1. Dipendenza: Definizione e Concetti Fondamentali

1.1 Cos'è una dipendenza?

Una **dipendenza** è lo stato qualitativo di essere influenzato, determinato o soggetto a un altro componente. In termini pratici, un componente A dipende da un componente B quando cambiamenti in B possono richiedere modifiche in A.

1.2 Tipi di cambiamenti

- **Cambiamenti interni**: modifiche all'implementazione che non alterano l'interfaccia pubblica
- **Cambiamenti esterni**: modifiche all'interfaccia o al comportamento estrinseco

1.3 Misura della dipendenza

La dipendenza misura la **probabilità di propagazione di cambiamenti** tra componenti dipendenti. Più forte è la dipendenza, maggiore è la probabilità che un cambiamento si propaghi.

---

2. Accoppiamento (Coupling)

2.1 Definizione

L'**accoppiamento** è la misura quantitativa del grado di dipendenza tra componenti.

- **Accoppiamento stretto (tightly-coupled)**: alta probabilità di propagazione di cambiamenti
- **Accoppiamento debole (loosely-coupled)**: bassa probabilità di propagazione di cambiamenti

2.2 Principio fondamentale

> *"Le dipendenze tra componenti devono essere minimizzate, rendendo i componenti loosely coupled"*
> — Gang of Four

Obiettivo: essere liberi di modificare un componente senza introdurre bug in altri componenti.

---

3. Tipi di Dipendenze in OOP

Esistono cinque principali tipi di dipendenze, ordinati dal più debole al più forte:

1. **Dependency (relazione)**
2. **Association**
3. **Aggregation**
4. **Composition**
5. **Inheritance**

---

4. Dependency (Relazione)

4.1 Caratteristiche

È la **forma più debole** di dipendenza:

- **Limitata nel tempo**: dura solo l'esecuzione di un metodo
- **Limitata nel codice condiviso**: solo l'interfaccia (signature dei metodi)

4.2 Esempio pratico

```
class Stampante {
    public void stampa(String testo) {
        System.out.println(testo);
    }
}

class DocumentoEditor {
    // Dependency: Stampante appare solo come parametro
    public void stampaDocumento(Stampante stampante, String contenuto) {
        stampante.stampa(contenuto);  // uso limitato a questo metodo
    }

    // Dependency: Stampante appare solo come valore di ritorno
    public Stampante ottieniStampanteDefault() {
        return new Stampante();
    }
}
```

Analisi del codice condiviso:

- Intervallo di dipendenza: limitato all'esecuzione del singolo metodo
- Codice condiviso: solo la signature del metodo `stampa()`

---

5. Association

5.1 Caratteristiche

Una classe contiene un **riferimento permanente** a un oggetto di un'altra classe:

- **Durata**: copre l'intera vita dell'oggetto
- **Impatto sulla costruzione**: l'oggetto dipendente deve essere fornito alla creazione
- **Codice condiviso**: virtualmente tutti i comportamenti della classe associata

5.2 Esempio pratico

```
class Motore {
    public void avvia() {
        System.out.println("Motore avviato");
    }

    public void accelera(int velocita) {
        System.out.println("Accelerando a " + velocita + " km/h");
    }

    public void ferma() {
        System.out.println("Motore fermato");
    }
}

class Automobile {
    private Motore motore;  // riferimento permanente

    // Association: il motore viene iniettato
    public Automobile(Motore motore) {
        this.motore = motore;
    }

    public void parti() {
        motore.avvia();
        motore.accelera(50);
    }
```

```
    public void fermati() {
        motore.ferma();
    }
}

// Utilizzo
Motore motoreBenzina = new Motore();
Automobile miaAuto = new Automobile(motoreBenzina);
```

Analisi del codice condiviso:

- Intervallo di dipendenza: tutta la vita dell'oggetto `Automobile`
- Codice condiviso: tutte le signature dei metodi di `Motore`

---

6. Aggregation e Composition

6.1 Caratteristiche comuni

Un tipo **possiede** l'altro, aggiungendo:

- Responsabilità di **creazione**
- Responsabilità di **eliminazione**

6.2 Differenza tra Aggregation e Composition

- **Aggregation**: il componente può essere condiviso con altri oggetti
- **Composition**: il componente è esclusivo e non condivisibile

6.3 Esempio: Composition

```
class Processore {
    private String modello;
    private int frequenza;

    public Processore(String modello, int frequenza) {
        this.modello = modello;
        this.frequenza = frequenza;
    }

    public void elabora() {
        System.out.println("Elaborazione con " + modello);
    }
```

```
}

class Computer {
    private Processore processore;  // composition: processore è parte
esclusiva

    public Computer() {
        // Computer DEVE sapere come costruire un Processore
        this.processore = new Processore("Intel i7", 3600);
    }

    public void accendi() {
        processore.elabora();
    }
}
```

6.4 Esempio: Aggregation

```
class Studente {
    private String nome;

    public Studente(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}

class Corso {
    private List<Studente> studenti;  // aggregation: studenti condivisi

    public Corso() {
        this.studenti = new ArrayList<>();
    }

    public void aggiungiStudente(Studente studente) {
        studenti.add(studente);  // studente esiste indipendentemente dal
corso
    }
}
```

Analisi del codice condiviso:

- Intervallo di dipendenza: tutta la vita dell'oggetto
- Codice condiviso: **anche il processo di costruzione** dell'oggetto dipendente

---

7. Inheritance (Ereditarietà)

7.1 Caratteristiche

È la **forma più forte** di dipendenza:

- Ereditarietà e riuso di tutto il codice non privato
- Ogni cambiamento al parent può impattare i figli
- Si riferisce all'*implementation inheritance*, non al subtyping

7.2 Esempio pratico

```
class Veicolo {
    protected int velocitaMassima;
    protected String targa;

    public Veicolo(String targa, int velocitaMassima) {
        this.targa = targa;
        this.velocitaMassima = velocitaMassima;
    }

    public void muovi() {
        System.out.println("Il veicolo si muove");
    }

    protected void verificaTarga() {
        System.out.println("Targa: " + targa);
    }
}

class Autobus extends Veicolo {
    private int numeroPosti;

    public Autobus(String targa, int velocitaMassima, int numeroPosti) {
        super(targa, velocitaMassima);  // dipendenza dal costruttore parent
        this.numeroPosti = numeroPosti;
    }

    public void caricaPasseggeri() {
```

```
        verificaTarga();  // dipendenza da metodo protected del parent
        System.out.println("Caricamento di max " + numeroPosti + "
passeggeri");
    }

    @Override
    public void muovi() {
        super.muovi();  // dipendenza dall'implementazione del parent
        System.out.println("Autobus in movimento");
    }
}
```

Analisi del codice condiviso:

- Intervallo di dipendenza: tutta la vita della classe
- Codice condiviso: **tutto il codice non privato** del parent (attributi, metodi, costruttori)
