Un laboratorio sta sviluppando un applicativo a supporto della simulazione software. Si è interessati a studiare la coesistenza di oggetti in un ambiente popolato da oggetti appartenenti a diverse tipologie con l'obiettivo di collezionare alcuni dati statistici. Inizialmente sono previsti oggetti di due tipologie diverse (rispettivamente *bianco* e *giallo* sul display). I bianchi si muovono casualmente nell'ambiente che popolano, mentre i gialli (e tutte gli altri oggetti delle tipologie che seguiranno) si dirigono verso un obiettivo.

La simulazione è gestita dal metodo **`simulazione.modello.Simulatore.simula()`** e si articola in una sequenza di passi discreti svolti all'interno di un ambiente delimitato (**`simulazione.modello.Ambiente`**). Gli oggetti si spostano occupando posizioni (**`simulazione.modello.Coordinate`**), anche coincidenti, di un piano cartesiano, spostandosi sempre e soltanto in una cella adiacente a quella corrente (quindi senza "salti"). Quando due oggetti occupano la stessa posizione, si genera un evento modellato da un apposito oggetto **`simulazione.modello.Contatto`** che viene creato e registrato da **`simula()`** allo scopo.

DOMANDA 1 (5%)
Modificare il codice della classe **`Coordinate`** affinché i test presenti nelle classi **`CoordinateTest`** comincino ad avere successo. Già dopo aver effettuato questa correzione, e` possibile verificare il corretto funzionamento dell'intera simulazione eseguendo il metodo **`main()`** della classe **`simulazione.Main`**.

(N.B. Per una più agevole comprensione della descrizione che segue, si consiglia di provare ad eseguire il metodo `main()` della classe `simulazione.Main`, osservare l'animazione della simulazione già dopo aver risposto a questa prima domanda, premendo il tasto ESCape dopo qualche secondo. *La simulazione stampa, a fine esecuzione, alcune statistiche raccolte durante l'esecuzione. Queste statistiche sono oggetto delle domande successive: è possibile premere il tasto ESCape per anticipare la fine della simulazione e la stampa delle statistiche in qualsiasi momento, anche senza attendere la terminazione di tutta la simulazione.)*

Inizialmente si studia una solo tipo di oggetto, modellata dalla classe **`Bianco`** del package **`simulazione.esemplare`**. Successivamente si introdurranno nella simulazione anche altri tipi che si differenziano per il comportamento. Ad esempio viene fornita la classe **`Giallo`** inizialmente ancora identica a **`Bianco`**.

Un progettista esperto fa notare al programmatore che molte linee di codice sono in comune tra la classe **`Bianco`** e la classe **`Giallo`** (già esistente), e che la stessa situazione si presenterà con nuove tipologie che si dovranno prevedibilmente aggiungere. Suggerisce di ristrutturare il codice introducendo una classe astratta **`Esemplare`** che accomuni gli oggetti di tutte le tipologie.

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

a) **(20%)** Introdurre una classe astratta **`simulazione.esemplare.Esemplare`** per generalizzare gli oggetti di ogni tipologia.
Modificare la classe **`Bianco`** e la classe **`Giallo`** di conseguenza. Cambiare il codice (in particolare il corpo del metodo **`simulazione.modello.Simulatore.popolaAmbiente()`**) di modo che anche gli esemplari di questa ed ogni altra nuova tipologia richiesta entrino a pieno titolo nella simulazione.

Un intero identificatore ("**`id`**") progressivo base 0 sia assegnato ad ogni nuovo oggetto secondo questi requisiti:
- ✔ Ogni tipologia possiede una progressione di identificatori distinta dalle progressioni usata per tutte le altre tipologie
- ✔ il numero progressivo viene incrementato ogni qualvolta un nuovo esemplare della stessa tipologia (e solo di quella) viene creato

A supporto e precisazione di questi requisiti viene richiesto di completare i test di unita` gia' presenti nella classe **`simulazione.esemplare.EsemplareTest`**.

b) **(10%)** Modificare il comportamento della classe **`Giallo`** (di colore giallo): i gialli selezionano un altro esemplare "obiettivo" quando sono creati e si spostano solo orizzontalmente mentre cercano di inseguirlo, al contrario, non possono spostarsi verticalmente E' loro obiettivo un esemplare di tipo **`Bianco`** scelto casualmente. Quindi modificare sia il costruttore della classe **`Giallo`** sia il metodo **`Giallo.mossa()`** per implementare la logica desiderata.

c) **(10%)** Creare la classe associata ad una nuova tipologia **`Rosso`** (di colore rosso, vedi anche **`simulazione.gui.CostantiGUI.RISORSA_IMMAGINE_ROSSO`**): tutti gli esemplari di questo tipo scelgono come obiettivo l'esemplare più lontano.

d) **(10%)** Creare la classe associata ad una nuova tipologia **`Verde`** (di colore verde, vedi **`RISORSA_IMMAGINE_VERDE`**): i verdi selezionano un altro esemplare **`Rosso`** quando sono creati e si spostano solo verticalmente mentre cercano di inseguirlo, al contrario, non possono spostarsi orizzontalmente.

Le domande che seguono richiedono il completamento del corpo di metodi nella classe **`simulazione.statistiche.Statistiche`**: questi sono dedicati al calcolo di alcune statistiche al termine di ciascuna simulazione. A supporto, sono anche forniti dei metodi di stampa dei risultati per facilitare la verifica manuale del corretto funzionamento. Si suggerisce di studiare il sorgente della classe **`Statistiche`** ed in particolare il metodo **`stampaStatistiche()`** per i dettagli.

DOMANDA 3 (30%)
Dopo aver completato il punto precedente:

- completare il corpo del metodo nella classe **`Statistiche`**:
  `public Map<Class<? extends Esemplare>,SortedSet<Contatto>> produciStatistiche(Collection<Contatto> contatti)`.
  Questo metodo deve scandire la collezione degli oggetti di tipo **`Contatto`** che riceve come parametro, per poi associare ad ogni tipologia tutti e soli i contatti a cui hanno partecipato. L'insieme è ordinato per il passo in cui i contatti si sono verificati e deve ammettere la presenza di diversi contatti che siano avvenuti nel medesimo passo della simulazione.
  E' possibile, ma solo se ritenuto necessario, modificare anche il codice della classe **`Contatto`** e della classe **`Esemplare`**.

- completare il corrispondente test-case **`testProduciStatistiche()`** all'interno di **`StatisticheTest`**

DOMANDA 4 (20%)
Scrivere una o più classi di test (posizionandole corrispondentemente alla classe sotto test, e denominandole di conseguenza: ovvero all'interno della directory **`test/simulazione/esemplare`** e chiamandola, ad esempio, **`RossoTest`**, **`VerdeTest`**), con test-case *minimali* per verificare il corretto funzionamento dei metodi che si occupano della scelta della prossima mossa. Ripetere l'esercizio per quante più tipologie possibili. E' possibile, ma solo se ritenuto conveniente e senza compromettere il resto del progetto, modificare anche il codice della classi che modellano il comportamento delle diverse tipologie per favorirne la *testabilità*.