Una software house sta sviluppando un prototipo di videogioco di guerra tra *fazioni* (vedere il codice della classe **`Factory.Fazione`**). Ciascuna include uno stesso numero di carrarmati (vedere il codice della classe **`Tank`**) che lancia proiettili (classe **`Proiettile`**) all'interno di un campo di battaglia (classe **`Campo`**) recintato ove si trovano anche le loro basi (modellate dalla classe **`Fortino`**).
I tank occupano posizioni di un piano cartesiano modellate dalla classe **`Coordinate`** e si spostano incrementalmente lungo 8 possibili direzioni come modellate dalla classe **`Direzione`**.

Il prototipo, durante una simulazione articolata in un numero finito di passi discreti, produce e colleziona statistiche per valutare l'efficacia delle contrapposte strategie adottate dai carrarmati delle diverse fazioni. Tutti i carrarmati della medesima fazione hanno lo stesso comportamento e lasciano delle tracce dei loro cingolati all'interno campo di battaglia (tracce che poi tendono a sparire nel tempo). Tutti i carrarmati, indipendentemente dalla loro fazione, sono soggetti agli stessi vincoli imposti dalla logica presente nel metodo **`war.simulatore.GestoreLogicaTank.simula()`** a cui si rimanda per i dettagli.

Il comportamento si può sintetizzare con la scelta di quando cambiare direzione (vedi metodo **`Tank#decideDiCambiareDirezione()`**); della direzione da seguire (vedi metodo **`Tank#cambioDirezione()`**); di quando sparare (vedi metodo **`Tank#decideDiSparare()`**). Tutti i tank sono già vincolati dal codice della simulazione a tornare al fortino per ricaricare subito dopo lo sparo (vedi metodo **`war.simulatore.Simulatore#simula()`** e **`war.simulatore.GestoreLogicaTank#simula()`**) .

DOMANDA 1 (5%)
Modificare il codice della classe **`war.Coordinate`** e della classe **`war.Direzione`** affinché tutti i test già presenti nelle classi **`CoordinateTest`** e **`DirezioneTest`** comincino ad avere successo.

*(N.B. Per una più agevole comprensione della descrizione che segue, si consiglia di provare ad eseguire il metodo **`main()`** della classe **`war.gui.Main`** ed osservare l'animazione della simulazione già dopo aver risposto a questa prima domanda. A fine esecuzione, la simulazione stampa alcune statistiche di cui nelle successive domande. Il tasto ESCape anticipa la fine della simulazione e la stampa).*

Inizialmente la classe **`Tank`** implementa una semplice strategia *non informata*: consiste nel fare scelte completamente casuali, per ciascuno degli aspetti appena citati.

Si richiede di studiare e confrontare la sua efficacia con quelle di nuove, e più articolate, strategie. Un programmatore esperto fa notare che conviene far diventare *astratta* la classe **`Tank`** ed introdurre una sua classe estesa concreta **`Explorer`** che realizzi la strategia non informata. In questo modo, le future strategie si possono realizzare con nuove classi concrete che estandano **`Tank`** ed opportuni refactoring del codice di **`Factory`** e **`Factory.Fazione`**, oltre che dei loro utilizzatori. Implementare le modifiche suggerite dal programmatore esperto, quindi verificare il funzionamento dell'applicazione anche dopo le modifiche.

DOMANDA 2 (50%)
Pertanto il programmatore esperto suggerisce di ristrutturare l'applicazione come segue:

a) **(20%)** Rendere astratta la classe **`war.Tank`** per generalizzare tutte le strategie presenti e future. Introdurre la classe **`Explorer`** (che rimane di colore bianco) per estrapolare e sintetizzare la strategia di battaglia non informata che era insita nella classe concreta **`Tank`** già fornita prima di questo refactoring. Nelle classi che facevano già uso di **`Tank`**, rivedere il codice decidendo se continuare ad usare il tipo, ora astratto, **`Tank`**, oppure sostituire i nuovi tipi concreti tra cui lo stesso **`Explorer`**.
Si segnalano utilizzi *almeno* nei metodi **`war.simulatore.Simulatore#creaTank()`** e nel costruttore di **`war.tank.Factory`**.

b) **(20%)** Creare una nuova tipologia di carrarmato aggiungendo la classe **`Shooter`** (in *verde*): questa fazione di carrarmati cerca e si dirige verso celle nelle immediate adiacenze in cui vi siano tracce di cingolati dalla più forte intensità escludendo quelle lasciate da essi stessi. In loro assenza, ripiegano per una scelta casuale. (Sugg.: sfruttare i servizi del seguente metodo di **`war.Campo`**:
`int rilevaTracciaVerso(Coordinate riferimento, Direzione dir)`)

c) **(10%)** Ciascun carrarmato possiede un intero identificatore ("**`id`**") progressivo base 0 assegnato sulla base della propria fazione, ed incrementato ogni qualvolta che un nuovo esemplare di quella tipologia di tank viene creato: scrivere i test di unità presenti nella classe **`war.tank.TankTest`** (metodi test-case di nome **`testIdProgressivi…()`**) a supporto e precisazione di questo requisito. Correggere il codice principale affinché i test scritti abbiano sempre successo e risultino perfettamente *isolati*, ovvero producano lo stesso risultato ed abbiano successo indipendentemente dall'insieme dei test di unità con i quali vengono eseguiti.

Le domande che seguono richiedono il completamento di metodi che calcolano delle statistiche stampate a video al termine di ciascuna simulazione. Gli scheletri dei metodi sono già presenti nel codice fornito e vanno opportunamente completati. Sono anche già forniti a supporto dei metodi di stampa dei loro risultati per facilitarne il controllo e l'ispezione. Studiare il sorgente della classe **`war.simulatore.Statistiche`** ed in particolare il metodo **`stampaStatisticheFinali()`** per i dettagli.

DOMANDA 3 (10%)
Dopo aver completato il punto precedente:

a) completare il codice del metodo **`Map<Tank, Integer> distruttiPerTank(Set<Proiettile>)`** nella classe **`Statistiche`**. In particolare questo metodo deve scandire l'elenco degli oggetti di tipo **`Proiettile`** che riceve come parametro, e contare il numero di tank distrutti, associandolo al corrispondente oggetto **`Tank`** distruttore.
(Sugg.: considerare i metodi di **`Proiettile getLanciatore()/getDistrutto()`**, entrambi che restituiscono un **`Tank`**).

E' possibile, ma solo se ritenuto necessario, modificare anche il codice della classe **`Proiettile`** e della classe **`Tank`**.

DOMANDA 4 (10%)
Dopo aver completato il punto precedente:

a) completare il codice del seguente metodo all'interno della classe **`Statistiche`**
`Map<Fazione, Set<Proiettile>> vulnerabilitaPerTipo(Set<Proiettile>)`. In particolare questo metodo deve scandire la collezione di oggetti **`Proiettile`** che riceve come parametro, e raggrupparli per tipologia del tank che *hanno distrutto*, associando un insieme di proiettili all'oggetto **`Fazione`** rappresentate ciascuna delle tipologia di tank distrutti.
(Sugg.: si consideri il metodo **`Proiettile#getDistrutto()`**).

E' possibile, ma solo se ritenuto necessario, modificare anche il codice della classe **`Proiettile`** e della classe **`Tank`**.

DOMANDA 5 (10%)
Completare il metodo **`List<Fazione> ordinaStrategieDiCombattimento()`** presente nella classe **`Statistiche`**.
Restituisce la classifica di tutte le tipologie di tank (rappresentati ancora dal corrispondente oggetto **`Fazione`**) sulla base delle tipologie di tank che sono riusciti a distruggere. Utilizzare un criterio di ordinamento *esterno*.
(Sugg.: provare a riutilizzare quanto già scritto in risposta alle precedenti domande come indicato anche dalla segnatura del metodo).

DOMANDA 6 (20%)
Scrivere una nuova classe di test **`war.CampoTest`** (posizionandola all'interno della directory del progetto **`war/test`**) con almeno tre test-case *minimali* per verificare il corretto funzionamento del metodo già esistente all'interno della classe **`Campo`**:

`int rilevaTracciaVerso(Coordinate riferimento, Direzione dir)`

E' possibile, ma solo se ritenuto conveniente e senza compromettere il resto del progetto, modificare anche il codice della classe **`Campo`** per favorirne la *testabilità*.

DOMANDA 7 (+10%) *FACOLTATIVA*
Aggiungere una nuovo sottotipo di **`Tank`** alla simulazione: i **`Gunner`** (di colore blue) si muovono casualmente ma decidono di sparare solo quando pensano di poter colpire un nemico (ovvero, si trovano su una cella che risulti, nella direzione corrente, orizzontalmente o verticalmente allineata all'avversario).