Programma del corso

- **Titolo:** Object Oriented Programming: Applicazioni Java Avanzate
- **Durata Totale:** 40 Ore (15 Teoria / 25 Lab)
- **Prerequisiti:** Java Base + Concetti OOP (Classi, Ereditarietà, Polimorfismo).
- **Obiettivo Finale:** Sviluppare un'applicazione Client/Server con interfaccia grafica (es. una chat multi-utente o un gestionale remoto).

Modulo 1: Consolidamento e Input/Output Avanzato (6 Ore)

Questo modulo serve a scaldare i motori e a gestire la persistenza dei dati, fondamentale prima di inviarli in rete.

Teoria (2 ore):

- Breve recap: Collections Framework (List, Map, Set) e gestione delle Eccezioni (try-catch-finally).
- **Il sistema di I/O in Java:**
  - Differenza tra Byte Streams (`InputStream/OutputStream`) e Character Streams (`Reader/Writer`).
  - Il pattern Decorator nell'I/O (es. `BufferedReader` dentro `FileReader`).
  - **Serializzazione:** Come salvare lo stato di un oggetto (`Serializable`).
  - Gestione dei percorsi e manipolazione dei file (`java.io.File` o cenni a `java.nio.Path`).

Laboratorio (4 ore):

- **Esercizio 1:** Creazione di un semplice log system che scrive su file di testo.
- **Esercizio 2 (Il "Blocco Note"):** Creare una classe che legge un file CSV, lo trasforma in oggetti Java (es. lista di `Utenti`), permette di aggiungerne uno e risalva il tutto.
- **Esercizio 3:** Serializzare e deserializzare un oggetto complesso su disco.

Modulo 2: Interfacce Grafiche (GUI) con Swing (10 Ore)

Ho scelto Swing perché è nativamente supportato dal GUI Builder di Netbeans (Matisse) e Eclipse (WindowBuilder), permettendo di creare interfacce velocemente senza troppo codice boilerplate, ideale per un corso di 40 ore.

Teoria (4 ore):

- Concetti base delle GUI: Contenitori (`JFrame`, `JPanel`) e Componenti (`JButton`, `JLabel`, `JTextField`, `JTextArea`).
- **Layout Managers:** Capire come disporre gli elementi (`BorderLayout`, `FlowLayout`, `GridLayout`).
- **Event-Driven Programming:** Il modello a eventi di Java.
    - Interfacce Listener (`ActionListener`, `MouseListener`).
    - Classi anonime e Lambda expressions per gestire gli eventi (Java 8+).

Laboratorio (6 ore):

- **Esercizio 1:** Disegnare una calcolatrice o un form di registrazione utente usando il GUI Builder dell'IDE.
- **Esercizio 2:** Gestire gli eventi: far succedere qualcosa quando si clicca "Invia" (validazione input, messaggi di dialogo `JOptionPane`).
- **Esercizio 3:** Collegare il Modulo 1 al Modulo 2: creare una GUI che salva i dati inseriti su un file di testo.

Modulo 3: Multithreading e Concorrenza (8 Ore)

Necessario per non "congelare" l'interfaccia grafica quando l'applicazione lavora o attende la rete.

Teoria (3 ore):

- Processi vs Thread.
- La classe `Thread` e l'interfaccia `Runnable`.
- Ciclo di vita di un Thread (New, Runnable, Running, Blocked, Terminated).
- **Problemi della concorrenza:** Race conditions e l'uso della keyword `synchronized`.
- **Thread e GUI:** Il concetto di *Event Dispatch Thread (EDT)*. Perché non bisogna mai bloccare l'interfaccia grafica.

Laboratorio (5 ore):

- **Esercizio 1:** Creare due thread che stampano numeri in console contemporaneamente.
- **Esercizio 2 (Cronometro):** Creare una GUI con un cronometro che avanza. Se non si usa un thread separato, l'interfaccia si blocca. Implementazione corretta con Thread.
- **Esercizio 3:** Simulazione "Produttore/Consumatore" semplificata per capire l'accesso sincronizzato alle risorse.

Modulo 4: Architetture di Rete e Socket (12 Ore)

Focus: Non solo "come scrivere il codice", ma "come progettare il sistema distribuito".

Teoria (5 Ore totali):

- **Parte A: Architettura Client-Server e le sue implicazioni (2.5 Ore)**
  - **Il concetto di "Ruolo":** Chi fa cosa? Il Server come fornitore di risorse (passivo/in ascolto) e il Client come richiedente (attivo).
  - **Stateful vs Stateless:**
    - Il server deve ricordarsi chi sono? (Es. Login effettuato vs richiesta HTTP semplice).
    - Implicazioni sulla memoria del server quando ci sono 1000 client connessi.
  - **Progettare un Protocollo Applicativo:**
    - Non basta stabilire la connessione, bisogna decidere una "lingua".
    - Esempio: "Se invio 'LOGIN|user|pass', il server risponde 'OK' o 'ERR'".
  - **Le 8 fallacie dei sistemi distribuiti (Semplificate):**
    - La rete *non* è affidabile (gestire `SocketTimeoutException`).
    - La latenza *non* è zero (perché l'interfaccia grafica non deve aspettare la rete -> collegamento al Modulo 3 sui Thread).
    - La larghezza di banda *non* è infinita.
- **Parte B: Implementazione Tecnica in Java (2.5 Ore)**
  - Le classi `java.net.ServerSocket` e `java.net.Socket`.
  - Il flusso di comunicazione: `InputStream` e `OutputStream` su Socket.

    - Gestione della concorrenza lato Server:
        - Il ciclo `while(true)` del server.
        - Accettare una connessione (`accept()`) e delegarla subito a un Thread separato (il pattern "Thread-per-Client").

Laboratorio (7 Ore):

- **Esercizio 1 (Protocollo a mano):** Utilizzare "Telnet" o "Putty" come client per collegarsi a un semplice Server Java scritto da loro. Questo fa capire che il Client non deve per forza essere scritto in Java, basta che rispetti il protocollo TCP.
- **Esercizio 2 (Server Multi-thread):** Scrivere un server che accetta più connessioni contemporaneamente e assegna a ciascuna un ID univoco.
- **Esercizio 3 (L'Applicazione Finale):**
    - Sviluppo della parte di Networking dell'applicazione finale.
    - Implementazione di un semplice protocollo di messaggistica (es. stringhe separate da caratteri speciali).
    - Gestione della disconnessione improvvisa del client (il server non deve crashare se il client chiude la finestra).

I **3 errori classici** dei junior developer da prevenire:

1. **Server Monolitico:** Scrivono server che gestiscono un solo client alla volta (se Alice è connessa, Bob deve aspettare).
2. **Protocolli "a caso":** Inviano dati senza una struttura, rendendo impossibile il parsing dall'altra parte.
3. **UI congelate:** Non capendo la latenza di rete, mettono le chiamate al server dentro il click del bottone senza usare un Thread, bloccando l'applicazione finché il server non risponde.

Modulo 5: Recap e Deployment (4 Ore)

Chiusura del cerchio e creazione dell'eseguibile.

Teoria (1 ore):

- Best practices finali.
- Come esportare il progetto: creare un file **JAR eseguibile**.
- Cenni a Java Web (giusto per dare contesto sul "dopo", es. Spring Boot, Servlet) ma senza approfondire, visto il limite di ore.

Laboratorio (3 ore):

- Finalizzazione del progetto del Modulo 4.
- Creazione del file `.jar` e test dell'applicazione fuori dall'IDE.
- Code review di gruppo sui progetti finali.

Strumenti Consigliati

1. **IDE:** Se vuoi facilitare la creazione delle GUI, **NetBeans** è ancora imbattibile per la didattica Swing grazie all'editor visuale drag-and-drop.
2. **JDK:** Java 17 o 21 (LTS).