Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea in Informatica

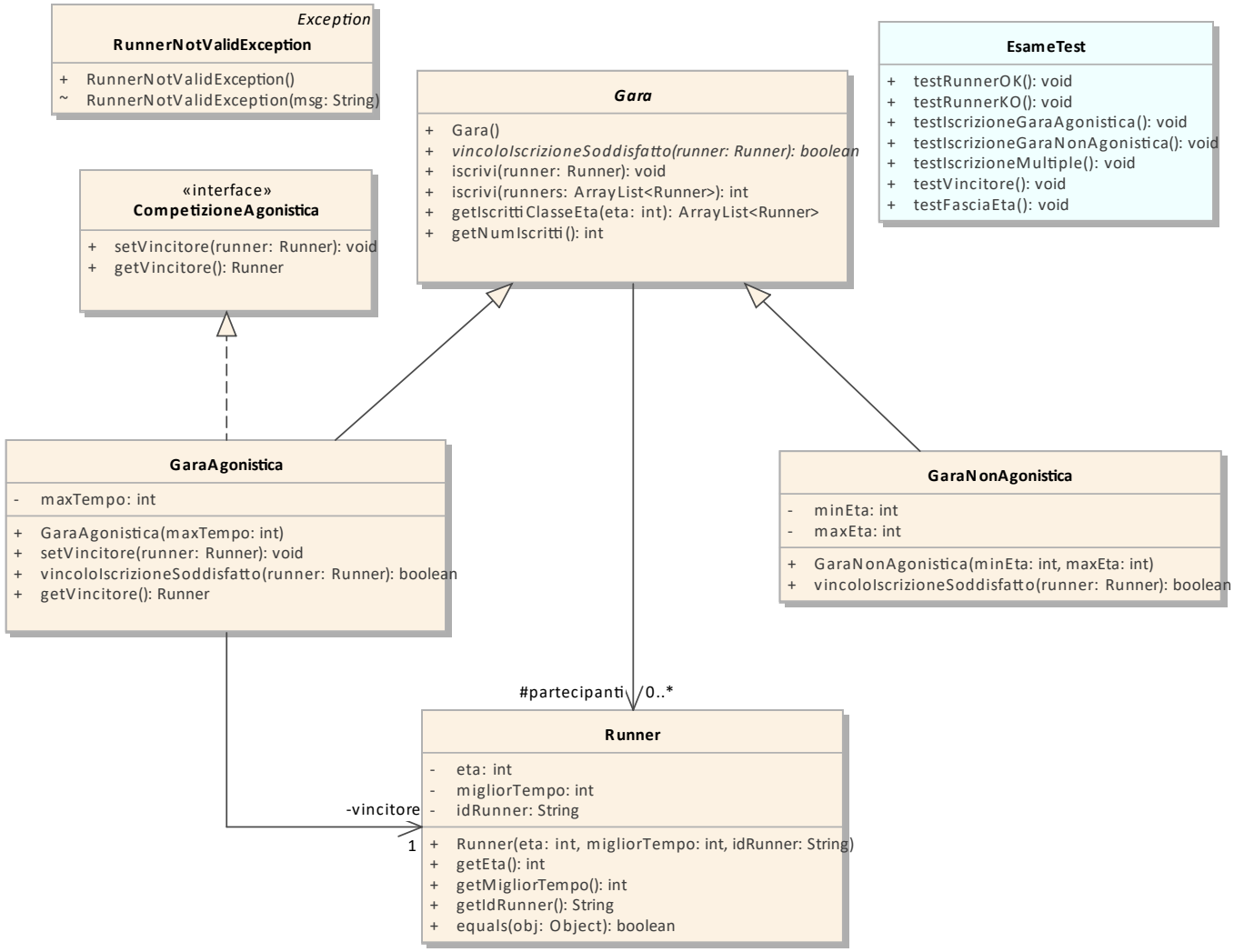
Programmazione 2

19 Gennaio 2022 – Appello

Testo parte di pratica

Si consideri un programma per la gestione delle iscrizioni alle gare di corsa (come ad esempio le maratone). Le gare possono essere di tipo non agonistico (non prevedono un vincitore e possono iscriversi solamente Runner appartenenti ad una certa fascia di età) oppure agonistico (prevedono un vincitore e le iscrizioni sono possibili solamente se il miglior tempo fatto dall'iscritto in gare precedenti è inferiore ad un tempo massimo ammissibile associato alla gara).

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 **`EsameTest`** 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 `RunnerNotValidException`:

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

Classe `Runner`:

- ✓ Rappresenta un generico partecipante ad una gara di corsa ed è caratterizzato da tre attributi: `eta` (indica l'età del partecipante), `migliorTempo` (indica il miglior tempo fatto in gare precedenti dal partecipante),

`idRunner` (è un identificatore univoco del partecipante). La classe implementa 3 metodi `getter` che ritornato i valori dei 3 attributi.

- ✓ Il costruttore `Runner(int eta, int migliorTempo, String idRunner)` inizializza i 3 attributi e solleva una eccezione di tipo `RunnerNotValidException` qualora `eta` assuma valori inferiori a 10 oppure superiori a 100, `migliorTempo` sia inferiore o uguale a 0, `idRunner` sia `null` oppure la stringa vuota.
- ✓ Il metodo `equals(Object obj)` confronta due `Runner` ritornando `true` nel caso i due runner abbiamo lo stesso `idRunner`.

Interface CompetizioneAgonistica:

- ✓ E' una interfaccia che rappresenta una generica competizione agonistica. Essa prescrive l'implementazione di un metodo per definire il vincitore della competizione, metodo `void setVincitore(Runnner runner) throws RunnerNotValidException`, e un metodo per ritornare il vincitore della competizione `Runner getVincitore()`.

Classi `Gara, GaraNonAgonistica e GaraAgonistica`:

- ✓ Rappresentano una gerarchia di 3 classi per la gestione delle gare.

Classe `Gara`

- ✓ E' una classe astratta
- ✓ Implementa l'attributo `partecipanti` che è un `ArrayList` contenente tutti i partecipanti iscritti alla gara. Il `costruttore` inizializza l'attributo con un `ArrayList` vuoto.
- ✓ Dichiara il metodo astratto `boolean vincoloIscrizioneSoddisfatto(Runner runner)` che controlla se `runner` soddisfa i vincoli per l'iscrizione imposti dalla gara.
- ✓ Implementa il metodo `iscrivi(Runner)` che aggiunge il `Runner` alla lista dei partecipanti. Il `Runner` non viene aggiunto e il metodo solleva una eccezione nel caso in cui il `Runner` sia già iscritto alla gara oppure nel caso il `Runner` non soddisfi i vincoli controllati dal metodo `vincoloIscrizioneSoddisfatto`.
- ✓ Implementa il metodo `int iscrivi(ArrayList<Runner> runners)` che aggiunge alla lista dei partecipanti i runner che non sono già iscritti alla gara e che superano i vincoli controllati dal metodo `vincoloIscrizioneSoddisfatto`. I runner già iscritti o che non superano i vincoli vengono ignorati, e il metodo continua a valutare gli altri. Alla fine, il metodo ritorna il numero di runner che non sono stati iscritti.
- ✓ Implementa il metodo `int getNumIscritti()` che ritorna il numero di iscritti alla gara.
- ✓ Implementa il metodo `ArrayList<Runner> getIscrittiClasseEta(int eta)` che ritorna un `ArrayList` con tutti i partecipanti che hanno una età strettamente maggiore dell'età indicata dal parametro del metodo.

Classe `GaraNonAgonistica`

- ✓ Estende la classe `Gara` ed è caratterizzata dagli attributi `minEta` e `maxEta` che rappresentano l'età minima e massima che un `Runner` può avere per iscriversi alla gara.
- ✓ Il costruttore `GaraNonAgonistica (int minEta, in maxEta)` inizializza i due attributi. Si assume che i valori passati al costruttore siano sempre corretti, non deve quindi essere controllata la loro validità.
- ✓ Il metodo `vincoloIscrizioneSoddisfatto(Runner runner)` ritorna `true` nel caso l'età del runner sia maggiore o uguale a `minEta` e minore o uguale a `maxEta`, altrimenti ritorna `false`.

Classe `GaraAgonistica`

- ✓ Estende la classe `Gara` e implementa l'interfaccia `CompetizioneAgonistica` ed è caratterizzata dagli attributi `maxTempo` e `vincitore`.
- ✓ Il costruttore `GaraAgonistica (int maxTempo)` inizializza l'attributo `vincitore` a `null` e l'attributo `maxTempo` utilizzando il valore del parametro. Si assume che i valori passati al costruttore siano sempre corretti, non deve quindi essere controllata la loro validità.
- ✓ Il metodo `vincoloIscrizioneSoddisfatto(Runner runner)` ritorna `true` se il miglior tempo del `runner` è strettamente inferiore a `maxTempo`, altrimenti ritorna `false`.
- ✓ Il metodo `void setVincitore(Runner runner)` utilizza il parametro per inizializzare l'attributo `vincitore`. Il metodo ritorna l'eccezione `RunnerNotValidException` se il `runner` passato come parametro non compare nella lista dei partecipanti.
- ✓ Il metodo `Runner getVincitore()` ritorna il valore dell'attributo `vincitore`.