Vacanze Romane - Spiegazione Tecnica del Progetto

Panoramica del Progetto

Vacanze Romane è un'applicazione Java per gestire prenotazioni di weekend turistici a Roma. Il sistema mi permette di: - Creare viaggi con hotel, attrazioni e guide turistiche - Salvare e caricare dati da file - Visualizzare statistiche aggregate - Generare report HTML

Nel progetto ho utilizzato il pattern **MVC** (Model-View-Controller) per separare logica di business, presentazione e controllo.

Perché ArrayList?

Cos'è ArrayList

`ArrayList` è una lista dinamica che può crescere o ridursi automaticamente. È come un array, ma senza limiti di dimensione fissati.

Dove lo uso

```
private ArrayList<WeekendTrip> trips;
```

Nel `WeekendTripRepository`, ho utilizzato ArrayList per memorizzare tutti i viaggi prenotati.

Perché l'ho scelto

1. Dimensione Dinamica

```
//  Con array normale:
WeekendTrip[] trips = new WeekendTrip[10]; // massimo 10 trip
// Cosa succede se arriva l'11° cliente? ERRORE!

//  Con ArrayList:
ArrayList<WeekendTrip> trips = new ArrayList<>();
trips.add(trip1);  // OK
trips.add(trip2);  // OK
trips.add(trip100); // OK - nessun limite!
```

2. Operazioni Semplificate

```
// Aggiungere un elemento
trips.add(trip);

// Ottenere il numero di elementi
```

```java
int count = trips.size();

// Verificare se è vuota
if (trips.isEmpty()) { ... }

// Iterare facilmente
for (WeekendTrip trip : trips) {
    // processa ogni trip
}
```

3. Caso d'uso reale Nel metodo `getAllTrips()` restituisco una **copia difensiva**:

```java
public ArrayList<WeekendTrip> getAllTrips() {
    return new ArrayList<>(trips); // copia per sicurezza
}
```

Questo impedisce modifiche non autorizzate alla lista interna.

Alternative considerate

- **Array normale**: troppo rigido, dimensione fissa
- **LinkedList**: più lenta nell'accesso per indice
- **ArrayList**: la mia scelta ottimale per accesso rapido e dimensione variabile

---

Perché HashMap?

Cos'è HashMap

`HashMap` è come un dizionario: associa **chiavi univoche** a **valori**. Ogni chiave porta a un valore specifico.

Sintassi: `HashMap<TipoChiave, TipoValore>`

Dove lo uso

```java
public HashMap<String, Integer> hotelCounts = new HashMap<>();
public HashMap<String, Integer> saturdayAttractionCounts = new HashMap<>();
public HashMap<String, Integer> sundayAttractionCounts = new HashMap<>();
public HashMap<String, Integer> guideCounts = new HashMap<>();
```

Nel DTO `StatisticsData`, ho utilizzato HashMap per contare quante volte ogni hotel/attrazione/guida viene scelto.

Esempio Pratico di Funzionamento

Immagino di processare 4 prenotazioni:

```
Trip 1: Hotel GRAND_HOTEL
Trip 2: Hotel GRAND_HOTEL
Trip 3: Hotel HOTEL_ROMA
Trip 4: Hotel GRAND_HOTEL
```

Processo di conteggio con getOrDefault():

```
// Trip 1
hotelCounts.put("GRAND_HOTEL", hotelCounts.getOrDefault("GRAND_HOTEL", 0) + 1);
// getOrDefault restituisce 0 (chiave non esiste)
// mappa: {"GRAND_HOTEL": 1}

// Trip 2
hotelCounts.put("GRAND_HOTEL", hotelCounts.getOrDefault("GRAND_HOTEL", 0) + 1);
// getOrDefault restituisce 1 (chiave esiste)
// mappa: {"GRAND_HOTEL": 2}

// Trip 3
hotelCounts.put("HOTEL_ROMA", hotelCounts.getOrDefault("HOTEL_ROMA", 0) + 1);
// getOrDefault restituisce 0 (nuova chiave)
// mappa: {"GRAND_HOTEL": 2, "HOTEL_ROMA": 1}

// Trip 4
hotelCounts.put("GRAND_HOTEL", hotelCounts.getOrDefault("GRAND_HOTEL", 0) + 1);
// getOrDefault restituisce 2
// mappa: {"GRAND_HOTEL": 3, "HOTEL_ROMA": 1}
```

Codice reale dal progetto

```
// WeekendTripRepository.java:234-244
for (WeekendTrip trip : trips) {
    hotelName = trip.getAffiliatedHotel().name();
    stats.hotelCounts.put(
        hotelName,
        stats.hotelCounts.getOrDefault(hotelName, 0) + 1
    );

    saturdayAttr = trip.getSaturdayVisit().name();
    stats.saturdayAttractionCounts.put(
        saturdayAttr,
        stats.saturdayAttractionCounts.getOrDefault(saturdayAttr, 0) + 1
    );
}
```

Vantaggi di HashMap

1. Accesso Ultra-Rapido (O(1))

```
// Trovare un valore è quasi istantaneo
int count = hotelCounts.get("GRAND_HOTEL"); // velocissimo!
```

2. Nessuna Duplicazione Le chiavi sono uniche. Ogni hotel appare una sola volta come chiave.

3. Dinamicità Non mi serve sapere in anticipo quanti hotel/attrazioni esistono:

```
HashMap<String, Integer> counts = new HashMap<>();
// Si espande automaticamente quando serve
```

4. Metodo getOrDefault() Perfetto per contatori:

```
// Se la chiave non esiste, parte da 0
int nuovoValore = counts.getOrDefault("CHIAVE", 0) + 1;
```

Alternative considerate

- **ArrayList di coppie**: più lenta (O(n) per ricerca)
- **Array paralleli**: scomodo e propenso a errori
- **HashMap**: la mia scelta ottimale per conteggi e lookup rapidi

---

Perché StringBuilder?

Cos'è StringBuilder

`StringBuilder` è una classe per costruire stringhe in modo **efficiente** e **mutabile**.

Problema con String normale

```
//  String normale (INEFFICIENTE)
String errors = "";
errors = errors + "Errore 1\n";  // crea oggetto String #1
errors = errors + "Errore 2\n";  // crea oggetto String #2
errors = errors + "Errore 3\n";  // crea oggetto String #3
// Risultato: 3 oggetti String creati in memoria!
```

Perché è inefficiente? Le String in Java sono **immutabili**: ogni concatenazione (`+`) crea un nuovo oggetto in memoria.

Soluzione con StringBuilder

```
//  StringBuilder (EFFICIENTE)
StringBuilder errors = new StringBuilder();
errors.append("Errore 1\n");  // modifica lo stesso oggetto
errors.append("Errore 2\n");  // modifica lo stesso oggetto
errors.append("Errore 3\n");  // modifica lo stesso oggetto
// Risultato: 1 solo oggetto StringBuilder in memoria!
```

Dove lo uso

Nel metodo `validate()` della classe `WeekendTrip`:

```
// WeekendTrip.java:79-106
public String validate() {
    StringBuilder errors = new StringBuilder();

    if (name == null || name.trim().isEmpty())
        errors.append("- Il nome non può essere vuoto\n");

    if (surname == null || surname.trim().isEmpty())
        errors.append("- Il cognome non può essere vuoto\n");

    if (date == null)
        errors.append("- La data non può essere nulla\n");
    else if (!date.isAfter(LocalDate.now()))
        errors.append("- La data deve essere nel futuro\n");

    if (saturdayVisit == null)
        errors.append("- L'attrazione del sabato non può essere nulla\n");

    // ... altri controlli ...

    return errors.toString(); // converte in String normale
}
```

Vantaggi di StringBuilder

1. Performance

- Concatena 7+ stringhe con **1 solo oggetto** in memoria
- Ideale per loop e costruzioni incrementali

2. Mutabilità

```
StringBuilder sb = new StringBuilder();
sb.append("A");
sb.append("B");
```

```java
sb.append("C");
// Lo stesso oggetto viene modificato
```

3. Metodi Utili

```java
// Aggiungere testo
sb.append("testo");

// Convertire in String
String result = sb.toString();

// Lunghezza
int len = sb.length();

// Verificare se vuoto
if (sb.length() == 0) { ... }
```

Quando usare StringBuilder

USO StringBuilder quando: - Concateno molte stringhe in un loop - Costruisco stringhe dinamicamente - Aggiungo stringhe condizionalmente (come nel validate)

NON mi serve StringBuilder per: - Concatenazione semplice: `"Hello " + "World"` (OK così) - Stringhe statiche

---

Pattern Architetturali

Repository Pattern

File: `WeekendTripRepository.java`

Responsabilità: - Gestisce la persistenza dei dati (file) - Operazioni CRUD (Create, Read, Update, Delete) - Generazione ID progressivi - Calcolo statistiche aggregate

Perché lo uso:

```java
// Il Service non sa se i dati sono su file, database o API
repository.addTrip(trip);
repository.getAllTrips();
repository.saveTrips();
```

Se domani volessi usare un database invece di file, modificherei solo il Repository!

DTO Pattern (Data Transfer Object)

File: `StatisticsData.java`

Cos'è: Un contenitore di dati puri senza logica di business.

```
public class StatisticsData {
    public HashMap<String, Integer> hotelCounts = new HashMap<>();
    public HashMap<String, Integer> saturdayAttractionCounts = new HashMap<>();
    public BigDecimal totalRevenue = BigDecimal.ZERO;
    public int totalTrips = 0;
}
```

Flusso dati:

```
Repository (calcola) → DTO (trasporta) → View (visualizza)
```

Vantaggi: - Trasporta dati tra layer senza esporre logica interna - Evita dipendenze circolari

MVC Pattern (Model-View-Controller)

Struttura:

```
Model (entities):    WeekendTrip, Attraction, Hotel, TourGuide
Controller:          WeekendTripService, WeekendTripRepository
View:                WeekendTripView
```

Separazione responsabilità: - **Model**: rappresenta i dati e la logica di dominio - **Controller**: gestisce il flusso dell'applicazione - **View**: si occupa della presentazione all'utente

---

Altre Scelte Tecniche

BigDecimal per Prezzi

```
BigDecimal totalPrice = getTotalPrice();
```

Perché non `double`? I `double` hanno errori di arrotondamento con i decimali:

```
double price = 0.1 + 0.2; // risultato: 0.30000000000000004
```

BigDecimal è preciso:

```
BigDecimal price = new BigDecimal("0.1").add(new BigDecimal("0.2"));
// risultato: 0.3
```

Enum per Costanti

```
public enum Attraction {
    COLOSSEO("Colosseo", new BigDecimal("16.00")),
    FONTANA_DI_TREVI("Fontana di Trevi", BigDecimal.ZERO);
}
```

Vantaggi: - Type-safety (solo valori validi) - Codice più leggibile - Centralizza i dati

LocalDate per Date

```
LocalDate date = LocalDate.parse("2026-06-15");
```

Vantaggi: - API moderna (Java 8+) - Immutabile (thread-safe) - Metodi utili: `isAfter()`, `isBefore()`, `plusDays()`

Riepilogo Scelte Tecniche

Struttura Dati	Motivo d'Uso	Dove
ArrayList	Lista dinamica di viaggi	Repository
HashMap	Conteggio rapido per statistiche	StatisticsData
StringBuilder	Concatenazione efficiente errori	WeekendTrip.validate()
BigDecimal	Precisione monetaria	Calcoli prezzi
Enum	Costanti type-safe	Attraction, Hotel, TourGuide
LocalDate	Gestione date moderna	Data viaggio

Complessità Algoritmica

ArrayList

- **Accesso per indice**: O(1) - istantaneo
- **Ricerca per valore**: O(n) - deve scorrere tutta la lista
- **Aggiunta in coda**: O(1) ammortizzato

HashMap

- **Inserimento**: O(1) - quasi istantaneo
- **Ricerca per chiave**: O(1) - quasi istantaneo
- **Iterazione**: O(n) - deve scorrere tutte le coppie

StringBuilder

- **Append**: O(1) ammortizzato
- **toString()**: O(n) - crea una copia della stringa
