📄 Eesercizi sulle Collections

✅ Struttura del progetto

```
JavaCollectionsPractice
├── pom.xml
└── src
    └── main
        └── java
            ├── esercizio1
            │   ├── MainEsercizio1.java
            │   └── NumeriDuplicati.java
            ├── esercizio2
            │   ├── MainEsercizio2.java
            │   └── ManipolaListe.java
            └── esercizio3
                ├── MainEsercizio3.java
                └── Indirizzario.java
```

⚙️ Configurazione Maven

Nel file `pom.xml` aggiungi la dipendenza per Logback:

```
<dependencies>
    <!-- Framework per il logging semplice e configurabile -->
    <dependency>
        <groupId>ch.qos.logback</groupId>
```

```
        <artifactId>logback-classic</artifactId>
        <version>1.4.14</version>
    </dependency>
</dependencies>
```

📚 Esercizio 1 – Gestire duplicati con HashSet

🪛 Motivazione scelta collection:

Usiamo **HashSet** perché questa struttura dati non permette duplicati e consente di identificare facilmente gli elementi ripetuti inseriti dall'utente.

📌 Classe NumeriDuplicati.java

Per creare questa classe in IntelliJ:

- Tasto destro sul package `esercizio1` → New → Java Class → nome: `NumeriDuplicati`

Codice spiegato con commenti dettagliati:

```
package esercizio1;

import java.util.HashSet;
import java.util.Set;

public class NumeriDuplicati {

    private final Set<Integer> numeri;
    private final Set<Integer> duplicati;

    // Costruttore inizializza i due HashSet vuoti
    public NumeriDuplicati() {
```

```
        this.numeri = new HashSet<>();
        this.duplicati = new HashSet<>();
    }

    public void aggiungiNumero(int num) {
        // Il metodo add restituisce false se l'elemento esiste già nell'HashSet
        if (!numeri.add(num)) {
            // Se add restituisce false, num è un duplicato e viene aggiunto al set duplicati
            duplicati.add(num);
        }
    }

    // restituisce i numeri duplicati inseriti
    public Set<Integer> getDuplicati() {
        return duplicati;
    }

    // restituisce tutti i numeri unici inseriti
    public Set<Integer> getNumeriUnici() {
        return numeri;
    }
}
```

> **Nota:**
> `if (!numeri.add(num))`
> è importante perché il metodo `add` degli `HashSet` ritorna `false` se l'elemento è già presente.
> Usandolo così, individuiamo subito i duplicati.

🟢 Classe MainEsercizio1.java

- Nel package `esercizio1` : New → Java Class → nome: `MainEsercizio1`

```java
package esercizio1;

import java.util.Scanner;

public class MainEsercizio1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        NumeriDuplicati gestioneNumeri = new NumeriDuplicati();

        System.out.print("Quanti numeri vuoi inserire? ");
        int n;
        try {
            n = Integer.parseInt(sc.nextLine());
        } catch (NumberFormatException e) {
            System.out.println("Devi inserire un numero intero valido.");
            return;
        }

        for (int i = 0; i < n; i++) {
            System.out.print("Inserisci numero #" + (i + 1) + ": ");
            try {
                int numero = Integer.parseInt(sc.nextLine());
                gestioneNumeri.aggiungiNumero(numero);
            } catch (NumberFormatException e) {
                System.out.println("Input non valido, riprova.");
                i--;
            }
        }

        System.out.println("Numeri duplicati inseriti: " + gestioneNumeri.getDuplicati());
        System.out.println("Numero di valori unici: " + gestioneNumeri.getNumeriUnici().size());
        System.out.println("Lista dei valori unici: " + gestioneNumeri.getNumeriUnici());
    }
}
```

📚 Esercizio 2 – Lavorare con ArrayList

🪛 Motivazione scelta collection:

Utilizziamo una **ArrayList** perché ci serve una lista ordinata che permetta facilmente l'accesso tramite indice e l'aggiunta veloce di elementi alla fine.

📌 Classe ManipolaListe.java

Per creare questa classe in IntelliJ:

- Tasto destro sul package `esercizio2` → New → Java Class → nome: `ManipolaListe`

```
package esercizio2;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class ManipolaListe {

    // restituisce lista ordinata di N interi casuali
    public static List<Integer> generaListaCasualeOrdinata(int n) {
        List<Integer> lista = new ArrayList<>();
        Random rand = new Random();

        for (int i = 0; i < n; i++) {
            lista.add(rand.nextInt(101)); // da 0 a 100
        }

        Collections.sort(lista);
```

```
        return lista;
    }

    // restituisce lista originale + inversa
    public static List<Integer> listaConInversione(List<Integer> listaOriginale) {
        List<Integer> risultato = new ArrayList<>(listaOriginale);
        List<Integer> inversa = new ArrayList<>(listaOriginale);
        Collections.reverse(inversa);
        risultato.addAll(inversa);
        return risultato;
    }

    // stampa elementi pari o dispari a seconda del booleano
    public static void stampaElementi(List<Integer> lista, boolean stampaPari) {
        for (int i = 0; i < lista.size(); i++) {
            if (stampaPari && i % 2 == 0)
                System.out.print(lista.get(i) + " ");
            else if (!stampaPari && i % 2 != 0)
                System.out.print(lista.get(i) + " ");
        }
        System.out.println();
    }
}
```

🟢 Classe MainEsercizio2.java

```
package esercizio2;

import java.util.List;
import java.util.Scanner;

public class MainEsercizio2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Inserisci la lunghezza della lista: ");
        int n = Integer.parseInt(sc.nextLine());

        List<Integer> listaCasuale = ManipolaListe.generaListaCasualeOrdinata(n);
        System.out.println("Lista casuale ordinata: " + listaCasuale);

        List<Integer> listaConInversa = ManipolaListe.listaConInversione(listaCasuale);
        System.out.println("Lista con inversione: " + listaConInversa);

        System.out.print("Stampare elementi pari? (true/false): ");
        boolean stampaPari = Boolean.parseBoolean(sc.nextLine());
        ManipolaListe.stampaElementi(listaConInversa, stampaPari);
    }
}
```

Ecco la parte mancante relativa all'**Esercizio 3** completa di spiegazioni dettagliate e commenti chiarificatori.

📚 Esercizio 3 – Gestire un indirizzario con HashMap

🪛 Motivazione scelta collection:

Utilizziamo una **HashMap** perché dobbiamo gestire coppie chiave-valore (nome-numero), il che consente un accesso rapido ai dati tramite la chiave (il nome).

📌 Classe Indirizzario.java

Per creare questa classe in IntelliJ:

- Tasto destro sul package `esercizio3` → New → Java Class → nome: `Indirizzario`

```java
package esercizio3;

import java.util.HashMap;
import java.util.Map;

public class Indirizzario {
    private final Map<String, String> contatti;

    // Costruttore inizializza una HashMap vuota
    public Indirizzario() {
        this.contatti = new HashMap<>();
    }

    // Inserisce o aggiorna un contatto nell'indirizzario
    public void aggiungiContatto(String nome, String numero) {
        contatti.put(nome, numero);
    }

    // Rimuove un contatto per nome e restituisce true se il contatto esisteva e viene rimosso
    public boolean rimuoviContatto(String nome) {
        return contatti.remove(nome) != null;
    }

    // Cerca un nome a partire dal numero di telefono (ricerca inversa)
    public String cercaNomePerNumero(String numero) {
        for (Map.Entry<String, String> entry : contatti.entrySet()) {
            if (entry.getValue().equals(numero))
                return entry.getKey();
        }
        return null;
    }

    // Restituisce il numero di telefono a partire dal nome
    public String cercaNumeroPerNome(String nome) {
        return contatti.get(nome);
```

```
    }

    // Stampa tutti i contatti presenti nell'indirizzario
    public void stampaContatti() {
        if (contatti.isEmpty()) {
            System.out.println("L'indirizzario è vuoto.");
            return;
        }
        System.out.println("Elenco dei contatti:");
        contatti.forEach((nome, numero) -> System.out.println(nome + ": " + numero));
    }
}
```

🟢 Classe MainEsercizio3.java

Per creare questa classe in IntelliJ:

- Tasto destro sul package `esercizio3` → New → Java Class → nome: `MainEsercizio3`

```
package esercizio3;

import java.util.Scanner;

public class MainEsercizio3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Indirizzario indirizzario = new Indirizzario();
        boolean continua = true;

        while (continua) {
            System.out.println("\n1. Aggiungi contatto");
            System.out.println("2. Rimuovi contatto");
            System.out.println("3. Cerca nome per numero");
```

```
System.out.println("4. Cerca numero per nome");
System.out.println("5. Stampa tutti i contatti");
System.out.println("6. Esci");

System.out.print("Scelta: ");
String scelta = sc.nextLine();

switch (scelta) {
    case "1":
        System.out.print("Nome: ");
        String nome = sc.nextLine();
        System.out.print("Numero: ");
        String numero = sc.nextLine();
        indirizzario.aggiungiContatto(nome, numero);
        System.out.println("Contatto aggiunto.");
        break;
    case "2":
        System.out.print("Nome da rimuovere: ");
        nome = sc.nextLine();
        if (indirizzario.rimuoviContatto(nome))
            System.out.println("Contatto rimosso.");
        else
            System.out.println("Contatto non trovato.");
        break;
    case "3":
        System.out.print("Numero da cercare: ");
        numero = sc.nextLine();
        nome = indirizzario.cercaNomePerNumero(numero);
        if (nome != null)
            System.out.println("Numero associato a: " + nome);
        else
            System.out.println("Numero non trovato.");
        break;
    case "4":
        System.out.print("Nome da cercare: ");
        nome = sc.nextLine();
```

```
                    numero = indirizzario.cercaNumeroPerNome(nome);
                    if (numero != null)
                        System.out.println("Il numero di " + nome + " è " + numero);
                    else
                        System.out.println("Nome non trovato.");
                    break;
                case "5":
                    indirizzario.stampaContatti();
                    break;
                case "6":
                    continua = false;
                    break;
                default:
                    System.out.println("Scelta non valida, riprova.");
            }
        }

        sc.close();
    }
}
```

🎯 Scelta delle Collection:

Collection	Perché la utilizziamo?	Uso principale nell'esercizio
HashSet	Evitare duplicati e avere performance veloci per inserimento e ricerca.	Identificare numeri duplicati rapidamente.
ArrayList	Permette accesso rapido tramite indice, mantiene ordine di inserimento, permette duplicati.	Ordinare, invertire e stampare liste di numeri.

Collection	Perché la utilizziamo?	Uso principale nell'esercizio
HashMap	Accesso immediato tramite chiave univoca, associazione chiave-valore.	Gestione contatti: ricerca veloce di numeri telefonici per nome e viceversa.