Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea in Informatica

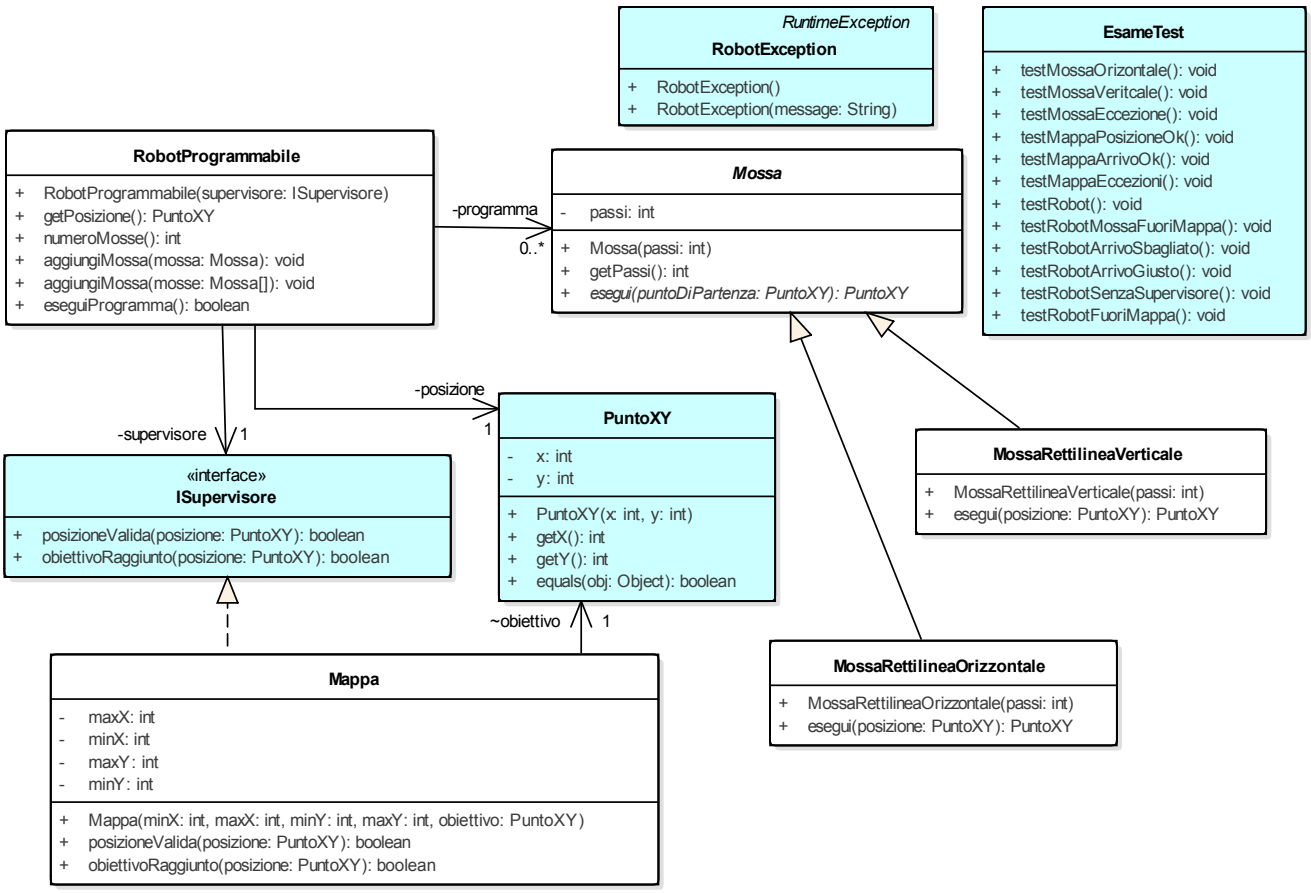
Programmazione 2

Appello Completo

Testo parte di pratica

Un'azienda vende robot giocattolo che possono essere programmati per eseguire una sequenza di spostamenti (mosse) orizzontali e verticali. Ad ogni mossa, il robot comunica con un supervisore che decide se la posizione raggiunta è valida o meno, e stabilisce se il percorso effettuato raggiunge un obiettivo previsto. Un particolare tipo di supervisore valuta gli spostamenti del robot rispetto a una mappa a schema rettangolare, accettando come valide le mosse che mantengono il robot all'interno dello schema previsto, e fissa un punto dello schema come obiettivo da raggiungere.

Il sistema da implementare è rappresentato in maggior dettaglio dal diagramma UML e dalle specifiche di seguito riportate.



La classe `EsameTest` (già fornita) contiene un insieme di test che devono essere fatti girare di volta in volta in modo da verificare la corretta realizzazione del software. **Come requisito minimo per ottenere una valutazione positiva, lo studente deve garantire che la sua implementazione non presenti errori di compilazione e superi almeno 3 casi di test fra quelli dati.**

Le classi in azzurro sono già implementate, le rimanenti sono da implementare. Il diagramma UML è completo: non occorrono altri metodi.

Classe `RobotException` (classe già data): rappresenta un'eccezione lanciata in diversi punti del codice.

Classe `PuntoXY` (classe già data): rappresenta un punto del piano espresso in coordinate cartesiane `x` e `y`.

Interfaccia `ISupervisore` (già data): rappresenta il comportamento di un supervisore che può decidere se la posizione del robot è valida (metodo `posizioneValida`) e se essa corrisponde ad un obiettivo previsto (metodo `obiettivoRaggiunto`).

Classi `Mossa, MossaRettilineaOrizontale e MossaRettilineaVerticale`:

- ✓ È una gerarchia di classi per rappresentare spostamenti di un numero di `passi` impostabile via parametro.
- ✓ Si implementi la gerarchia di classi, massimizzando il riuso di codice nella classe astratta.
- ✓ Il numero di `passi` si imposta con il costruttore e deve essere un numero positivo o negativo, ma non nullo, altrimenti il costruttore solleva l'eccezione `RobotException`.
- ✓ L'attributo `passi` è immutabile e accessibile in lettura.
- ✓ Il metodo `esegui` prende in input un `PuntoXY` di partenza e restituisce il `PuntoXY` raggiunto dopo uno spostamento del numero di unità corrispondenti al valore (positivo o negativo) dell'attributo `passi`: se il movimento è orizzontale, lo spostamento incide solo sul valore della coordinata `x`, mentre se il movimento è verticale, lo spostamento incide solo sul valore della coordinata `y`. Ad esempio, se il robot si trova nel punto di coordinate x=10 e y=2, uno spostamento verticale di 10 passi farebbe raggiungere al robot la posizione x=10, y=12, mentre uno spostamento orizzontale di -5 passi fa raggiungere al robot la posizione x=5, y=2.

Classe `RobotProgrammabile`:

- ✓ Rappresenta un robot caratterizzato dalla propria `posizione` (oggetto di tipo `PuntoXY`) nel piano cartesiano, un `programma` (ovvero un collezione di mosse) da eseguire, e un `supervisore`.
- ✓ Il costruttore inizializza `posizione` al punto di coordinate `<0,0>`, `programma` come una `ArrayList` inizialmente vuota, e `supervisore` in base al valore del parametro in input. Solleva l'eccezione `RobotException` nel caso `supervisore` sia `null`, o se la posizione iniziale non è valida secondo il `supervisore`.
- ✓ Il metodo `getPosizione` restituisce il valore dell'attributo `posizione`.
- ✓ Il metodo `aggiungiMossa(Mossa)` permette di aggiungere una mossa al programma del robot. La mossa viene aggiunta solo se il parametro è diverso da `null`, e viene ignorata in caso contrario.
- ✓ Il metodo `aggiungiMossa(Mossa[])` aggiunge tutte le mosse dell'array in input, mantenendo quanto sopra relativamente all'aggiunta di ogni singola mossa.
- ✓ Il metodo `numeroMosse` restituisce il numero di mosse che fanno parte del programma del robot.
- ✓ Il metodo `eseguiProgramma` modifica la `posizione` del robot in base all'esecuzione di tutte le mosse del `programma` in sequenza. Dopo ogni mossa, controlla che la posizione raggiunta sia valida secondo il `supervisore` e solleva l'eccezione `RobotException` in caso di non validità. Il metodo restituisce `true` se la posizione finale del robot corrisponde all'obiettivo atteso dal `supervisore`, o `false` in caso contrario.

Classe `Mappa`:

- ✓ Implementa l'interfaccia `ISupervisore`, definendo un tipo di supervisore che valuta gli spostamenti del robot rispetto a una mappa a schema rettangolare in cui la coordinata `x` può variare fra gli estremi `xMin` e `xMax` inclusi, e la coordinata `y` fra gli estremi `yMin` e `yMax` inclusi. La mappa definisce inoltre un punto del piano come `obiettivo` da raggiungere.
- ✓ Il costruttore inizializza tutti gli attributi in base a parametri passati in input, controllando esplicitamente che `xMin` sia strettamente minore di `xMax`, che `yMin` sia strettamente minore di `yMax`, che `obiettivo` non sia `null` e sia una punto compreso nella mappa, e sollevando l'eccezione `RobotException` nel caso che i parametri non soddisfino questi vincoli.
- ✓ Il metodo `posizioneValida` valuta se un dato `PuntoXY` è compreso nella mappa, restituisce `true` se il punto è compreso nella mappa, o `false` in caso contrario.
- ✓ Il metodo `obiettivoRaggiunto` valuta se un `PuntoXY` raggiunge l'obiettivo definito nella mappa, restituisce `true` se il punto in input è uguale al punto `obiettivo`.