Progetto per l'esame del corso di Programmazioen ad Oggetti, a.a. 2019/2020

Il progetto va svolto o singolarmente o in gruppo di al più 2 persone. Per avere un gruppo da 3 si deve avere una buona motivazione e la mia autorizzazione.
Il progetto deve essere consegnato 3 giorni prima dell'esame scritto. Metterò un contenitore sul sito per la consegna.

Requisiti comuni

Lo svolgimento di tutti i progetti deve tenere in considerazione i seguenti requisiti. In particolare, la proposta scelta deve essere implementata seguendo i seguenti principi :

- le classi interne del sistema devono essere corredate da classi di test;
- la documentazione deve essere prodotta con **javadoc** (trovate una sintetica descrizione di come annotare il codice sul sito del corso);
- ogni progetto deve essere accompagnato da una breve relazione in cui vengono spiegate ed illustrate le scelte adottate
- inoltre per quanto riguarda l'interfaccia utente (che deve essere separata dalla classi che realizzano la logica del sistema) non è necessaria una interfaccia grafica, per cui si deve poter accedere alla funzionalità del programma da riga di comando e i risultati delle operazioni e nel caso del gioco le configurazioni successive devono essere mostrate sulla console.

Nello svolgere il progetto usare i principi visti nel corso. In particolare, buona strutturazione delle classi, uso di classi astratte e interfacce se necessario. Ogni classe ha un compito preciso (e fa solo quello) , ha campi privati (o protected in caso si pensi possa avere sottoclassi), uso di eccezioni e non "stampe" per segnalazioni di malfunzionamenti, metodi di dimensione limitata, ecc...

L'interfaccia utente deve essere semplicemente un `main` dal quale è possibile eseguire tutte le operazioni e nel quale vengono catturate le eventuali eccezioni sollevate che vengono gestite chiedendo all'utente di chiedere di nuovo l'esecuzione di una operazione.

Valutazione

La discussione del progetto con i singoli studenti verterà sulle modalità di implementazione adottate e sulla padronanza di alcuni dei concetti necessari per realizzare il progetto e/o spiegati a lezione. La valutazione del progetto sarà fatta in base alla: conformità dell'implementazione scelta per risolvere il problema con il paradigma di programmazione a oggetti e alla conformità del codice presentato alle regole di buona programmazione.

Lo studente deve implementare una delle 2 seguenti proposte:

Proposta 1. Realizzare una ListaDellaSpesa

Realizzare un programma per mantenere varie **liste della spesa**, che hanno un nome e un contenitore degli **articoli** che devono essere comprati. Gli articoli sono classificati secondo **categorie** definite dall'utente (rappresentate da una stringa) e hanno associata una *quantità* (usate un numero `float`). L'associazione fra un articolo e una categoria può essere *modificata* e ci possono essere articoli con non sono categorizzati (potete avere una categoria predefinita "Non Categorizzati"). Quando *inserite* un articolo questo è inizialmente non categorizzato e poi potete *modificarne la categoria* come potete *modificare la quantità* associata che per default è 1. La *rimozione* di un articolo non comporta il suo scomparire dalla lista, ma il passaggio dello stesso dalla lista della spesa a una **lista di articoli cancellati**. Potete *selezionare* un articolo dalla lista dei cancellati e riportarlo nella lista della spesa. La lista dei cancellati può essere svuotata. Deve essere possibile data una stringa trovare un articolo che ha la stringa come prefisso, cercando sia nella lista della spesa che in quella dei cancellati. La lista della spesa deve essere iterabile per permettere ulteriori elaborazioni da parte del codice cliente.

Per darvi un'idea di quali operazioni si possono fare guardate a questo link l'app OurGroceries

https://www.ourgroceries.com

ovviamente a noi interessa modellare SOLAMENTE LE CLASSI DEL DOMINIO non l'interfaccia grafica.

Quindi la struttura è molto simile a quella della Rubrica. La classe **ListaSpesa** è l'analogo della classe Rubrica La classe **Articolo** è una classe interna che contiene: nome, quantità e categoria di un articolo. La categoria è una stringa.

In più avete una classe **GestioneListe** che contiene: un campo statico *listeSpesa* con l'associazione fra il nome di una lista e un riferimento a un oggetto di tipo ListaSpesa e un campo static *categorie* che lista tutte le categorie (che sono comuni a tutte le liste della spesa).

Metodi STATICI per creare una lista, cancellare una lista e per inserire e cancellare una catgoria. Poi questa classe fa da interfaccia alle operazioni delle varie liste. Allego uno scheletro di una classe **GestioneRubriche** per darvi un'idea di come realizzare **GestioneListe**.

Vi elenco la funzionalità che dovete realizzare:

1) Creare una lista dandole un nome
2) Creare una categoria
3) Leggere/scrivere una lista su un file (potete scegliere se usare un file di testo oppure serializzare la classe)
4) Iterare sugli elementi di una lista (che devono essere mantenuti ordinati)
5) Inserire un articolo in una lista con una quantità (inizialmente l'articolo non è categorizzato)
6) Cercare un articolo fornendo il prefisso della stringa (cercare anche nella lista degli articoli rimossi). La ricerca ritorna l'indice dell'articolo nella lista della spesa o in quella dei rimossi.
7) Rimuovere un articolo da una lista (questo lo mette nella lista dei rimossi).
8) Ripristinare un articolo dalla lista dei rimossi
9) Modificare la categoria, modificare la quantità di un articolo
10) Svuotare la lista degli articoli rimossi

Le operazioni 7), 8) e 9) sono fatte dopo la 6) e possono usare come input l'indice nella lista ritornato dalla ricerca.

Le classi **Articolo** e **ListaSpesa** devono ridefinire *toString()* di Object e Articolo deve ridefinire *equals(Object obj)* di Object

```java
import java.util.ArrayList;
import java.util.HashMap;

public class GestioneRubriche {

	private static HashMap<String, Rubrica> listaRubriche;

	public static void inizializza() {
		listaRubriche = new HashMap<String, Rubrica>();
	}

	public static void crea(String nomeRubrica, int maxDim) throws ???? {
		if(!listaRubriche.containsKey(nome)) {
			listaRubriche.put(nome, new Rubrica(nomeRubrica,maxDim));
		}
		.....
	}

	public static boolean cancella(String nome) {
		 if(listaRubriche.containsKey(nome)) {
			listaRubriche.remove(nome);
			}
		.....
	}

	public static void aggiungiElRubrica(String nomeRub, String con) throws ???  {
	}

	public static ArrayList<String> cercaElRubrica(String nomeRub,String prefisso) {
				.....
	}

	public static boolean rimuoviElRubrica(String Rubrica,String prefisso) {
				.....
	}

	public static String toStringRubrica(String Rubrica) {
				.....
	}
}
```

Proposta 2. Realizzare Pac-Man

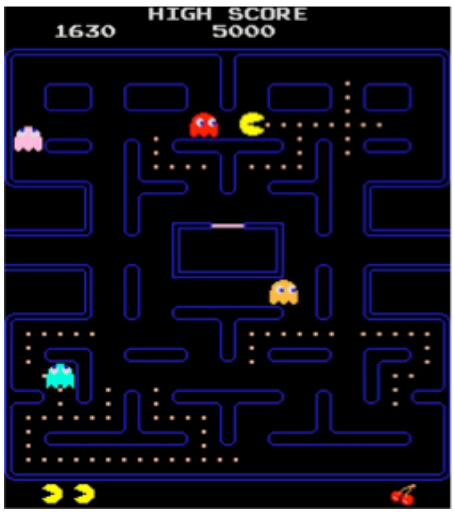
Realizzare un programma per simulare (con notevoli restrizioni) il video-gioco Pac-Man. In particolare data una configurazione iniziale per il campo di gioco (posizione di pac-man, dei fantasmi e delle pillole) e una sequenza di mosse si dovrà simulare l'avanzamento della partita.

Il Gioco

Nella versione originale del gioco, il giocatore deve guidare una creatura sferica di colore giallo, chiamata Pac-Man, facendole mangiare tutti i numerosi puntini disseminati ordinatamente all'interno del labirinto e, nel far questo, deve evitare di farsi mangiare a sua volta da quattro fantasmi, pena la perdita immediata di una delle vite a disposizione. Per facilitare il compito al giocatore sono presenti, presso gli angoli dello schermo di gioco, quattro pillole speciali (power pills) che rovesciano la situazione rendendo vulnerabili i fantasmi, che diventano blu e, per qualche istante, invertono la loro marcia; per guadagnare punti, è possibile in questa fase andare a caccia degli stessi fantasmi, per mangiarli. Una volta fagocitati, però, questi tornano alla base (il rettangolo al centro dello schermo) sotto forma di un paio di occhi, per rigenerarsi ed attaccare di nuovo pac-man. Completato un labirinto attraverso la fagocitazione di tutti i puntini, Pac-Man passa a quello successivo, identico nella struttura.

Durante il gioco, ogni cosa che finisce in bocca a Pac-Man viene contabilizzata sotto forma di punti. I puntini disseminati lungo il labirinto valgono 10 punti ognuno mentre le power pills di punti ne offrono 50 ognuna.

È possibile ottenere punti extra anche mangiando i fantasmi una volta resi vulnerabili da una delle power pills: in tal caso si ottengono 200, 400, 800 e 1.600 punti fagocitando i fantasmi in sequenza (200 punti con il primo fantasma, 400 con il secondo e così via). Esiste anche un'altra possibilità per l'aumento dei propri punti: a un certo momento, durante l'esecuzione di ogni livello, appare al centro del labirinto un'icona, il più delle volte rappresentante un frutto. Se il giocatore è abbastanza abile da recuperarla prima che scompaia,



accresce il proprio punteggio.

Semplificazioni

Ovviamente non è possibile, nell'ambito del progetto, implementare interamente il gioco stesso ma dovrete realizzarne una simulazione che permetta il passaggio da una configurazione ad un'altra. Il tutto con le seguenti limitazioni/varianti rispetto al gioco originale.

- Pac-Man si muove in una direzione e quando arriva contro un muro, l'utente specifica se deve continuare andando su, giù, a destra o a sinistra;
- i fantasmi (blinky, pinky, inky, clyde) seguono una logica preconfezionata (che decidete voi);
- l'area centrale di ristoro dei fantasmi non esiste;

- quando fagocitato da Pac-Man il fantasma riappare nella configurazione successiva in un punto qualsiasi;
- l'effetto della pillola dura per cinque configurazioni successive;
- si hanno solo tre vite, l'esser mangiato dai fantasmi ha solo l'effetto di perdere una vita, la situazione di pillole, puntini e fantasmi rimane inalterata e il gioco riparte con Pac-Man e i fantasmi nella situazione iniziale;
- non ci sono i bonus.

Dovete realizzare in Java il programma descritto nelle sezioni precedenti rappresentando con classi tutti gli elementi del gioco: Pillole, Puntini, Muri, SpazioVuoto, Pac-Man, Fantasmi, ecc.... A questi è necessario dare una strutturazione che differenzi elementi che si muovono o meno. Per cui potete definire una classe astratta Elemento con le precedenti classi come sottoclassi concrete. Inoltre il movimento può essere descritto da una interfaccia che viene implementata dagli elementi che si muovono.

Se volete definire una diversa strutturazione, ne dovete dare una convicente giustificazione.

La configurazione del gioco (lo schermo del videogioco con tutte le informazioni rilevanti) sarà rappresentato da una classe Configurazione. Questa conterrà una matrice NxM con N e M non definiti a priori (verranno letti all'inizializzazione dei gioco) contenente gli elementi del gioco (se volete potete isolare questa componente in una classe CampoDiGioco). Una istanza di una Configurazione sarà rappresentata da una stringa codificata in una stringa che deve codificare:

- la dimensione del campo di gioco
- le informazioni sulle vite di PacMan, il punteggio, l'eventuale effetto data da una pillola
- il contenuto del campo di gioco

Il costruttore della classe creerà una configurazione del gioco a partire da una stringa che deve essere il nome di un file su disco (questo nome sarà memorizzato in un campo fileName) che contiene una sequenza di configurazioni. L'ultima configurazione diventa quella corrente.

Fra i metodi di questa classe in aggiunta al metodo

- move()

che fa avanzare la configurazione di un passo (modificandola) ci saranno i metodi che permettono di:

- leggere/modificare I singoli elementi del campo di gioco
- ritornare/modificare tutta la configurazione; nel primo caso si ritorna la stringa che la codifica e nel secondo si modifica a partire da una stringa (codifica di una configurazione)
- salvare la configurazione corrente appendendola al file contenuto nel campo fileName
- fare un ripristino facendo diventare corrente l'ultima configurazione del fileName
- toString()

L'interfaccia con l'utente deve permettere di far eseguire il gioco dall'inizio (nel qual caso si scrive su file la configurazione iniziale), salvare e/o rispristinare configurazioni.

Se volete (ma non è richiesto) potete anche far cambiare dall'utente la direzione di PacMan.