ESERCITAZIONE DI PREPARAZIONE - Programmazione ad Oggetti in C#

Classe: 4ª
Tempo consigliato: 1 ora e 30 minuti
Tipo: Esercitazione guidata di preparazione alla verifica

OBIETTIVO DELL'ESERCITAZIONE

Questa esercitazione ti prepara di esercitarti su tutti gli argomenti fondamentali: **ereditarietà, interfacce, polimorfismo, collezioni generiche e GUI**.

ISTRUZIONI GENERALI

1. Creare un nuovo progetto **Windows Forms Application** in C# denominato `Esercitazione_Cognome`
2. Il progetto deve compilare ed eseguire senza errori
3. L'obiettivo è imparare, non completare velocemente

IL PROBLEMA

Una palestra moderna vuole digitalizzare la gestione dei propri corsi e iscrizioni. La palestra offre diverse tipologie di attività: corsi di gruppo (yoga, spinning, pilates), personal training individuali, abbonamenti in sala pesi e accesso alla piscina. Alcuni servizi richiedono una prenotazione obbligatoria, altri sono ad accesso libero.

Scenario di utilizzo

Il responsabile della palestra deve poter:

- Registrare nuovi servizi/corsi specificando le informazioni specifiche di ciascuno (orari, istruttore, posti disponibili, ecc.)
- Visualizzare tutti i servizi offerti e cercare per tipologia, istruttore o fascia oraria
- Gestire le prenotazioni: registrare quando un cliente prenota un corso e quando annulla la prenotazione
- Monitorare la disponibilità: sapere quanti posti liberi ci sono per ogni corso
- Ottenere statistiche: numero totale di prenotazioni attive, corsi più frequentati, ricavi previsti

Alcuni dettagli operativi:

- Ogni servizio ha caratteristiche comuni (codice, nome, costo, durata) e caratteristiche specifiche del suo tipo
- Non tutti i servizi richiedono prenotazione: la sala pesi è ad accesso libero, mentre i corsi di gruppo hanno posti limitati
- Ogni tipo di servizio prenotabile ha regole diverse (i corsi di gruppo hanno numero massimo partecipanti, il personal training è 1-a-1)
- La palestra vuole monitorare il fatturato previsto in base alle prenotazioni attive

Esempio concreto di utilizzo

Il responsabile accende il programma e:

1. Inserisce un nuovo corso di Yoga (livello principianti, lunedì 18:00-19:00, istruttore Maria Rossi, max 15 partecipanti, costo 12€)
2. Inserisce una sessione di Personal Training (istruttore Luca Bianchi, durata 60 min, costo 50€, 1-a-1)
3. Inserisce l'accesso alla sala pesi (libero, costo giornaliero 10€, nessuna prenotazione richiesta)
4. Visualizza tutti i servizi disponibili
5. Un cliente vuole prenotare il corso di Yoga: il responsabile registra la prenotazione con nome cliente e data
6. Cerca tutti i corsi tenuti dall'istruttore "Maria Rossi"
7. Visualizza le statistiche: numero totale prenotazioni, corso più richiesto, fatturato previsto
8. Verifica quali corsi hanno posti quasi esauriti (meno di 3 posti liberi)

COSA DEVI FARE

Progetta e implementa una soluzione completa:

Architettura e Progettazione

- **Crea una gerarchia di classi** per rappresentare i diversi tipi di servizi offerti dalla palestra
- **Usa l'ereditarietà** per evitare di ripetere il codice comune
- **Definisci un'interfaccia** per i servizi che richiedono prenotazione
- **Applica il polimorfismo** per gestire servizi diversi in modo uniforme

Implementazione

- **Incapsula i dati**: attributi privati, accesso tramite proprietà o metodi
- **Costruttori multipli**: prevedi diversi modi per creare oggetti
- **Metodi virtual/override**: permetti personalizzazioni nelle classi derivate

- **Collezioni generiche**: usa List per i servizi e Dictionary per ricerche veloci (es. per codice servizio o istruttore)

Interfaccia Utente

- **Form Windows** con controlli per inserire nuovi servizi/corsi
- **Visualizzazione servizi** in ListBox o DataGridView
- **Funzioni di ricerca** (per istruttore, tipologia, orario)
- **Gestione prenotazioni** (prenota, annulla, visualizza disponibilità)
- **Statistiche** (totale prenotazioni, fatturato, corsi popolari, ecc.)

SUGGERIMENTI METODOLOGICI

Passo 1: Progetta le classi (20 minuti)

Prima di scrivere codice, rispondi a queste domande su carta:

- Quali classi mi servono? (almeno 3-4 tipi di servizi diversi)
- Cosa hanno in comune tutti i servizi? (attributi e metodi della classe base)
- Cosa differenzia un tipo di servizio dall'altro? (attributi specifici)
- Quali servizi richiedono prenotazione? (interfaccia)

Esempio di progettazione:

```
Classe base: ???
  - Attributi comuni: codice, nome, costo, durata
  - Metodi comuni: ???

Classi derivate: CorsoGruppo, PersonalTraining, SalaPesi, ...
  - Attributi specifici: ???
  - Implementano interfaccia prenotazione? Sì/No

Interfaccia: ???
  - Metodi: prenota, annullaPrenotazione, verificaDisponibilità, ...
```

Passo 2: Implementa le classi base (20 minuti)

Inizia creando:

1. La classe base con gli attributi comuni
2. L'interfaccia per la prenotazione
3. Almeno 2 classi derivate semplici

Testa subito: crea un Form minimale e prova a istanziare un oggetto, verifica che compili.

Passo 3: Aggiungi la gestione collezioni (20 minuti)

Crea una classe "Gestore" (o "GestorePalestra" o come preferisci) che:

- Contenga una `List<TuaClasseBase>` per tutti i servizi
- Abbia metodi per: aggiungi, visualizza tutti, cerca per istruttore
- Usi un `Dictionary<string, TuaClasseBase>` per ricerca per codice

Passo 4: Implementa l'interfaccia prenotazione (15 minuti)

Nelle classi che richiedono prenotazione:

- Aggiungi attributi per gestire le prenotazioni (nomeCliente, dataPrenotazione, postiDisponibili)
- Implementa tutti i metodi dell'interfaccia
- Usa `override` se vuoi personalizzare comportamenti

Passo 5: Crea l'interfaccia grafica (30 minuti)

Progetta un Form con:

- GroupBox per inserimento dati (TextBox per codice, nome, costo, istruttore, ecc.)
- ComboBox o RadioButton per scegliere il tipo di servizio
- Pulsanti: Aggiungi, Visualizza Tutto, Cerca, Statistiche, Prenota
- ListBox per visualizzare risultati
- Label per statistiche

Passo 6: Testa e migliora (15 minuti)

- Inserisci almeno 5-6 servizi di tipi diversi
- Prova tutte le funzionalità
- Correggi errori
- Aggiungi funzionalità se hai tempo

CHECKLIST DI AUTOVALUTAZIONE

Usa questa checklist per verificare di aver applicato tutti i concetti:

Classi e Ereditarietà

- [ ] Ho creato una classe base con attributi comuni
- [ ] Ho creato almeno 3 classi derivate
- [ ] Le classi derivate hanno attributi specifici
- [ ] Le classi derivate estendono correttamente la classe base
- [ ] Non ho duplicato codice inutilmente

Interfacce e Polimorfismo

- [ ] Ho definito un'interfaccia per un comportamento comune
- [ ] Almeno 2 classi implementano l'interfaccia
- [ ] Ho implementato tutti i metodi dell'interfaccia
- [ ] Ho usato `virtual` e `override` per personalizzare metodi
- [ ] Ho sfruttato il polimorfismo (es. List<ClasseBase> contenente oggetti di classi derivate)

Incapsulamento e Costruttori

- [ ] Gli attributi sono privati
- [ ] Ho creato proprietà (get/set) o metodi per accedere ai dati
- [ ] Ogni classe ha almeno un costruttore
- [ ] Ho usato overload dei costruttori dove opportuno

Collezioni

- [ ] Ho usato `List<T>` per gestire collezioni di oggetti
- [ ] Ho usato `Dictionary<K,V>` per ricerche veloci
- [ ] Ho implementato metodi di ricerca/filtraggio (Find, FindAll o foreach)
- [ ] Ho implementato metodi di analisi/statistiche

Interfaccia Grafica

- [ ] Il Form compila e si apre senza errori
- [ ] Posso inserire nuovi elementi
- [ ] Posso visualizzare tutti gli elementi
- [ ] Posso cercare/filtrare elementi
- [ ] Posso gestire l'interfaccia (prenotazioni)
- [ ] Posso vedere statistiche

CRITERI DI AUTOVALUTAZIONE

Dopo aver completato l'esercitazione, valuta te stesso onestamente:

Eccellente (90-100): Ho completato tutto, il codice funziona perfettamente, ho applicato tutti i concetti correttamente, l'interfaccia è completa e funzionale.

Buono (75-89): Ho completato la maggior parte, alcuni aspetti possono essere migliorati, qualche funzionalità manca ma i concetti fondamentali ci sono.

Sufficiente (60-74): Ho applicato i concetti base (classi, ereditarietà, collezioni) ma mancano parti importanti (interfacce, polimorfismo, alcune funzionalità).

Insufficiente (<60): Ho difficoltà con i concetti fondamentali, il programma non compila o mancano troppi elementi.

Se sei sotto "Buono", ripeti l'esercitazione!

DOMANDE DI RIFLESSIONE

Dopo aver completato l'esercitazione, rifletti su queste domande:

1. **Perché ho usato l'ereditarietà?** Quale codice ho evitato di duplicare?
2. **Perché ho usato un'interfaccia?** Quali classi hanno un comportamento comune?
3. **Come ho sfruttato il polimorfismo?** Dove ho gestito oggetti diversi in modo uniforme?
4. **Perché ho usato List e Dictionary?** Quali vantaggi mi hanno dato?
5. **Cosa cambierei della mia progettazione?** Dove posso migliorare?

Rispondere a queste domande ti aiuterà a capire i concetti, non solo ad applicarli meccanicamente.

ESTENSIONI FACOLTATIVE

Se hai completato tutto e vuoi sfidare te stesso:

1. **Aggiungi un sistema di abbonamenti:** interfaccia `IAbbonabile` con metodi per gestire abbonamenti mensili/annuali
2. **Aggiungi ricerche complesse:** corsi in una fascia oraria, servizi con posti quasi esauriti
3. **Aggiungi persistenza dei dati:** salva/carica i servizi e prenotazioni da file di testo
4. **Migliora la GUI:** aggiungi icone, colori, validazione degli input, calendario per orari

---

RISORSE UTILI

- **Dispense:** OOP1_Basi, OOP2_Ereditarietà, OOP3_Polimorfismo
- **Esercizi visti:** Esercizio Sistema Pagamenti, Forme Geometriche, Plugin System
- **Progetto MockTest:** esempio completo di interfacce e collezioni

---

Buon lavoro e buona preparazione!

Nota: Se durante l'esercitazione incontri difficoltà, **è normale!** Annota i dubbi e chiedi chiarimenti. Meglio sbagliare ora che durante la verifica.