Esercizio
Si progetti un'applicazione per gestire le attività di una tipografia che si occupa di stampare sia libri che quotidiani. Data la classe astratta così definita:

```
public abstract class Stampabile {
        String getTitolo();
        int getPagine();
}
```

si progettino le classi concrete Libro e Quotidiano. Il libro ha un titolo, un autore ed un numero di pagine, un quotidiano ha un titolo, una data ed un numero di pagine. Entrambe le classi sono sottoclassi di Stampabile.
Si progetti la classe Tipografia, con una struttura dati daStampare che possa contenere sia libri che quotidiani. La struttutra dati daStampare è inizialmente vuota. Si progettino un costruttore ed i seguenti metodi:

- void inserisci(Stampabile s): inserisce un libro o un quotidiano nella struttura dati;
- boolean rimuovi(): rimuove dalla struttura dati il primo libro o quotidiano inserito; restituisce true se è stato rimosso, false altrimenti;
- boolean rimuovi(String s): rimuove dalla struttura dati il libro o il quotidiano il cui titolo sia "s"; restituisce true se è stato rimosso, false altrimenti;
- int conta(): restuisce il numero totale di pagine da stampare;
- boolean vuota() : restuisce true se la struttura dati è vuota, false altrimenti:
- void rimuoviTutto(): rimuove tutti i libri ed i quotidiani dalla struttura dati.

Si progetti infine un metodo toString() della classe Tipografia. Se la struttura dati è vuota, il metodo toString restituisce la stringa "Non ci sono testi da stampare", altrimenti restituisce una stringa contenente la concatenazione di tutti i titoli presenti nella struttura dati.

Esercizio
Si considerino la classe astratta Archiviabile e la classe Biblioteca:

```
public abstract class Archiviabile {
   int numeroOggettiInArchivio();
   int numeroPosizioniLibere();
}

public class Biblioteca {
        private String[] descrizioneOggetto;
}
```

dove l'array descrizioneOggetto contiene le descrizione dei libri oppure null se la posizione è libera.
Si modifichi la classe Biblioteca affinché estenda la classe astratta Archiviabile, dove il primo metodo calcola il numero totale di libri presenti in biblioteca, il secondo metodo le posizioni libere.

Esercizio
Definire un insieme di classi pubbliche per gestire un archivio delle strade della provincia di Roma. Tutte le classi devono essere sottoclassi della classe astratta `Strada` che contiene
i metodi di accesso

- `String getDenominazione( )`: restituisce il nome della strada;

- `double getLunghezzaKM( )`: restituisce la lunghezza della strada;
- `int getNumeroCorsie( )`: restituisce il numero di corsi della strada;

ed il metodo

- `double VelocitaMaxKmH()`: restituisce la velocità massima in chilometri della specifica strada;

Una classe St`radaUrbana` che rappresenta una strada urbana, di cui interessa sapere anche la città nella quale è presente, sottoclasse concreta di `Strada.` La classe deve fornire i seguenti metodi pubblici aggiuntivi:

- il costruttore St`radaUrbana(String denominazione, String citta, double lunghezzaKM)`: costruttore con tre argomenti: nome, città in cui si trova la strada e lunghezza; il costruttore inizializza il numero delle corsie a 2;
- il metodo `public String citta()` che restituisce il nome della città in cui si trova la strada
- il metodo `public String Informazioni()` che restituisce i dati della strada nel seguente formato: "URBANA <denominazione>(<citta>) - <numero corsie> - <lunghezza> Km"

Progettare opportunamente la classe in modo che per tutte le strade urbane il limite di velocità sia fissato a 50 Km/h.

Una classe St`radaExtraUrbana` che rappresenta una strada extraurbana, sottoclasse concreta di `Strada.` La classe deve anche memorizzare la lista di città attraversate dalla strada.
La classe deve fornire i seguenti metodi pubblici aggiuntivi:

- il costruttore St`radaExtraUrbana(String denominazione, double lunghezzaKM, int NumeroCorsie)`: costruttore con tre argomenti: nome, lunghezza e numero di corsie; il costruttore inizializza la lista delle città attraversate alla lista vuota;
- il metodo `public List<String> cittaAttraversate()` che restituisce la lista delle città attraversate
- il metodo `public void aggiungiCitta(String citta)` che aggiunge in coda una nuova città all'elenco delle città attraversate.
- il metodo `public String Informazioni()` che restituisce i dati della strada nel seguente formato: "EXTRAURBANA <denominazione> - <numero corsie> - <lunghezza> Km - <velocità> Km/h"

Progettare opportunamente la classe in modo che per tutte le strade extraurbane il limite di velocità sia fissato a 90 Km/h. •

Una classe `UtilitaStrade`: classe contenente:

- un metodo statico pubblico `double lunghezzaTotale( List<Strada> l)` che, data una lista l di strade, restituisce la lunghezza totale (in KM) di tutte le strade che compaiono in l.
- un metodo statico pubblico `List<String> elencoCitta( List<Strada> l)` che, data una lista l di strade, restituisce una lista contenente tutte le città attraversate dalle varie strade, nello stesso ordine in cui appaiono in l.

Esercizio
Definire un insieme di classi pubbliche per gestire una pizzeria.
La classe Ingrediente, che deve fornire i seguenti metodi pubblici

- Il costruttore `Ingrediente (String nome, double prezzo)`
- `String getNome( )`: restituisce il nome dell'ingrediente;
- `double getPrezzo( )`: restituisce il prezzo dell'ingrediente;

La classe Pizza, che deve fornire i seguenti metodi pubblici

- Il costruttore `Pizza (String nome, ArrayList <Ingrediente> ingredienti)`
- `String getNome( )`: restituisce il nome della pizza;
- `ArrayList <Ingrediente> getIngredienti( )`: restituisce la lista degli ingredienti;
- `double getPrezzo( )`: restituisce il prezzo della pizza come somma del prezzo degli ingredienti;

La classe Pizzeria serve per la gestione e la vendita delle pizze. Per ogni pizzeria ci interessa sapere il nome, gli ingredienti e le pizze presenti nella pizzeria. La classe deve fornire i seguenti metodi pubblici

- Il costruttore `Pizzeria (String nomePizz)` che costruisce una pizzeria di nome NomePizz e con lista di ingredienti e pizze inizialmente vuoto;
- `void aggiungiIngrediente (Ingrediente i)` con cui aggiunge un ingrediente a quelli posseduti dalla pizzeria. Se nella pizzeria esiste già un ingrediente con lo stesso nome di quello che si vuole aggiungere il metodo non fa niente.
- `Ingrediente getIngrediente(String nome)` che permette di recuperare un determinato ingrediente attraverso il nome, se un ingrediente con il nome specificato è presente in pizzeria
- `ArrayList <Ingrediente> elencoIngredienti ()` per visualizzare l'elenco degli ingredienti a disposizione nella pizzeria
- `Pizza getPizza` che permette di recuperare una determinata pizza attraverso il nome, se una pizza con il nome specificato è presente in pizzeria
- `void aggiungiPizza (Pizza p)`che permette di aggiungere una pizza a quelle presenti nella pizzeria a patto che nella pizzeria non sia già presente una pizza con lo stesso nome e che la pizzeria abbia a disposizione tutti gli ingredienti previsti dalla pizza che si vuole aggiungere, altrimenti non fa niente.
- `double ordine (String[] pizze)` che riceve i nomi delle pizze desiderate e calcola il totale da pagare per l'ordinazione richiesta, a patto che nella pizzeria siano presenti tutte le pizze i cui nomi compaiono nell'ordinazione (altrimenti restituisce 0).

Esercizio
Un'agenzia di viaggio desidera gestire un parco mezzi a noleggio. Per ciascun mezzo è di interesse la targa (di tipo String) e se al momento è noleggiato. Se il mezzo è un pullman, è di interesse anche il numero di posti (int). Se il mezzo invece è un'auto, la sua cilindrata (int). Scrivere una gerarchia di classi pubbliche con tutte le var private che permettano di modellare le informazioni descritte.
Si progettino inoltre i seguenti metodi per supportare le seguenti funzionalità relative ai mezzi:

- costruttore;
- metodi `getTarga()`, `getCilindrata()` (per le sole auto) e `getPosti()` (per i soli pullman) che permettono di leggere i campi corripondenti
- `equals()` uguaglianza di due mezzi;
- `costoNoleggio()` restituisce il costo giornaliero di noleggio del mezzo (come double), pari a cilindrata/20 per le auto, e 5*numero posti per i pullman;

- `noleggiato()` restituisce un valore booleano che indica se il mezzo è noleggiato oppure no

L'agenzia di viaggi vuole poter gestire il suo parco mezzi mediante una classe pubblica ParcoMezzi con metodi che implementiono le seguenti funzionalità:

- `aggiungiMezzo()`: che, dato un mezzo, lo aggiunge al parco dell'agenzia; se la targa del mezzo è già utilizzata stampa il messaggio "targa esistente";
- `eliminaMezzo()`: che, data la targa b di un mezzo, lo elimina dal parco dell'agenzia; se la targa b non è utilizzata stampa il messaggio "targa inesistente";
- `listaAuto()`: che restituisce una lista con tutte e sole le auto del parco mezzi;
- `noleggiaMezzo()`: che restituisce un mezzo da noleggiare, modificando lo stato di tale mezzo per indicare che è stato noleggiato; se non esiste un mezzo libero stampa il messaggio "mezzi esauriti";
- `restituitoMezzo()`:che, data la targa b di un mezzo, lo rende di nuovo disponibile.

Esercizio
Si progetti un'applicazione per gestire appelli di esame. Si progetti una classe Iscritto con tre variabili di istanza: nome, matricola e voto.
Data la classe astratta:

```
public abstract class Appello {
     void iscrivi(Iscritto studente);
     \\ che iscrive uno studente all'appello. All'atto dell'iscrizione il voto dello studente è -1;
     void registraVoto(Iscritto studente, int voto);
     void printIscritti();
     }
```

si progettino le seguenti classi:

- la classe concreta AppelloScritto che estende Appello. Un appello scritto deve contenere una quantità variabile di iscritti, non nota a priori, che può crescere arbitrariamente. In particolare avrà quattro variabili di istanza:
  - nome del corso;
  - CFU: numero dei crediti associati all'esame;
  - ore: numero di ore a disposizione per l'esame;
  - iscritti: la lista degli studenti iscritti,

  ed i seguenti metodi:
  - i metodi get corrispondenti a tutte le variabili d'istanza;
  - `getPromossi()` restituisce la lista degli iscritti che hanno superato l'esame;
  - `totPromossi()` restituisce il numero degli iscritti che hanno superato l'esame;
  - `getMedia()` restituisce il voto medio dell'appello, considerando tutti e soli gli studenti che hanno superato l'esame;
- la classe Idoneità: un'idoneità è un particolare tipo AppelloScritto in cui il voto registrato può essere solo 0 (bocciato) o 1 (promosso).

Si progetti una classe SessioneEsame che possa contenere al massimo sei appelli sia scritti che idoneità. La classe Sessione Esame deve avere i seguenti metodi:

- `totAppello ()`: restituisce il numero totale degli appelli presenti nella sessione;

- `totPromossi()`: restituisce il numero totale degli iscritti che hanno superato un esame di un appello nella sessione (se un iscritto ha superato più esami va contato più volte);
- `add(Appello a)`: inserisce un nuovo appello nella sessione;
- `getFasciaVoti(double min, double max)`: restituisce una ArrayList contenente tutti gli iscritti che hanno superato un numero di esami compreso tra min e max.

Esercizio
Si progetti un'applicazione per la gestione di dati riguardanti atleti.
Data la classe astratta:

```
public abstract class Atleta
     {
     String getNome();
     String getCognome();
     double rendimento();
     }
```

si progettino le seguenti classi:

- la classe concreta Calciatore che estende Atleta con i seguenti metodi
  Un calciatore ha quattro variabili d'istanza: nome, cognome, numero partite, goal segnati;

  ed i seguenti metodi:
  - i metodi get corrispondenti a tutte le variabili d'istanza;
  - `Calciatore(String nome, String cognome)`: costruttore avente come argomento il nome e il cognome del calciatore. Il costruttore inizializza numero di partite e goal segnati a zero.
  - `Calciatore(String nome, String cognome, int presenze, int goal)`: costruttore con quattro argomenti: nome e cognome del calciatore, numero di partite giocate (presenze) e numero di goal segnati (goal);
  - `void aggiungiPresenza()` : incrementa di 1 il numero di presenze;
  - `void aggiungigoal(int g)`: incrementa di g il numero di goal;
  - `double rendimento()`: restituisce il valore del rapporto goal/presenze se il numero di presenze è maggiore di 0; viceversa restituisce -1 se il giocatore non ha mai giocato;
- la classe concreta Nuotatore che estende Atleta con i seguenti metodi
  Un nuotatore ha quattro variabili d'istanza: nome, cognome, numero medaglie oro, numero medaglie argento;

ed i seguenti metodi:

- i metodi get corrispondenti a tutte le variabili d'istanza;
- `Nuotatore(String nome, String cognome, int oro, int argento)`: costruttore con quattro argomenti: nome e cognome del calciatore, numero di medaglie d'oro e numero di medaglie d'argento;
- `void aggiungiMedaglie(int o, int a)`: incrementa di o il numero di medaglie d'oro e di a il numero di medaglie d'argento;
- `double rendimento()`: restituisce la somma del numero di medaglie d'argento diviso 10 con il prodotto del numero delle medaglie d'oro diviso due;

Progettare la classe Squadra, per gestire le informazioni relative ad una collezione di atleti rappresentata tramite array, che implementa il seguente metodo:

- `double rendimentoMedio()`: restituisce il rendimento medio di tutta la squadra (considerando solo gli atleti che hanno un rendimento diverso da zero);
- `int contaSopraMedia( )`: restituisce il numero degli atleti che hanno un rendimento superiore alla media (considerando solo gli atleti che hanno un rendimento diverso da zero);