Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea in Informatica

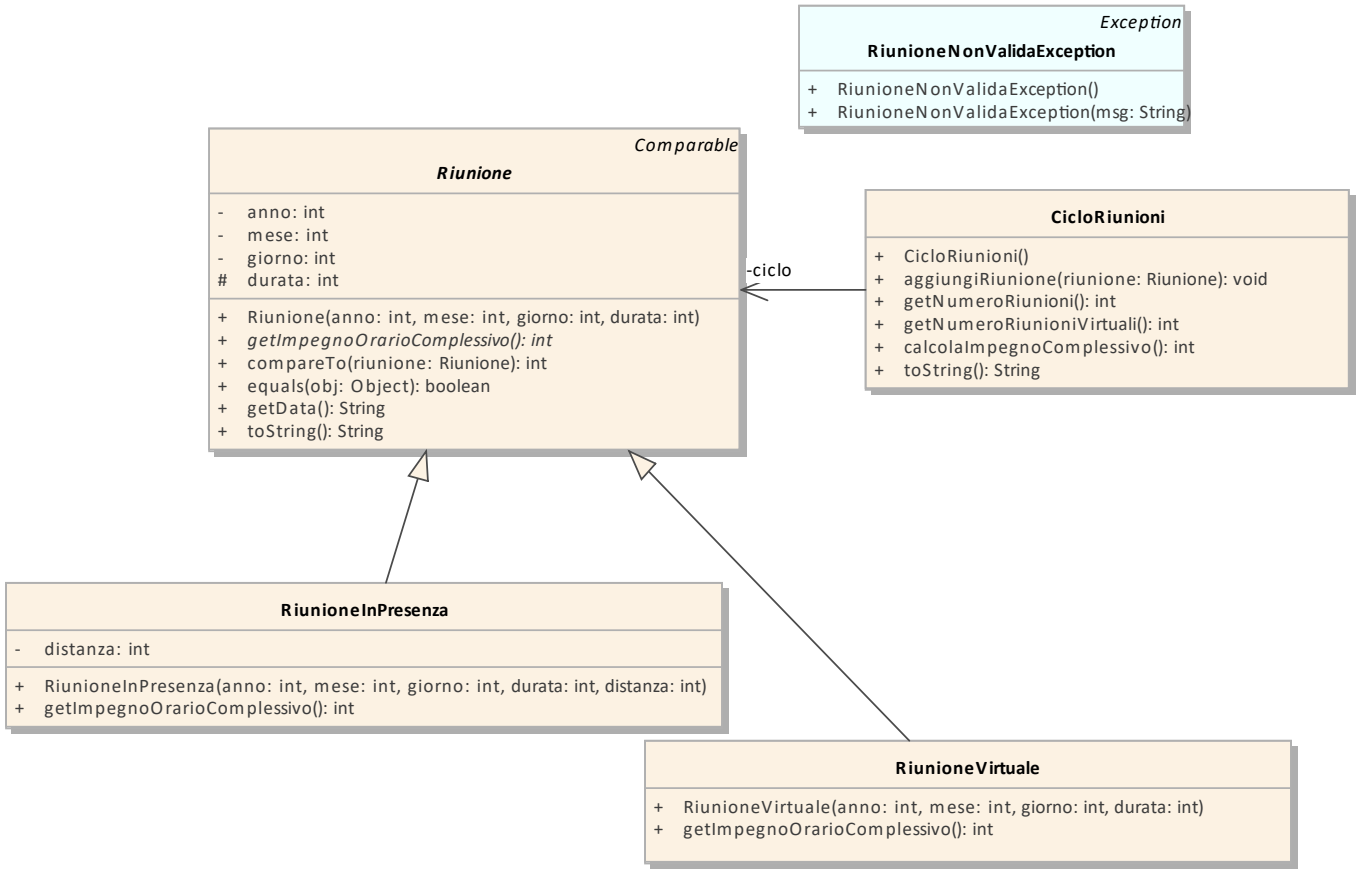
Programmazione 2

18 Giugno 2021 – Secondo Compitino

Testo parte di pratica

Si consideri un programma che aiuta un partecipante a gestire un `CicloRiunioni` che possono essere `Virtuali` oppure `InPresenza`. Le riunioni sono pianificate in una certa data e hanno una durata predeterminata. Inoltre, le riunioni in presenza si svolgono ad una distanza nota dalla dimora del partecipante. Il ciclo di riunioni viene mantenuto ordinato per data.

Implementare le classi esattamente come rappresentate dal seguente diagramma UML. **Il diagramma include tutti e i soli metodi richiesti, compresi quelli di incapsulamento.**



Viene fornita la classe **`TestEsame`** che contiene un insieme di casi di test che devono essere fatti girare di volta in volta in modo da verificare la corretta realizzazione del programma.

Classe `RiunioneNonValidaException`:

Rappresenta una eccezione che può essere sollevata dal programma in determinate occasioni.

Classi `Riunione, RiunioneVirtuale e RiunioneInPresenza`:

✓ Rappresentano una gerarchia di 3 classi per la rappresentazione delle riunioni

Classe Riunione

- ✓ `Riunione` è una classe *astratta* con 4 attributi: `anno`, `mese`, `giorno` e `durata`. I primi tre attributi rappresentano la data di svolgimento della riunione, mentre il quarto rappresenta la durata prevista per la riunione in ore.
- ✓ Il costruttore `Riunione(int anno, int mese, int giorno, int durata)` inizializza i 4 attributi sollevando una eccezione di tipo `RiunioneNonValidaException` qualora `anno` assuma un valore minore di 2021, `mese` sia minore di 1 oppure maggiore di 12, `giorno` sia minore di 1 oppure maggiore di 31, oppure `durata` sia minore di 1.
- ✓ Il metodo `getImpegnoOrarioComplessivo()` è un metodo astratto che calcola l'impegno orario complessivo del partecipante alla riunione tenendo conti degli eventuali tempi di spostamento.
- ✓ Il metodo `getData()` ritorna una stringa con la data nel formato `giorno/mese/anno`. Ad esempio, se `giorno=10`, `mese=4`, e `anno=2021`, la stringa ritornata deve essere "`10/4/2021`".
- ✓ Il metodo `toString()` ritorna una stringa contenente la data nel formato prodotto dal metodo `getData()` e l'impegno orario complessivo.
- ✓ Il metodo `equals()` confronta due riunioni e ritorna `true` se hanno la stessa data. Due riunioni con la stessa data sono uguali anche se di tipo differente. Quindi una `RiunioneVirtuale` è uguale ad una `RiunioneInPresenza` se le due riunioni hanno la stessa data.
- ✓ Il metodo `int compareTo(Riunione riunione)` ritorna un valore che rappresenta l'ordinamento per data delle riunioni. Si ricorda che nel caso `this` abbia una data minore della riunione `riunione`, il valore di ritorno deve essere negativo; nel caso le due date siano coincidenti il valore di ritorno deve essere 0; altrimenti un numero positivo.

Classe `RiunioneVirtuale`

- ✓ Estende la classe `Riunione`
- ✓ Il metodo `getImpegnoOrarioComplessivo()` ritorna il valore della `durata` della riunione.

Classe `RiunioneInPresenza`

- ✓ Estende la classe `Riunione` aggiungendo un attributo `distanza` che rappresenta il numero di ore necessarie a raggiungere il luogo della riunione.
- ✓ Il costruttore permette di inizializzare gli attributi di `Riunione` e l'attributo `distanza`. Se l'attributo `distanza` è inizializzato con un valore <=0, il costruttore solleva una eccezione del tipo `RiunioneNonValidaException`.
- ✓ Il `metodo getImpegnoOrarioComplessivo()` ritorna il valore della durata della riunione aumentato di 2 volte il valore di `distanza` (2 volte perché si calcola il tempo sia per andare alla riunione, sia per tornare dopo averla fatta).
- ✓ **Classe `CicloRiunioni`:**
- ✓ Ha un attributo che memorizza un insieme di riunioni. Un ciclo di riunioni non ammette due riunioni uguali (cioè due riunioni pianificate in una stessa data) e mantiene le riunioni ordinate per data. La collezione utilizzata deve quindi essere un TreeSet.
- ✓ Il costruttore di default inizializza opportunamente la collezione.
- ✓ Il `metodo aggiungiRiunione(Riunione riunione)` aggiunge la riunione `riunione` alla collezione se diversa da `null`, altrimenti solleva una eccezione di tipo `RiunioneNonValidaException`.
- ✓ Il metodo `getNumeroRiunioni()` ritorna il numero di riunioni complessivo presenti nella collezione
- ✓ Il metodo `getNumeroRiunioniVirtuali()` ritorna il numero di riunioni virtuali presenti nella collezione
- ✓ Il metodo `calcolaImpegnoComplessivo()` ritorna il numero totale di ore di impegno che il ciclo di riunioni prevede. L'impegno richiesto da ciascuna riunione deve essere calcolato utilizzando il metodo `getImpegnoOrarioComplessivo()`.
- ✓ Il metodo `toString()` ritorna una stringa con il contenuto della collezione. Notare che il contenuto deve essere stampato ordinato per data.