Parte 1: Array e Funzioni

Esercizio 1: Rotazione Circolare

Implementa una funzione che ruota gli elementi di un array di n posizioni verso destra.

Specifiche:

- Nome funzione: `rotateArray`
- Parametri:
  - `int[] array`: l'array da ruotare
  - `int n`: numero di posizioni di rotazione
- Return: `int[]`: nuovo array ruotato
- Esempio: `[1,2,3,4,5]` con n=2 diventa `[4,5,1,2,3]`

Esercizio 2: Trova Sequenza

Implementa una funzione che trova la sottosequenza crescente più lunga in un array.

Specifiche:

- Nome funzione: `longestIncreasingSequence`
- Parametri: `int[] array`
- Return: `int[]`: array contenente la sottosequenza trovata
- Esempio: in `[1,2,3,1,5,6,7,2]` trova `[1,5,6,7]`

Esercizio 3: Fusione Ordinata

Implementa una funzione che fonde due array ordinati mantenendo l'ordinamento.

Specifiche:

- Nome funzione: `mergeSorted`
- Parametri:
  - `int[] array1`: primo array ordinato
  - `int[] array2`: secondo array ordinato
- Return: `int[]`: nuovo array ordinato contenente tutti gli elementi
- Esempio: `[1,3,5]` e `[2,4,6]` diventano `[1,2,3,4,5,6]`

Esercizio 4: Array Pari e Dispari

Implementa una funzione che divide un array in due array separati: uno contenente i numeri pari e l'altro i numeri dispari.

Specifiche:

- Nome funzione: `dividiPariDispari`
- Parametri: `int[] array`
- Return: `int[][]`: array di due array (array[0] contiene i pari, array[1] contiene i dispari)
- Esempio: `[1,2,3,4,5,6]` diventa `[[2,4,6],[1,3,5]]`

Esercizio 5: Compressione Array

Implementa una funzione che comprime un array contando le occorrenze consecutive di ogni numero.

Specifiche:

- Nome funzione: `comprimi`
- Parametri: `int[] array`
- Return: `int[][]`: array di coppie [numero, conteggio]
- Esempio: `[1,1,1,2,2,3,3,3,3]` diventa `[[1,3],[2,2],[3,4]]`

Esercizio 6: Filtraggio Array

Implementa una funzione che filtra un array secondo un criterio dato (maggiore di un valore).

Specifiche:

- Nome funzione: `filtra`
- Parametri:
  - `int[] array`
  - `int valore`
- Return: `int[]`: nuovo array contenente solo i numeri maggiori del valore dato
- Esempio: `[1,4,2,7,3,9]` con valore=3 diventa `[4,7,9]`

Parte 2: Classi

Per ogni classe, creare una classe Tester che verifica tutti i metodi implementati.

Esercizio 7: Sistema Biblioteca

Crea un sistema per gestire una biblioteca.

Specifiche:

1. Crea una classe `Libro` con:
   - Attributi privati:
     - `titolo` (String)

        - `autore` (String)
        - `isbn` (String)
        - `disponibile` (boolean)
    - Costruttore che inizializza tutti gli attributi
    - Metodi get/set appropriati
    - Metodo `toString()` che restituisce una stringa formattata con le info del libro
2. Crea una classe `Biblioteca` con:
    - Attributo privato:
        - `libri` (array di `Libro`)
    - Metodi:
        - `aggiungiLibro(Libro libro)`: aggiunge un libro alla biblioteca
        - `rimuoviLibro(String isbn)`: rimuove un libro dato l'ISBN
        - `prestaLibro(String isbn)`: segna un libro come non disponibile
        - `restituisciLibro(String isbn)`: segna un libro come disponibile
        - `cercaLibro(String titolo)`: cerca e restituisce libri con quel titolo
        - `libriDisponibili()`: restituisce un array con tutti i libri disponibili

Esercizio 8: Gestore Studenti

Crea un sistema per gestire gli studenti di una classe.

Specifiche:

1. Crea una classe `Voto` con:
    - Attributi privati:
        - `materia` (String)
        - `valore` (double)
        - `data` (String)
    - Costruttore e metodi get/set appropriati
2. Crea una classe `Studente` con:
    - Attributi privati:
        - `nome` (String)
        - `cognome` (String)
        - `matricola` (String)
        - `voti` (array di `Voto`)
    - Metodi:
        - `aggiungiVoto(Voto voto)`
        - `mediaVoti()`: calcola la media di tutti i voti
        - `mediaMateria(String materia)`: calcola la media dei voti di una materia
        - `votiInsufficienti()`: restituisce un array di voti insufficienti (<6)

Esercizio 9: Registro Spese

Crea un sistema per gestire le spese personali.

Specifiche:

1. Crea una classe `Spesa` con:
   - Attributi privati:
     - `descrizione` (String)
     - `importo` (double)
     - `categoria` (String)
     - `data` (String)
   - Costruttore e metodi get/set appropriati
   - Metodo `toString()` per visualizzare la spesa
2. Crea una classe `RegistroSpese` con:
   - Attributo privato:
     - `spese` (array di `Spesa`)
   - Metodi:
     - `aggiungiSpesa(Spesa spesa)`
     - `rimuoviSpesa(int indice)`
     - `totaleSpese()`: calcola il totale di tutte le spese
     - `totalePerCategoria(String categoria)`
     - `speseDelGiorno(String data)`
     - `categoriaConSpesaMaggiore()`: trova la categoria dove si è speso di più