Documento Riassuntivo: Week - 2. Collezioni, Stream, Lambda, StringBuilder e Gestione File in Java

1. Le Collection in Java

1.1. Creazione di Collezioni Immutabili con List.of e Set.of

Java 9 introduce metodi statici per creare collezioni immutabili:

```
// Creazione di una lista immutabile
List<String> lista = List.of("elemento1", "elemento2", "elemento3");

// Creazione di un insieme (Set) immutabile
Set<String> insieme = Set.of("elemento1", "elemento2", "elemento3");
```

> **Nota:** Le collezioni create con questi metodi non possono essere modificate (aggiunte o rimosse).

1.2. Tipi di Set: HashSet, LinkedHashSet e TreeSet

- **HashSet:**
  - **Caratteristiche:** Non garantisce l'ordine degli elementi.
  - **Uso:** Ottimo per operazioni di ricerca e inserimento veloci.
  - **Esempio:**

    ```
    Set<String> hashSet = new HashSet<>(Set.of("B", "A", "C"));
    ```

- **LinkedHashSet:**
  - **Caratteristiche:** Mantiene l'ordine di inserimento.
  - **Uso:** Quando l'ordine degli elementi è importante.
  - **Esempio:**

```
Set<String> linkedHashSet = new LinkedHashSet<>(Set.of("B", "A", "C"));
```

- **TreeSet:**
  - **Caratteristiche:** Mantiene gli elementi ordinati secondo l'ordine naturale (o un comparatore specifico).
  - **Uso:** Quando serve avere una visualizzazione ordinata degli elementi.
  - **Esempio:**

```
Set<String> treeSet = new TreeSet<>(Set.of("B", "A", "C"));
```

1.3. Creazione di Collezioni Mutabili con ArrayList

Per avere una collezione modificabile (mutabile), si utilizza la classe **ArrayList**:

```
// Creazione di una lista mutabile vuota
List<String> listaMutable = new ArrayList<>();

// Aggiunta di elementi
listaMutable.add("elemento1");
listaMutable.add("elemento2");
listaMutable.add("elemento3");

// Modifica o rimozione degli elementi
listaMutable.set(0, "nuovoElemento1");
listaMutable.remove("elemento2");
```

```
// Creazione di una lista mutabile a partire da una collezione esistente
List<String> listaImmutabile = List.of("elemento1", "elemento2", "elemento3");
List<String> listaMutableDaImmutabile = new ArrayList<>(listaImmutabile);
listaMutableDaImmutabile.add("elemento4");
```

> **Nota:** Con ArrayList puoi modificare la collezione dinamicamente, a differenza delle collezioni immutabili.

2. Streams e Lambda Expressions

2.1. Cos'è uno Stream?

Uno **Stream** è una sequenza di elementi che consente di eseguire operazioni funzionali (filtraggio, mapping, raggruppamento, ecc.) in modo dichiarativo.

2.2. Lambda Expressions

Le **lambda** sono espressioni compatte che rappresentano una funzione anonima.
Esempio:

```
List<Integer> numeri = List.of(1, 2, 3, 4, 5);
// Filtra i numeri pari e somma i risultati
int sommaPari = numeri.stream()
    .filter(n -> n % 2 == 0)
    .mapToInt(n -> n)
    .sum();
System.out.println("Somma numeri pari: " + sommaPari);
```

2.3. Funzioni di Raggruppamento (Collectors)

Utilizzando il metodo **Collectors.groupingBy** è possibile raggruppare gli elementi in base a una chiave.
Esempio:

```
Map<Boolean, List<Integer>> partizione = numeri.stream()
    .collect(Collectors.partitioningBy(n -> n % 2 == 0));
```

> **Method Reference (::):**
> Ad esempio, `String::toUpperCase` è una forma abbreviata per:
>
> ```
> s -> s.toUpperCase()
> ```
>
> Questo rende il codice più leggibile e conciso.

3. StringBuilder

3.1. Cos'è e Perché Usarlo

Lo **StringBuilder** è una classe che permette di costruire e modificare stringhe in modo efficiente.

Vantaggi:

- Evita la creazione di numerosi oggetti String quando si concatenano molte parti.
- È particolarmente utile all'interno di cicli o per la creazione di output complessi.

3.2. Esempio di Utilizzo

```
StringBuilder sb = new StringBuilder();
sb.append("Nome: ").append("Mario");
```

```
sb.append(", Età: ").append(30);
String risultato = sb.toString();
System.out.println(risultato);  // Output: Nome: Mario, Età: 30
```

4. Gestione dei File con Apache Commons IO

4.1. Introduzione

La libreria **Apache Commons IO** offre metodi utili per leggere e scrivere file senza doversi preoccupare di gestire manualmente stream e buffer.

4.2. Scrivere un File

```
import org.apache.commons.io.FileUtils;
import java.io.File;
import java.nio.charset.StandardCharsets;
import java.io.IOException;

String contenuto = "Questo è un esempio di testo.";
try {
    FileUtils.writeStringToFile(new File("output.txt"), contenuto, StandardCharsets.UTF_8);
} catch (IOException e) {
    e.printStackTrace();
}
```

4.3. Leggere un File

```
try {
    String contenutoLetto = FileUtils.readFileToString(new File("output.txt"), StandardCharsets.UTF_8);
    System.out.println(contenutoLetto);
```

```
} catch (IOException e) {
    e.printStackTrace();
}
```

5. Serializzazione con Delimitatori (@ e #)

5.1. Concetto di Serializzazione

La serializzazione consiste nel trasformare un oggetto in una stringa (o in un altro formato) per poterlo salvare su file o trasmettere.

Delimitatori usati in questo esempio:

- `@` : separa i campi di un record (es. nome, età, ecc.).
- `#` : separa i record (ossia, i diversi oggetti).

5.2. Esempio di Serializzazione

Immaginiamo di voler serializzare una lista di oggetti di tipo `Persona` :

```
class Persona {
    String nome;
    int eta;

    public Persona(String nome, int eta) {
        this.nome = nome;
        this.eta = eta;
    }

    @Override
    public String toString() {
```

```
        // Utilizza "@" per separare i campi
        return nome + "@" + eta;
    }
}

// Creazione di una lista di persone
List<Persona> persone = List.of(new Persona("Mario", 30), new Persona("Luigi", 25));

// Serializzazione: concatenazione di record usando "#" come delimitatore
StringBuilder sb = new StringBuilder();
for (Persona p : persone) {
    sb.append(p.toString()).append("#");
}
// Rimuove l'ultimo delimitatore "#"
if (sb.length() > 0) {
    sb.setLength(sb.length() - 1);
}

String datiSerializzati = sb.toString();
System.out.println(datiSerializzati);
// Output atteso: "Mario@30#Luigi@25"
```

5.3. Come Deserializzare

Per leggere i dati:

- **Step 1:** Dividi la stringa in record utilizzando il delimitatore `#`.
- **Step 2:** Per ogni record, usa `@` per separare i campi e ricostruire l'oggetto.

```
String[] records = datiSerializzati.split("#");
for (String record : records) {
    String[] campi = record.split("@");
    String nome = campi[0];
    int eta = Integer.parseInt(campi[1]);
```

```
    // Ricostruisci l'oggetto Persona
    Persona persona = new Persona(nome, eta);
    System.out.println("Persona: " + persona);
}
```

6. Tabella Riassuntiva

Argomento	Descrizione	Esempio / Caratteristiche
List.of / Set.of	Metodi statici per creare collezioni immutabili.	`List.of("a","b")` `Set.of("a", "b")`
ArrayList	Collezione mutabile: consente di aggiungere, modificare o rimuovere elementi dinamicamente.	`new ArrayList<>()` `add()` , `remove()` , `set()`
HashSet	Set non ordinato, ottimo per ricerche e inserimenti veloci.	`new HashSet<>(List.of("a", "b"))`
LinkedHashSet	Set che mantiene l'ordine di inserimento.	`new LinkedHashSet<>(List.of("a", "b"))`
TreeSet	Set ordinato in base all'ordine naturale o con comparatore.	`new TreeSet<>(List.of("a", "b"))`
Streams e Lambda	Sequenza di elementi per operazioni funzionali (filtraggio, mapping, raggruppamento, ecc.) mediante lambda e method reference.	`stream().filter(n -> n % 2 == 0)` `String::toUpperCase`
Collectors.groupingBy / PartitioningBy	Raggruppamento o partizionamento degli elementi di uno stream in base a una funzione chiave o predicato.	`collect(Collectors.groupingBy(...))` `collect(Collectors.partitioningBy(...))`

Argomento	Descrizione	Esempio / Caratteristiche
StringBuilder	Classe per concatenare stringhe in modo efficiente, evitando la creazione di numerosi oggetti String.	`new StringBuilder().append("testo")`
Apache Commons IO	Libreria per leggere e scrivere file in modo semplificato, gestendo stream e buffer in background.	`FileUtils.writeStringToFile(file, string, charset)` `FileUtils.readFileToString(file, charset)`
Serializzazione con delimitatori	Trasforma oggetti in stringhe utilizzando delimitatori per separare campi (`@`) e record (`#`), ideale per formati didattici e di base.	Esempio: `"Mario@30#Luigi@25"` Split con `@` e `#` per deserializzare