

# **HOMEWORK 4**

---

Classi Astratte  
Riflessione  
Eccezioni e I/O  
Tipi Enumerativi

# Esercizio 0 (prerequisito)

- Chi non avesse concluso la scrittura dei test, lo faccia in questo homework, prima di fare le modifiche al codice in risposta agli esercizi che lo compongono

# Esercizio 1

- Scrivere una classe astratta **AbstractComando** per eliminare le implementazioni “vuote” dei metodi **setParametro()** dalle classi concrete che implementano l'interfaccia **Comando** (in particolare dalle classi che modellano comandi privi di parametri come ad es. **ComandoGuarda**, **ComandoAiuto**)
- Scrivere nuovi test per la nuova classe astratta e rifattorizzare i test della gerarchia **Comando** già sviluppati durante lo svolgimento dei precedenti homework

# Esercizio 2

- Scrivere la classe astratta **AbstractPersonaggio** e le classi concrete che la estendono
  - **Strega**
  - **Mago**
  - **Cane**
- Scrivere le classi **ComandoSaluta** e **ComandoInteragisci** che modellano i comandi attraverso i quali il giocatore può rispettivamente salutare e interagire con un personaggio
- Queste modifiche sono descritte anche nelle trasparenze reperibili nella pagina del materiale didattico del corso:  
**POO-classi-astratte-enum**

# Esercizio 3

- Modificare la classe astratta **AbstractPersonaggio** introducendo il metodo astratto:

```
public String riceviRegalo(Attrezzo attrezzo, Partita partita)
```

- Scrivere la classe **ComandoRegala**, attraverso la quale il giocatore può regalare un attrezzo al personaggio presente nella stanza
  - ✓ in una stanza può trovarsi un solo personaggio; affinché un attrezzo possa essere effettivamente regalato il parametro del comando *regala* deve essere il nome di uno degli attrezzi presenti nella borsa

# Esercizio 3 (cont.)

- Nelle classi **Cane**, **Strega**, **Mago** implementare il metodo astratto:

```
public String riceviRegalo(Attrezzo attrezzo)
```

- un cane riceve un regalo, se questo è il suo cibo preferito lo accetta, e butta a terra un attrezzo, altrimenti non lo accetta e lascia cadere il regalo nella stanza
  - una strega riceve un regalo, che trattiene scoppiando a ridere
  - un mago riceve un regalo, gli dimezza il peso e lo lascia cadere nella stanza
- La stringa restituita rappresenta il messaggio che deve essere prodotto dal comando quando eseguito (analogamente al comando *interagisci*)

# Esercizio 4

- Riorganizziamo la gestione dell'I/O, disaccoppiando *prima* i comandi dall'uso diretto (tramite stampe) di **System.io**, operando come segue:
- togliamo ai comandi la responsabilità di stampare messaggi
  - anziché stampare a video, i comandi ritornano un messaggio in una stringa: modifichiamo il tipo di ritorno del metodo **esegui()** di Comando da **void** a **String**
  - dopo l'esecuzione di ogni comando, la classe **Gioco** (metodo **processaIstruzione()**) stampa il valore restituito dal metodo **esegui()**
- quindi* l'intero gioco dall'uso diretto di **System.io**, vedi esercizio seguente (>>)

# Esercizio 5

- Disaccoppiamo tutto l'I/O del gioco, operando nel seguente modo:

- Introdurre l'interface

```
public interface InterfacciaUtente {  
    public void mostraMessaggio(String messaggio);  
    public String prendiIstruzione();  
}
```

- Scrivere la classe **InterfacciaUtenteConsole** che implementa **InterfacciaUtente** usando lo standard input (**System.in**) e lo standard output (**System.out**) per interagire con l'utente
- Modificare il codice della classe **Gioco** affinché deleghi alla classe **InterfacciaUtenteConsole** la gestione dell'I/O (la classe **Gioco** avrà una variabile di istanza di tipo **InterfacciaUtente** opportunamente inizializzata nel costruttore)



# Esercizio 6

- Introdurre ed utilizzare la fabbrica di comandi basata sulla riflessione (vedi slide sulla *riflessione*) per la creazione degli oggetti **Comando**
- Ricontrollare tutto per assicurarsi che finalmente l'elenco dei comandi disponibili nel gioco sia effettivamente specificato una sola volta nel codice
  - ✓ A riprova, basta controllare quali modifiche sono necessarie per aggiungere un nuovo comando e quindi renderlo *perfettamente* funzionante ed integrato con il resto del gioco

# Esercizio 7

- Modificare la classe **Labirinto** affinché il labirinto venga caricato da file utilizzando la classe **CaricatoreLabirinto**
  - ✓ fornita nella pagina del materiale didattico del sito del corso
- Scrivere dei test sulla classe per individuare e correggere gli errori di **CaricatoreLabirinto**

## *Suggerimenti:*

- ✓ per favorire la leggibilità dei test ed il loro autocontenimento, e per non vincolare i test all'effettiva presenza di file sul disco, fare uso di fixture specificate tramite stringhe direttamente nei test ed utilizzare **StringReader** per farle leggere dal caricatore
- ✓ Scrivere diversi test-case su fixture di complessità crescenti: labirinto «monolocale», «bilocale», ecc.

# Esercizio 8

- Modificare la classe

**CaricatoreLabirinto** affinché sia possibile caricare anche personaggi, stanze chiuse, stanze buie ecc. ecc.

✓ *Suggerimento:* serve la riflessione

# Esercizio 9 (facoltativo)

- Modificare l'applicazione affinché la specifica delle costanti non sia cablata nel codice ma sia esternalizzata in un opportuno file di properties **diadia.properties** da distribuire assieme al codice stesso. Ad es.
  - il numero di CFU iniziali
  - il peso max della borsa
- Esportare l'applicativo in formato .jar e verificarne il funzionamento in un ambiente diverso da quello di sviluppo nonostante la dipendenza verso risorse aggiuntive rispetto al codice (ad es. il file **diadia.properties**)
  - ✓ *Suggerimento:* non cablare nel codice il percorso fisico del file di properties, ma *solo* il suo nome logico

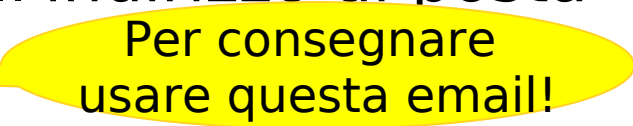
# Esercizio 10

- Modificare l'applicazione affinché siano utilizzati i tipi enumerativi laddove ancora restistono dei concetti di primo ordine per il gioco che siano lascamente tipati come stringhe

# Esercizio 11 (facoltativo)

- Suggestire (ed applicare) uno o più utilizzi delle classi nidificate nello studio di caso
- Scrivere dei test di unità a supporto della verifica di correttezza del codice prima e dopo l'utilizzo

# TERMINI E MODALITA' DI CONSEGNA

- La soluzione deve essere inviata al docente entro le 21:00 del 14 giugno 2019 come segue:
  - Svolgere in gruppi di max 2 persone
  - Esportare (con la funzione File->Export di Eclipse) il progetto realizzato nel file **homework4.zip**
  - Inviare il file **homework4.zip** all'indirizzo di posta elettronica [poo.roma3@gmail.com](mailto:poo.roma3@gmail.com)  Per consegnare usare questa email!
  - Nel corpo del messaggio riportare eventuali malfunzionamenti noti, ma non risolti
  - L'oggetto (subject) *DEVE* iniziare con la stringa **[2019-HOMEWORK4]** seguita dalle matricole
  - Ad es.: **[2019-HOMEWORK4] 512345 554321**

# TERMINI E MODALITA' DI CONSEGNA

- Attenzione:
  - senza l'invio di questo homework, non sarà possibile continuare il percorso HQ di questo anno accademico
  - con l'invio di questo homework, non sarà più possibile partecipare al percorso HQ nei successivi anni accademici