclipse-workspace/didattica/diadia/`
 - Trascinare *tutta* la cartella `src` dentro il git repository già creato
 - Successivamente ripetere l'operazione con la cartella `test`
- Quindi salviamo le modifiche
 - tasto verde *Commit changes*



Consegna con GitHub (3)

- Caricare *tutte* le modifiche rispetto al precedente homework
- Creare una nuova *release*
 - Selezionare <<create new release>> a dx nella pagina

The screenshot shows a GitHub repository named 'crescenz'. At the top, there are buttons for 'main', '1 branch', and '1 tag'. Below this is a table of files: 'src' (Versione iniziale (base)), '.gitignore' (Initial commit), and 'README.md' (Initial commit). The 'README.md' file is selected, showing its content which includes the title 'diadia' and a description 'Lo Studio di Caso a Supporto del Corso di Programmazione Orientata agli Oggetti'. On the right side, there is a sidebar with 'About' (Lo Studio di Caso a Supporto del Corso di Programmazione Orientata agli Oggetti), 'Releases' (1 tags, Create a new release), and 'About' (Lo Studio di Caso a Supporto del Corso di Programmazione Orientata agli Oggetti). A yellow arrow points from the text 'Selezionare <<create new release>> a dx nella pagina' to the 'Create a new release' link in the 'Releases' section.

- Scegliere il cosiddetto *tag* della release

- Questa volta usare **versione.B**
- Come nome della release inserire
- 2025-HOMEWORK**B** <<matricola1>> <<matricola2>>

– Ad es. 2025-HOMEWORKB 612345 654321

- Nella descrizione scrivere eventuali malfunzionamenti noti ma non risolti