# HOMEWORK 2

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 POO-08
- 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»
– 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 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'instanziamento della corretta classe concreta associata a ciascun comando perché ancora non abbiamo studiato gli strumenti più opportuni per farlo correttamente

# 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

- Le recenti modifiche cambiano l'implementazione del gioco senza modificarne affatto il comportamento
- Subito dopo averle effettuate, riverificare con i test sviluppati in questo e negli homework precedenti la correttezza del codice per confermare che non si siano introdotti errori
  - ✓ ovvero che non ci sia *regressione*
- In presenza di test che cominciano a fallire, quando 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 sino a renderne palesi le cause

# Esercizio 4

- Implementare ed introdurre nel gioco la «stanza magica», come descritto nella trasparenza POO-10-estensione-protected
- Fare sia la versione con campi protetti sia quella che rispetta il principio dell'information hiding facendo utilizzare da parte delle classi estese solo la parte pubblica della classe base
- Chiamare le due classi rispettivamente **`StanzaMagicaProtected`** e **`StanzaMagica`**

# TDD (Facoltativo)

- ✓ N.B. È perfettamente lecito e consigliabile fare l'esercizio 7 anche prima degli esercizi 5&6

# Esercizio 5-6

- 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 5: stanza buia

- La classe **`StanzaBuia`** deve avere una variabile di istanza di tipo **`String`**, in cui memorizza il nome dell'attrezzo che consente di avere la descrizione completa della stanza
- Il metodo **`getDescrizione()`** va sovrascritto (override) 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 richiesto viene impostato attraverso il costruttore

# Esercizio 6: stanza bloccata

- La classe **`StanzaBloccata`** deve avere due variabili di istanza di tipo **`String`**:
  - una memorizza il nome della direzione bloccata
  - una memorizza 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 6: 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. 'chiave') e il nome della direzione bloccata vanno impostati attraverso il costruttore

# Esercizio 7

- 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 5&6

# TERMINI E MODALITA' DI CONSEGNA

- La soluzione deve essere inviata al docente entro le 21:00 del 30 aprile 2015 come segue:
  - Svolgere in gruppi di max 2 persone
  - Esportare (con la funzione File->Export di Eclipse) il progetto realizzato nel file **homework2.zip**
  - Inviare il file **homework2.zip** all'indirizzo di posta elettronica poo.roma3@gmail.com
  - Nel corpo del messaggio riportare eventuali malfunzionamenti noti, ma non risolti
  - L'oggetto (subject) deve iniziare con la stringa **[2015-HOMEWORK2]** seguita dalle matricole mittenti
  - Ad es.: **[2015-HOMEWORK2] 412345 454321**