Inizializzazione e cleanup

Due dei problemi maggiori all'interno della programmazione sono inizializzazione e il cleanup:

- **Inizializzazione**: Uno dei problemi comuni è dimenticarsi di inizializzare variabili. Questo può portare a **errori nel codice, come riferimenti a valori nulli o non definiti**. I costruttori sono utili proprio per garantire che gli attributi di un oggetto siano inizializzati correttamente al momento della creazione.
- **Cleanup**: Riguarda la gestione della memoria. È importante rimuovere o deallocare risorse non più necessarie per evitare perdite di memoria. In Java, la gestione della memoria è principalmente automatizzata tramite il garbage collection, ma è comunque importante fare attenzione a non lasciare oggetti non referenziati che possono occupare memoria inutilmente.

Costruttori e this

I costruttori sono metodi speciali utilizzati per creare oggetti. Un costruttore ha lo stesso nome della classe e può avere parametri per inizializzare gli attributi dell'oggetto. Ecco un esempio:

```
public class Persona {
    private String nome;

    // Costruttore
    public Persona(String nome) {
        this.nome = nome; // 'this.nome' si riferisce all'attributo
dell'oggetto
    }
}
```

`this` serve a riferirsi agli attributi e ai metodi dell'oggetto corrente. Questo è particolarmente utile in situazioni in cui i nomi delle variabili locali (come i parametri del costruttore) coincidono con i nomi degli attributi della classe. Utilizzando `this`, chiarisci che stai parlando dell'attributo dell'oggetto e non del parametro.

Overloaded Constructor

Gli **overloaded constructor** sino costruttori multipli all'interno di una classe tutti con lo stesso nome, ma con diversi parametri.

Signature

Combinazione del nome della classe e deti parametri è detta signature.

Nella situazione in cui volessimo creare un oggetto senza una determinata caratteristica se avessimo solo un costruttore non lo potremmo fare. (Il costruttore voule che gli passiamo tutti gli argomenti), quindi possiamo utilizzare altri costruttori con delle signature per costruire oggetti specifici.

```
public class pizza {

    String bread;
    String Sauce;
    String cheese;
    String topping;

    pizza(String bread, String Sauce, String cheese, String topping){

        this.bread = bread;
        this.Sauce = Sauce;
        this.cheese = cheese;
        this.topping = topping;
    }

    //Overloaded constructor
    pizza(String bread, String Sauce, String cheese){

        this.bread = bread;
        this.Sauce = Sauce;
        this.cheese = cheese;
    }


    //Overloaded constructor
    pizza(String bread, String Sauce){

        this.bread = bread;
        this.Sauce = Sauce;
    }

    //Overloaded constructor
    pizza(String bread){

        this.bread = bread;
```

```
    }

        //Overloaded contructor
    pizza(){
    }

}
```