

Dipartimento di Ingegneria e Scienze dell’Informazione e Matematica

CORSO DI LAUREA IN INFORMATICA

Insegnamento Laboratorio di programmazione ad oggetti

Monopoli-Univaq

|  |  |  |  |
| --- | --- | --- | --- |
| **Membri del team** | | | |
| **Cognome e nome** | **Matricola** | | **Indirizzo e-mail** |
| Scappa Marco | 279487 | | marco.scappa@student.univaq.it |
| Bucci Francesco | 279657 | | francesco.bucci2@student.univaq.it |
| Micolucci Jacopo | 293491 | | jacopo.micolucci@student.univaq.it |
|  | |  | | |
|  | | | | |
|  | |  | | |

A.A. 2023/2024

Sommario

[Capitolo 1 – Caso di studio 2](#_Toc38393257)

[1.1 – Descrizione dell’applicazione 2](#_Toc38393258)

[1.2 - Funzionalità 3](#_Toc38393259)

[1.3 Domain Model 4](#_Toc38393260)

# Capitolo 1 – Caso di studio

## 1.1 – Descrizione dell’applicazione

Il Progetto è una versione digitale del popolare gioco da tavolo Monopoli ed è stato sviluppato utilizzando il linguaggio di programmazione Java.

Il progetto è una fedele ricostruzione del gioco originale tranne per alcuni aspetti come, per esempio, la possibilità di ipotecare una proprietà per ricevere del denaro, la modifica nel testo delle carte imprevisti e probabilità, il nome e i costi delle caselle acquistabili, la grandezza del tabellone che varia in base al numero di partecipanti e la semplificazione dello scambio tra player che consente solo di scambiare una proprietà con un’altra.

Il progetto è strutturato affinché gli utenti possano scegliere all’inizio del gioco il numero di giocatori attraverso la classe commandLineSingleton che implementa metodi come readIntegerUnitlPossibleValue, il quale permette di scrivere in console un range di numeri, dove ad ogni intero corrisponde una scelta grazie alle istruzioni condizionali.

Il gioco permette anche la scelta di generare una partita contro la Cpu grazie alle implementazioni della classe CpuServiceImpl, dove ad ogni scelta il computer decide attraverso l’utilizzo della classe Random che effettua una scelta casuale tra quelle possibili in un determinato momento della partita.

Il tabellone di gioco è stato implementato in modo dinamico attraverso l’utilizzo di metodi come inizializzaPartita, che contiene metodi che permettono di scegliere quanti giocatori ci sono nella partita tramite l’input su console e quanto deve essere grande il tabellone in quella determinata partita (es. 2 giocatori = 24 caselle di cui 4 speciali e 20 divise tra 12 proprietà, 2 imprevisti e 2 probabilità, 2 stazioni, 1 servizio e 1 tassa).

Nel nostro specifico caso di implementazione abbiamo fatto si che in due giocatori solo uno dei due potesse essere gestito dal computer ( la stessa cosa vale per tre giocatori); in quattro giocatori al massimo solo due dei quatto possono essere gestiti da computer ( la stessa cosa vale per cinque giocatori), e infine nel caso di sei giocatori solo al massimo tre di questi sei possono essere gestiti da computer.

Esso include caselle che rappresentano proprietà immobiliari e stazioni ferroviarie insieme a spazi dedicati a società di servizi e imposte. Saranno presenti anche caselle speciali come “Prigione e Transito”, “Parcheggio gratuito", “Vai in Prigione” e “Via”, le quali saranno sempre presenti per ogni tipo di partita.

L’interfaccia utente basata su console permette ai giocatori di monitorare i propri progressi e valutare le strategie disponibili.

Nel progetto sono presenti delle classi astratte come Casella e le sue sottoclassi ovvero CasellaAcquistabile e CasellaNonAcquistabile che sono diverse dalle altre classi perché non possono essere istanziate direttamente ma devono essere estese da altre classi, ovvero i tipi di caselle presenti nel gioco come quelle Acquistabili (proprietà, servizi e stazioni) e NonAcquistabili (tasse, via, vai in prigione, parcheggio gratuito, imprevisti, probabilità e prigione e transito).

## 1.2 – Funzionalità

Il Progetto è diviso in directory chiamate: datamodel, service {Enumeration, Exception, Impl [Factoryes]}.

Nel datamodel si trovano le classiche classi che troviamo anche nel class diagram riportato in fondo al documento:

* **Carta**, contenente gli attributi delle carte imprevisti e probabilità per far si che abbiano effetto durante la partita;
* **Partita**, la classe principale che permette la generazione del tabellone, la generazione dei mazzi probabilità e imprevisti con i relativi effetti e i metodi per ottenere il giocatore di turno durante la partita e i turni della partita incrementati ad ogni turno;
* **Player,** contiene tutte le informazioni riguardante i giocatori come il nome, il bilancio o le informazioni sulle proprietà possedute durante il gioco;
* **Tabellone,** contenente la codifica del tabellone tramite hashmap;
* **Casella,** classe astratta che fa da superclasse alle altre classi (CasellaAcquistabile e CasellaNonAcquistabile) con metodi che permettono di aggiungere o rimuovere un giocatore che si trova su una determinata casella e di ottenere la lista dei giocatori presenti su quella casella;
* **CasellaNonAcquistabile,** classe astratta cheestende Casella e fa da classe base alle caselle che durante la partita non possono essere acquistate;
* **VaiInPrigione,** contenente il metodo che permette lo spostamento del giocatore in prigione;
* **ParcheggioGratuito,** classe che eredita l’attributo nomeCasella dalla classe base;
* **Tassa,** classe che contiene il costo che il giocatore è costretto a pagare quando passa sulla casella Tassa;
* **PrigioneETransito,** classe che contiene la codifica della casella tramite hashmap che permette di capire se un giocatore si trova in prigione oppure è solo in transito;
* **Via,** classe che contiene l’importo che ogni giocatore guadagna quando passa da questa casella durante la partita;
* **CasellaConCarta,** contiene il metodo per la specifica del tipo di casella in cui ci troviamo, ovvero se imprevisto o probabilità;
* **CasellaAcquistabile,** classe astratta che fa da superclasse per le caselle acquistabili durante la partita, la quale contiene gli attributi che verranno ereditati dalle sottoclassi;
* **Stazione,** classe che estende casellaAcquistabile e ne eredita gli attributi per specificare le informazioni riguardanti i costi sull’acquisto e la vendita delle stazioni, e il metodo per vedere quante stazioni si possiede;
* **Servizio,** classe che estende casellaAcquistabile e ne eredità gli attributi per specificare le informazioni riguardanti i costi sull’acquisto e la vendita dei servizi, e il metodo per vedere quanti servizi si possiede;
* **Proprietà,** classe che estende casellaAcquistabile e ne eredita gli attributi per specificare le informazioni riguardanti i costi sull’acquisto e la vendita delle proprietà, a differenza delle classi stazione e servizi però nelle proprietà ci sono più metodi per vedere se sulla proprietà si possono costruire case o alberghi oppure per vedere se si ha la serie completa di proprietà per poter costruire;
* **Albergo,** classe che insieme alla classe Casa permettono di capire chi è il proprietario di una determinata proprietà in modo tale da poterci poi costruire sopra una volta utilizzati i metodi della classe Proprietà;
* **Casa,** classe che insieme alla classe Albergo permettono di capire chi è il proprietario di una determinata proprietà in modo tale da poterci poi costruire sopra una volta utilizzati i metodi della classe Proprietà;
* **Scambio,** classe che contiene i metodi utili allo scambio come;
* **Cpu,** classe che estende Player e ne eredita gli attributi con super();
* **Umano,** classe che estende Player e ne eredita gli attributi con super();

Il progetto è stato organizzato in più classi, però le classi principali sono quelle che gestiscono la partita (Partita, PartitaService, PartitaServiceImpl).

Nella cartella di Service troviamo i servizi delle varie entità e le loro implementazioni :

* **CasellaService**, è l’interfaccia padre di una gerarchia di caselle;
* **CasellaAcquistabileService**, interfaccia che estende CasellaService e gestisce i servizi per le caselle acquistabili ;
* **CasellaNonAcquistabileService**, interfaccia che estende CasellaService e gestisce i servici per le caselle non acquistabili ;
* **PlayerService**, interfaccia che gestisce i servizi per i player ( CPU e Umano);
* **ScambioService**, interfaccia che gestisce i servizi per gli scambi tra i giocatori;
* **PartitaService**, interfaccia che gestisce i servizi per la partita;

Nella cartella impl sono definite le implementazioni delle interfaccie della cartella Service:

* **ProprietaServiceImpl**, classe che implementa l’interfaccia CasellaAcquistabileService e gestisce il caso delle proprietà;
* **ServizioServiceImpl**, classe che implementa l’interfaccia CasellaAcquistabileService e gestisce il caso dei servizi;
* **StazioneServiceImpl**, classe che implementa l’interfaccia CasellaAcquistabileService e gestisce il caso delle stazioni ;
* **ParcheggioGratuitoServideImpl**, classe che implementa l’interfaccia CasellaNonAcquistabileService e gestisce il caso del parcheggio gratuito;
* **PrigioneETransitoServiceImpl**,classe che implementa l’interfaccia CasellaNonAcquistabileService e gestisce il caso della prigione e transito;
* **ViaServiceImpl**, classe che implementa l’interfaccia CasellaNonAcquistabileService e gestisce il caso del via;
* **VaiInPrigioneServiceImpl**, classe che implementa l’interfaccia CasellaNonAcquistabileService e gestisce il caso del vai in prigione;
* **CasellaConCartaServiceImpl**, classe che implementa l’interfaccia CasellaNonAcquistabileService e gestisce il caso delle caselle con carta;
* **TassaServiceImpl**, classe che implementa l’interfaccia CasellaNonAcquistabileService e gestisce il caso delle Tasse;
* **UmanoServiceImpl**, classe che implementa l’interfaccia PlayerService e gestisce il caso del giocatore Umano;
* **CpuServiceImpl**, classe che implementa l’interfaccia PlayerService e gestisce il caso del giocatore Computer;
* **ScambioServiceImpl**, classe che implementa l’interfaccia ScambioService ;
* **PartitaServiceImpl**, classe che implementa l’interfaccia PartitaService;

Nella cartella Factoryes sono gestite tutte le classi Factory:

* **PlayerServiceFactory**, questa classe prende in input un ogetto di tipo Class<T extends Player> e restituisce l’implementazione giusta dei servizi ( Cpu oppure Umano) ;
* **CasellaAcqistabileServiceFactory** , questa classe prende in input un ogetto di tipo Class<T extends Casella> e restituisce l’implementazione giusta dei servizi ( Proprietà Servizio oppure Stazione);
* **CasellaServiceFactory**, questa classe prende in input un ogetto di tipo Class<T extends Casella> e restituisce l’implementazione giusta dei servizi ( Via, Prigione E Transito, Parcheggio Gratuito, Vai In Prigione, Casella Con Carta oppure Tassa);

Queste classi contengono principalmente il succo del progetto, all’interno di ogni classe troviamo le istruzioni necessarie per svolgere la determinata azione che contiene nel proprio titolo

## 1.3 – Domain Model

Di seguito mostriamo l’immagine del class diagram contenente le entità:

* **Carta**
* **Partita**
* **Player**
* **Tabellone**
* **Casella**
* **CasellaNonAcquistabile**
* **VaiInPrigione**
* **ParcheggioGratuito**
* **Tassa**
* **PrigioneETransito**
* **Via**
* **CasellaConCarta**
* **CasellaAcquistabile**
* **Stazione**
* **Servizio**
* **Proprieta**
* **Albergo**
* **Casa**
* **Scambio**
* **Cpu**
* **Umano**

## 

Immagine che contiene testo, schermata, diagramma, Piano

Descrizione generata automaticamente