Inversion of Control

Cosa imparerai

- Cos’è l’Inversion of Control e come si implementa

Inversion of Control

L'**Inversion of Control** (IoC) è un principio di progettazione dell'ingegneria del software mediante il quale il controllo di oggetti o porzioni di un programma viene trasferito al container o al framework.
È utilizzato molto nel contesto della programmazione orientata agli oggetti.

Facciamo un esempio per spiegare come funziona.

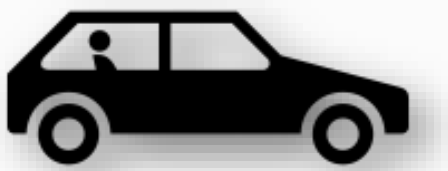
Supponiamo di guidare l'automobile per andare a lavoro.
Questo vuol dire che noi controlliamo l'auto e **mentre guidiamo l'auto non possiamo fare altro**.

Inversion of Control

Invertiamo lo scenario…

Invece di guidare l’auto noleggiamo un taxi.

In questo caso non siamo noi a guidare la macchina, lasciamo che sia l’autista a farlo.
In questo in modo noi potremo concentrarci sul nostro lavoro principale.



Questo si chiama inversione del controllo, da noi al tassista.

Inversion of Control

Facciamo un esempio nel mondo software…

Se sviluppiamo un programma, abbiamo due strade:

- ❑ Controllare tutto il flusso

```
public class Flusso {
    @SuppressWarnings("resource")
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.println("Inserisci il tuo nome: ");
        String nome = in.nextLine();

        System.out.println("Inserisci il tuo cognome: ");
        String cognome = in.nextLine();

        System.out.println("Inserisci la città di provenienza: ");
        String indirizzo = in.nextLine();

        System.out.println("Benvenuto nel corso, " + nome + " " + cognome + " proveniente da " + indirizzo);
    }
}
```

Il flusso, inserimento dati, visualizzazione messaggio di benvenuto, uscita dal programma, ecc…è gestito interamente nel metodo main() della classe

Inversion of Control

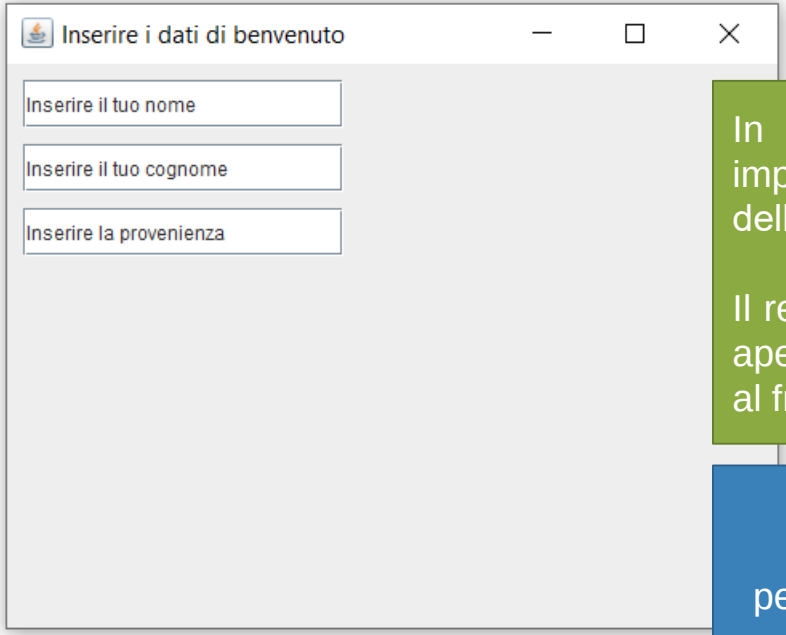
- Lasciare che una parte del flusso venga gestito da altre componenti

```java
public class FlussoGUI {
    public static void main(String[] args) {
        JFrame finestra = new JFrame();
        finestra.setTitle("Inserire i dati di benvenuto");
        finestra.setSize(400,400);
        finestra.setLayout(null);
        finestra.setVisible(true);

        JTextField nome = new JTextField();
        nome.setBounds(10,10, 200,30);
        nome.setText("Inserire il tuo nome");
        finestra.add(nome);

        JTextField cognome = new JTextField();
        cognome.setBounds(10,50, 200,30);
        cognome.setText("Inserire il tuo cognome");
        finestra.add(cognome);

        JTextField provenienza = new JTextField();
        provenienza.setBounds(10,90, 200,30);
        provenienza.setText("Inserire la provenienza");
        finestra.add(provenienza);
    }
}
```



In questo caso dobbiamo preoccuparci di implementare la logica applicativa a seguito dell'inserimento dei dati da parte dell'utente.

Il resto del controllo del flusso applicativo (ad es. apertura e chiusura della finestra) è demandato al framework (nell'esempio a Swing).

Questo comportamento viene detto **Inversion of Control** perché il controllo viene appunto spostato dallo sviluppatore al framework.

Inversion of Control

Facciamo un esempio…

Creiamo un software per la **gestione di una biblioteca**.

La nostra classe *Biblioteca*, si occupa di gestire la disponibilità, la prenotazione e la consegna dei libri.

```
public class Biblioteca {
    private BibliotecaService service;

    public Biblioteca() {
        super();

        service = new BibliotecaServiceImpl();
    }

    public List<Libro> getLibriDisponibili() {
        return service.getLibriDisponibili();
    }
}
```

Per accedere ai dati salvati su database, utilizza il servizio *BibliotecaService* che, attraverso la sua implementazione, si interfaccia con il database e gestisce il flusso delle informazioni.

Per gestire la dipendenza della classe *Biblioteca* dal servizio, nel costruttore della classe *Biblioteca* abbiamo inserito l'istruzione che crea l'istanza della classe *BibliotecaServiceImpl*.

Inversion of Control

L'Inversion of Control può essere ottenuta utilizzando vari design pattern:

- Service Locator pattern
- Factory pattern
- Dependency Injection (DI) pattern

```java
public class DependencyManager {
    public static BibliotecaService getBibliotecaService() {
        return new BibliotecaServiceImpl();
    }

}
```

```java
public class Biblioteca {
    public Biblioteca() {
        super();
    }

    public List<Libro> getLibriDisponibili() {
        return DependencyManager.getBibliotecaService().getLibriDisponibili();
    }
}
```

In questo esempio abbiamo utilizzato il **Factory pattern** che si occupa di creare le istanze delle classi.

Inversion of Control

Spring implementa l'Inversion of Control utilizzando il pattern Dependency Injection per la gestione delle dipendenze!

Di cosa abbiamo parlato in questa lezione

- Cos’è l’Inversion of Control e come si implementa