

HOMEWORK 1 - 2021

Introduzione Java
Testing

Esercizio 0 (Set-Up)

- Installare Eclipse sul proprio calcolatore (si suppone la piattaforma Java già installata)
- Importare **diadia_base** come progetto Eclipse
- Eseguire DiaDia da Eclipse
- La versione base è piuttosto limitata e contiene degli errori
 - alcuni si manifestano solo a tempo di esecuzione con il sollevamento di una eccezione e la terminazione inattesa del programma
- Correggere gli errori

Esercizio 1 (Testing)

- Scrivere le classi di test JUnit per le classi **Stanza**, **Partita**
- Scrivere almeno tre metodi di test per ciascuno dei metodi più importanti di queste classi
- Perseguire la qualità dei test, in particolare la loro *minimalità*. Allo scopo scrivere test-case
 - brevi e coincisi: che utilizzino meno oggetti possibile ed in uno stato iniziale più semplice possibile
 - fattorizzati: senza codice duplicato ma evitando di comprometterne leggibilità ed autocontenimento
 - indipendenti: che evitino di ripetere scenari di test già coperti da altri test-case

Esercizio 2 (Refactoring)

- Introdurre la classe **Labirinto**
 - ha la responsabilità di creare il labirinto, di memorizzare la stanza iniziale (entrata) e quella finale (uscita)
 - aggiungere un riferimento ad un'istanza di **Labirinto** nella classe **Partita** (che ovviamente dovrà essere liberata dalle responsabilità spostate nella nuova classe)
- Introdurre la classe **Giocatore** e la classe **Borsa**
 - **Giocatore** ha la responsabilità di gestire i CFU del giocatore e di memorizzare gli attrezzi in un oggetto istanza della classe **Borsa** >>(vedi codice a seguire)
 - aggiungere un riferimento ad un'istanza di **Giocatore** nella classe **Partita** (che ovviamente dovrà essere liberata dalle responsabilità spostate nella nuova classe)
- Nell'ordine:
 - scrivere classi di test JUnit per **Giocatore**, **Borsa** e **Labirinto**
 - introdurre le classi **Labirinto** e **Giocatore** e **Borsa** nel codice

Esercizio 3 (Refactoring, Nuovi Comandi)

- Modificare il gioco affinché il giocatore possa "prendere" e "posare" degli attrezzi
- Per implementare questa modifica è necessario fare le seguenti operazioni:
 - completare il codice della classe `Stanza` ed in particolare del metodo `removeAttrezzo()`
 - utilizzare il codice della classe `Borsa` riportato di seguito completando il metodo `removeAttrezzo()`
 - modificare il codice della classe `DiaDia` implementando il codice per l'esecuzione dei comandi `prendi` e `posa`
 - gli attrezzi presi vengono rimossi dalla stanza e aggiunti alla borsa
 - gli attrezzi posati vengono rimossi dalla borsa e aggiunti alla stanza
 - la sintassi per inserire questi comandi è la seguente:
 - `prendi <nomeAttrezzo>`
 - `posa <nomeAttrezzo>`
 - modificare la logica del comando «aiuto» che deve tener conto del nuovo comando disponibile

Classe Borsa: Codice (1)

```
public class Borsa {
    public final static int DEFAULT_PESO_MAX_BORSA = 10;
    private Attrezzo[] attrezzi;
    private int numeroAttrezzi;
    private int pesoMax;

    public Borsa() {
        this(DEFAULT_PESO_MAX_BORSA);
    }

    public Borsa(int pesoMax) {
        this.pesoMax = pesoMax;
        this.attrezzi = new Attrezzo[10]; // speriamo che bastino...
        this.numeroAttrezzi = 0;
    }

    public boolean addAttrezzo(Attrezzo attrezzo) {
        if (this.getPeso() + attrezzo.getPeso() > this.getPesoMax())
            return false;
        if (this.numeroAttrezzi==10)
            return false;
        this.attrezzi[this.numeroAttrezzi] = attrezzo;
        this.numeroAttrezzi++;
        return true;
    }

    public int getPesoMax() {
        return pesoMax;
    }

    public Attrezzo getAttrezzo(String nomeAttrezzo) {
        Attrezzo a = null;
        for (int i= 0; i<this.numeroAttrezzi; i++)
            if (this.attrezzi[i].getNome().equals(nomeAttrezzo))
                a = attrezzi[i];
        return a;
    }
}
(continua)
```

Classe Borsa: Codice (2)

```
public int getPeso() {
    int peso = 0;
    for (int i= 0; i<this.numeroAttrezzi; i++)
        peso += this.attrezzi[i].getPeso();
    return peso;
}

public boolean isEmpty() {
    return this.numeroAttrezzi == 0;
}

public boolean hasAttrezzo(String nomeAttrezzo) {
    return this.getAttrezzo(nomeAttrezzo)!=null;
}

public Attrezzo removeAttrezzo(String nomeAttrezzo) {
    Attrezzo a = null;
    // ---> TODO (implementare questo metodo) <---
    return a;
}

public String toString() {
    StringBuilder s = new StringBuilder();
    if (!this.isEmpty()) {
        s.append("Contenuto borsa (" + this.getPeso() + "kg/" + this.getPesoMax() + "kg): ");
        for (int i= 0; i<this.numeroAttrezzi; i++)
            s.append(attrezzi[i].toString() + " ");
    }
    else
        s.append("Borsa vuota");
    return s.toString();
}
}
```

Esercizio 4 (Package)

- Il progetto sta crescendo: organizziamo meglio le classi in package
 - mettere **Labirinto** e **Stanza** nel package
`it.uniroma3.diadia.ambienti`
 - mettere **Attrezzo** nel package
`it.uniroma3.diadia.attrezzi`
 - mettere **Giocatore** e **Borsa** nel package
`it.uniroma3.diadia.giocatore`
 - mettere **Comando**, **DiaDia** e **Partita** nel package
`it.uniroma3.diadia`

Esercizio 5

(Disaccoppiamento I/O)

- La gestione dell'Input e dell'Output è attualmente “disseminata” in molti punti del codice
 - uso diretto di `System.out` (per la stampa su video)
 - uso diretto di `System.in` (per la lettura da tastiera)
- Questa situazione non è ottimale
 - difficile trovare e cambiare i messaggi stampati
 - Dentro un test-case risulta poco agevole
 - sia controllare le stampe con delle asserzioni
 - sia iniettare dei comandi scelti nel test stesso
 - sarà difficile cambiare la modalità di interazione con l'utente (ad es. per passare ad una modalità alternativa basata sull'utilizzo di GUI)

Esercizio 5

➤ Riorganizziamo completamente la gestione dell'I/O, disaccoppiando il gioco dall'uso diretto e pervasivo di `System.out/System.in`, operando come segue:

- Introduciamo la classe `it.uniroma3.diadia.IOConsole` (fornita nella prossima slide) che centralizza l'accesso a `System.out/System.in`
- Creiamo una sola istanza di questa classe nell'unico metodo `main()` del gioco di ruolo `DiaDia.main()`
- Rifattorizziamo tutto il codice per far arrivare tale unica istanza ove serve
 - Può essere necessario aggiungere costruttori e/o modificare quelli già esistenti in alcune classi
- Eliminiamo tutte le stampe e le letture dirette dal resto del codice rimpiazzandole con un appropriato uso dei metodi della classe `IOConsole`

Esercizio 5: IOConsole - ATTENZIONE

- NON può essere modificato il codice di `IOConsole`
- NON può essere modificato il suo package
- Crearne UNA SOLA ISTANZA in tutto il codice nel metodo `DiaDia.main()`
- NON è più possibile usare `System.in` e `System.out` altrove

```
package it.uniroma3.diadia;
import java.util.Scanner;

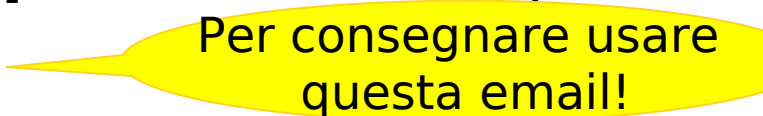
public class IOConsole {

    public void mostraMessaggio(String msg) {
        System.out.println(msg);
    }

    public String leggiRiga() {
        Scanner scannerDiLinee = new Scanner(System.in);
        String riga = scannerDiLinee.nextLine();
        scannerDiLinee.close();
        return riga;
    }
}
```

Omettere questa riga,
ci torneremo sopra

TERMINI E MODALITA' DI CONSEGNA

- La soluzione deve essere inviata al docente entro le 21:00 del'11 aprile 2021 come segue:
 - Svolgere in gruppi di max 2 persone
 - Esportare (con la funzione File->Export di Eclipse) il progetto realizzato nel file **homework1.zip**
 - Inviare il file **homework1.zip** all'indirizzo di posta elettronica 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 **[2021-HOMEWORK1]** seguita dalle matricole
 - Ad es.: **[2021-HOMEWORK1] 412345 454321**