Università degli Studi di Ferrara

Università degli Studi di Ferrara

DE: Department of Engineering Ferrara

Damiano Bressanin 138075

INGEGNERIA DEL SOFTWARE AVANZATA

PRESENTAZIONE DEL PROGETTO

SOMMARIO

- Descrizione
- Specifiche
- Progettazione
- Implementazione
- Test
- Deployment

DESCRIZIONE DEL PROGETTO (1/2)

Si vuole creare, utilizzando le competenze apprese nel corso di Ingegneria del Software Avanzata, un software in Java per supportare una piccola azienda agricola a conduzione familiare a regime forfettario agricolo nelle sue operazioni di vendita al minuto semplificando la gestione dei diversi regimi fiscali applicati ai tipi di prodotti venduti dall'azienda.

I prodotti venduti dall'azienda sono standard e le loro caratteristiche sono note a priori.
Ci sono tre categorie principali di prodotti, ognuna con specifiche esigenze fiscali:

- Prodotto agricolo: l'agricoltore non deve emettere uno scontrino fiscale sul posto. A fine giornata deve registrare queste vendite sul registro cartaceo dei corrispettivi;
- Prodotto commerciale: l'agricoltore deve fare lo scontrino fiscale immediato;
- Prodotto di trasformazione di un prodotto agricolo: l'agricoltore deve fare lo scontrino fiscale immediato.

In situazioni di vendite miste lo scontrino fiscale deve essere emesso solo per i prodotti commerciali e di trasformazione ma non per i prodotti agricoli. Tuttavia, l'importo totale che il cliente deve pagare include il prezzo di tutti i prodotti acquistati.

Descrizione del Progetto (2/2)

L'agricoltore inserirà nel software i prodotti scelti dal cliente e il programma calcolerà il totale da chiedere al cliente e le componenti di cui è costituito. Queste informazioni saranno utilizzate dall'agricoltore per emettere correttamente gli scontrini fiscali tramite il registratore di cassa e per richiedere l'importo esatto al cliente.

Attualmente, l'agricoltore gestisce queste operazioni manualmente, utilizzando carta e penna, e ciò può portare a errori ed inefficienze. Si vuole quindi aiutare l'agricoltore nel calcolo dei totali e nella distinzione tra le diverse categorie di prodotti con lo scopo di migliorare l'efficienza e di ridurre il rischio d'errore.

Il registro fiscale dei corrispettivi (cartaceo) ha colonne che rappresentano aliquote iva differenti e va compilato dividendo il totale delle vendite agricole giornaliere nelle rispettive colonne.

All'avvio del programma verranno inserite dall'utente le quantità di prodotti disponibili sul banco per la vendita e il software terrà traccia della somma del valore dei prodotti agricoli venduti giornalmente divisi per aliquote iva, facilitando la registrazione quotidiana delle vendite nel registro fiscale dei corrispettivi.

Questo sistema aiuterà l'agricoltore a gestire meglio le vendite giornaliere e a mantenere un'accurata contabilità fiscale.

SPECIFICHE

L'utente del programma deve poter:

- Incrementare la quantità dei prodotti all'interno dell'inventario;
- Decrementare la quantità dei prodotti all'interno dell'inventario;
- Visualizzare i prodotti nell'inventario;
- Aggiungere prodotti nel carrello;
- Rimuovere prodotti dal carrello;
- Visualizzare il contenuto del carrello;
- Visualizzare il totale da richiedere al cliente e le componenti di cui è costituito;
- Confermare la vendita dei prodotti presenti nel carrello;
- Visualizzare il registro delle vendite agricole creato dal software.

PROGETTAZIONE

L'applicazione è stata progettata utilizzando un'architettura monolitica e con un approccio bottom-up.
Sono state individuate le seguenti classi:

- **TipoProdotto:** enum che definisce le tipologie di prodotto;
- **Prodotto:** rappresenta un prodotto dell'azienda e le sue caratteristiche;
- **Inventario:** gestisce i prodotti disponibili per la vendita;
- **RegistroVendite:** gestisce il registro delle vendite agricole creato dal software;
- **Vendita:** gestisce il processo di vendita con un cliente;
- **Main:** gestisce l'interazione con l'utente.

IMPLEMENTAZIONE (1/5)

Il software è stato scritto in Java utilizzando Maven per il building, JUnit per i test, Git per il controllo di versione e GitHub per ospitare il repository Git.

IMPLEMENTAZIONE (2/5)

Visto che i prodotti venduti dall'azienda sono standard è stato scelto di utilizzare una versione molto semplificata del "Factory pattern" per la classe Prodotto in modo da creare istanze con un numero limitato di configurazioni predefinite. Quindi si ha il costruttore della classe che è privato e l'unico modo che si ha per creare un oggetto di tipo Prodotto è quello di utilizzare uno degli appositi metodi che andrà a chiamare il costruttore fornendogli le informazioni specifiche del prodotto da creare.

```java
public class Prodotto {
    private final String nome;
    private final double prezzo;
    private final TipoProdotto tipo;
    private final double IVA;

    private Prodotto(String nome, double prezzo, TipoProdotto tipo, double IVA) {
        this.nome = nome;
        this.prezzo = prezzo;
        this.tipo = tipo;
        this.IVA = IVA;
    }

    static Prodotto creaPiantaLavanda() {
        return new Prodotto(nome:"Pianta lavanda", prezzo:8.50, TipoProdotto.AGRICOLO, IVA:10.0);
    }

    static Prodotto creaPiantaSalvia() {
        return new Prodotto(nome:"Pianta salvia", prezzo:5.0, TipoProdotto.AGRICOLO, IVA:5.0);
    }
```

[...]

IMPLEMENTAZIONE (3/5)

Come strutture dati principali sono state utilizzate delle mappe HashMap lavorando quindi con coppie chiave-valore. Per poterle sfruttare al meglio, soprattutto per la gestione dei prodotti, nella classe Prodotto è stato eseguito l'override dei metodi equals e hashCode.

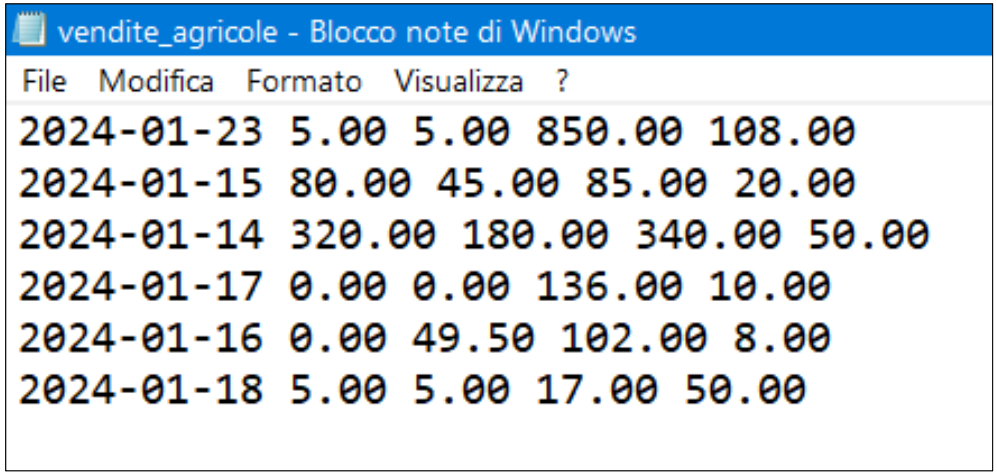
```
@Override
public boolean equals(Object o) {
    // due prodotti sono uguali se hanno lo stesso indirizzo di memoria
    if (this == o) {
        return true;
    }
    // se è null oppure sono classi diverse allora non sono uguali
    if (o == null || getClass() != o.getClass()) {
        return false;
    }
    // casting per poter accedere ai campi
    Prodotto prodotto = (Prodotto) o;
    return Objects.equals(this.getNome(), prodotto.getNome());
}

@Override
public int hashCode() {
    return Objects.hash(this.getNome());
}
```

IMPLEMENTAZIONE (4/5)

Dato che il software deve memorizzare in maniera persistente solo poche informazioni è stato scelto di utilizzare un file di testo e viene gestito dalla classe RegistroVendite.

Esempio di file delle vendite agricole e di come viene presentato all'utente:



Se il programma viene avviato più volte durante la stessa giornata e vengono salvate nuove vendite il file viene aggiornato in maniera opportuna.

```
REGISTRO VENDITE AGRICOLE:
Dettagli delle vendite per 2024-01-23:
Aliquota IVA 4.0%: €5.00
Aliquota IVA 5.0%: €5.00
Aliquota IVA 10.0%: €850.00
Aliquota IVA 22.0%: €108.00

Dettagli delle vendite per 2024-01-18:
Aliquota IVA 4.0%: €5.00
Aliquota IVA 5.0%: €5.00
Aliquota IVA 10.0%: €17.00
Aliquota IVA 22.0%: €50.00

Dettagli delle vendite per 2024-01-17:
Aliquota IVA 4.0%: €0.00
Aliquota IVA 5.0%: €0.00
Aliquota IVA 10.0%: €136.00
Aliquota IVA 22.0%: €10.00

Dettagli delle vendite per 2024-01-16:
Aliquota IVA 4.0%: €0.00
Aliquota IVA 5.0%: €49.50
Aliquota IVA 10.0%: €102.00
Aliquota IVA 22.0%: €8.00

Dettagli delle vendite per 2024-01-15:
Aliquota IVA 4.0%: €80.00
Aliquota IVA 5.0%: €45.00
Aliquota IVA 10.0%: €85.00
Aliquota IVA 22.0%: €20.00

Dettagli delle vendite per 2024-01-14:
Aliquota IVA 4.0%: €320.00
Aliquota IVA 5.0%: €180.00
Aliquota IVA 10.0%: €340.00
Aliquota IVA 22.0%: €50.00
```

IMPLEMENTAZIONE (5/5)

Per l'interazione con l'utente è stata sviluppata una Command Line Interface, offrendo un modo semplice ed efficace per sfruttare a pieno tutte le funzionalità del software.

```
MENÙ PRINCIPALE:
0. Esci
1. Gestione Inventario
2. Nuova Vendita
3. Visualizza vendite agricole in una data
4. Visualizza tutto registro vendite agricole


Inserisci un numero intero compreso tra: 0 e 4
```

```
MENÙ INVENTARIO:
0. Esci
1. Aggiungi prodotto
2. Rimuovi prodotto
3. Azzera prodotto
4. Aggiungi quantità a tutto l'inventario
5. Visualizza inventario


Inserisci un numero intero compreso tra: 0 e 5
```

```
MENÙ VENDITA:
0. Esci
1. Aggiungi prodotto al carrello
2. Rimuovi prodotto dal carrello
3. Azzera prodotto dal carrello
4. Svuota carrello
5. Visualizza carrello attuale
6. Visualizza prodotti disponibili (inventario)
7. Visualizza totali e riepilogo
8. Finalizza vendita


Inserisci un numero intero compreso tra: 0 e 8
```

TEST (1/2)

Per il testing è stato utilizzato il framework JUnit. È stata creata una suite di unit test comprendente oltre cinquanta test, ponendo particolare attenzione ai casi speciali.

Per esempio, ecco i test per il metodo «aggiungiProdottoCarrello» della classe Vendita:

```
@Test
void testAggiungiProdottoCarrello() {…

@Test
void testAggiungiProdottoCarrelloGiaEsistente() {…

@Test
void testAggiungiTantiProdottiCarrello() {…

@Test
void testAggiungiProdottoCarrelloQuantitaNegativa() {…

@Test
void testAggiungiProdottoCarrelloQuantitaZero() {…

@Test
void testAggiungiProdottoCarrelloTuttaQuantitaDisponibile() {…

@Test
void testAggiungiProdottoCarrelloTroppaQuantita() {…
```

TEST (2/2)

Per assicurare l'integrità e la riproducibilità dei test sono state usate le annotazioni fornite da JUnit come @BeforeAll, @BeforeEach e @AfterEach.

Questo approccio ha garantito che ogni test venisse eseguito in un contesto controllato e isolato, prevenendo interferenze tra test diversi e assicurando così l'affidabilità dei risultati.

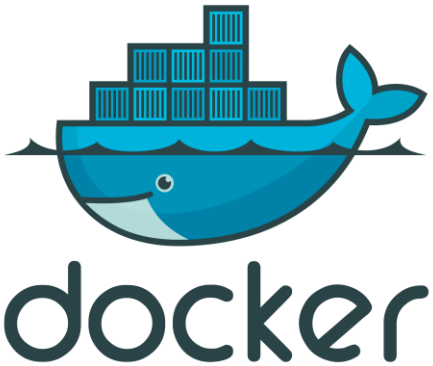
```
@BeforeAll
static void setUpProdotti() {…

@BeforeEach
void setUp() throws IOException {…

@AfterEach
void pulizia() {…
```

DEPLOYMENT (1/4)

Facendo uso delle GitHub Actions e di Docker Hub è stata realizzata una pipeline di Continuous Integration e Continuous Deployment.



Deployment (2/4)

Quindi, ogni volta che viene modificato il codice presente sul main branch, vengono eseguite in automatico le seguenti azioni:

DEPLOYMENT (3/4)

- GitHub Actions fornisce dei template per il workflow
- GitHub Marketplace per trovare Actions utili

```yaml
name: Java CI with Maven
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 11
      uses: actions/setup-java@v3
      with:
        java-version: '11'
        distribution: 'adopt'
        cache: maven
    - name: Build with Maven
      run: mvn -B package --file pom.xml
      working-directory: ./isaCalcAgri
    - name: Build and Push Docker Image
      uses: mr-smithers-excellent/docker-build-push@v6
      with:
        image: bressa98/progetto_isa
        tags: v1, latest
        dockerfile: isaCalcAgri/Dockerfile
        registry: docker.io
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}
```

DEPLOYMENT (4/4)


```
FROM openjdk:11-jdk

WORKDIR /app

# Cartella per volume
RUN mkdir /app/data

COPY isaCalcAgri/target/isaCalcAgri-1.0-SNAPSHOT.jar /app

CMD ["java", "-cp", "/app/isaCalcAgri-1.0-SNAPSHOT.jar", "it.isa.progetto.Main"]
```

Nel Dockerfile la cartella di lavoro è /app ma viene creata anche la cartella /app/data per il salvataggio del file di testo creato dall'applicazione.

In questo modo è possibile utilizzare un Volume Docker nella cartella /app/data in modo da mantenere i dati in modo persistente, anche dopo la distruzione del container o dopo un aggiornamento dell'immagine Docker.

GRAZIE PER L'ATTENZIONE

Università degli Studi di Ferrara