Un laboratorio di ricerca sta sviluppando un software per la simulazione di un ecosistema popolato da animali appartenenti a diverse specie e di diverso genere (modellato dal tipo enumerativo **`gen.sim.Genere`**): queste simulazioni hanno l'obiettivo di collezionare alcuni dati statistici utili a futuri studi sulla biologia delle specie simulate.

Il laboratorio è interessato a studiare la coesistenza di animali di due specie (quella *bianca* e quella *gialla* sul display di visualizzazione) che scelgono e si direzionano verso un altro animale "obiettivo" rispettivamente della propria specie, a scopo riproduttivo (generando un evento modellato dalla classe **`gen.sim.Incontro`**, o di una specie diversa, a scopo predatorio, generando un evento modellato dalla classe **`gen.sim.Scontro`**.

La simulazione è gestita dal metodo **`gen.sim.Simulatore.simula()`** e si articola in un sequenza di passi discreti si svolge all'interno di un ambiente ben delimitato (modellata dalla classe **`gen.sim.Ambiente`**), dove gli animali diversi si spostano occupando posizioni, anche coincidenti, di un piano cartesiano come modellato dalla classe **`gen.sim.Coordinate`**. Gli spostamenti, orizzontali o verticali, avvengono lungo le direzioni modellate dall'enum **`gen.sim.Direzione`**, e sono tutti di una cella alla volta.

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

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

Inizialmente si studia una sola specie, modellata dalla classe **`Bianco`** del package **`gen.tipo`**, ma successivamente sarà necessario introdurre anche la classe **`Giallo`** che finisce per differenziarsi per i comportanti diversi degli animali appartenenti alle due specie.

La simulazione comincia con la creazione di un pari numero di animali per ciascuna tipologia; ciascuno possiede una posizione di partenza ed una destinazione (quella di un altro animale nello stesso ambiente) ed in ogni passo della simulazione (vedi anche il metodo **`gen.tipo.Bianco.simula(int)`**) si avvicina verso la propria destinazione. A destinazione, od anche prima lungo il percorso, diversi animali possono finire per occupare la medesima posizione, ed a seconda della specie, del genere, e del numero di animali coinvolti, possono generarsi oggetti **`gen.sim.Incontro`** od oggetti **`gen.sim.Scontro`** che sono conservati per il successivo calcolo delle statistiche (vedi metodo **`gen.sim.Simulatore.simula()`**)

All'arrivo a destinazione ciascun animale decide una nuova destinazione secondo l'indole della propria specie. Tale scelta nella classe **`Bianco`** è delegata al metodo **`decidiProssimoObiettivo()`**.

Un progettista esperto fa notare al programmatore che esisterebbero molte linee di codice in comune tra la classe **`Bianco`** (già esistente) e la classe **`Giallo`** ancora da creare e suggerisce, visto che prevedibilmente altre specie potrebbero entrare a far parte della simulazione in futuro, di ristrutturare il codice introducendo un nuovo tipo astratto **`gen.tipo.Animale`** che accomuni tutte le specie, presenti e future.

DOMANDA 2 (60%)
Pertanto il progettista esperto suggerisce di ristrutturare l'applicazione come segue:

a) **(30%)** Introdurre una classe astratta **`gen.tipo.Animale`** per generalizzare gli animali di tipo **`Bianco`**, che scelgono come destinazione un'animale qualsiasi purche' della stessa specie. Nelle classi che facevano uso del primo tipo, generalizzare il codice usando solo il nuovo tipo astratto al posto del precedente tipo concreto. Si segnalano *almeno* questi utilizzi: metodi **`gen.sim.Simulatore.simula()`**, **`gen.gui.GUI.disegna()`**, ove alcune operazioni vanno opportunamente generalizzate per funzionare indipendentemente dal tipo dinamico degli oggetti coinvolti. Ciascun animale possiede un intero identificatore ("**`id`**") progressivo base 0 assegnato sulla base della propria tipologia, ed incrementato ogni qualvolta un nuovo esemplare di quella tipologia viene creato: a supporto e precisazione di questo requisito completare e ricorrere all'esecuzione dei test di unita` presenti nella classe **`gen.tipo.AnimaleTest`** (test-case **`testIdProgressiviPerBianchi()`**). Verificare ed eventualmente correggere il codice principale affinche' questi test abbiano sempre successo. Implementare le modifiche suggerite dal progettista esperto, quindi verificare il funzionamento dell'applicazione anche dopo le modifiche.

b) **(10%)** Creare la classe associata alla specie aggiungendo la classe **`Giallo`**: che scelgono come obiettivo un animale di una specie diversa (al momento, quindi, certamente di tipo **`Bianco`**, ma in futuro anche di altre specie). Cambiare il codice (in particolare il corpo del metodo **`gen.sim.Simulatore.creaPopolazione()`**) di modo che anche animali di nuove specie entrino a pieno titolo nella simulazione. Completare i test di unita` presenti nella classe **`gen.tipo.AnimaleTest`** (test-case **`testIdProgressiviPerAltroColore()`**) affinche' tutti i test funzionino.

c) **(10%)** Creare la classe associata alla nuova specie aggiungendo la classe **`Verde`**: tutte gli animali di questo tipo si comportano come **`Giallo`** prima dell'eta' di riproduzione (vedi costante **`gen.sim.CostantiSimulazione.MIN_ETA_RIPRODUZIONE`**) come **`Bianco`** dopo.

d) **(10%)** Creare la classe associata alla nuova specie aggiungendo la classe **`Rosso`**: tutte gli animali di questo tipo si comportano scelgono come obiettivo l'animale più vicino tra quelli di specie diversa per originare un scontro nel più breve tempo possibile (Sugg.: vedere il metodo **`gen.sim.Coordinate.distanza(Coordinate, Coordinate)`**, e considerare anche il metodo **`static Double.compare(double d1, double d2)`**).

Le domande che seguono richiedono il completamento del corpo di alcuni metodi nella classe **`gen.sim.Statistiche`**: questi sono dedicati al calcolo delle statistiche al termine di ciascuna simulazione. Sono anche gia` forniti a supporto dei metodi di stampa dei loro risultati per facilitarne la verifica del corretto funzionamento. Si suggerisce di studiare il sorgente della classe **`gen.sim.Statistiche`** ed in particolare il metodo **`stampaFinale()`** per i dettagli.

DOMANDA 3 (20%)
Dopo aver completato il punto precedente:

- completare il codice del metodo **`Map< Animale, List< Scontro > > scontriPerAnimale(Set< Scontro >)`** nella classe **`Statistiche`**. In particolare questo metodo deve scandire l'insieme degli oggetti di tipo **`Scontro`** che riceve come parametro, e raggrupparli sulla base dell'animale oggetto istanza di **`Animale`** che ha vinto tale scontro.
  E' possibile, ma solo se ritenuto necessario, modificare anche il codice della classe **`Scontro`** e della classe **`Animale`**.
  (Sugg.: usare il metodo **`gen.sim.Scontro.getVincente()`**).

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

DOMANDA 4 (20%)
Scrivere una o piu` classi di test (posizionandole corrispondentemente alla classe sotto test, e denominandole di conseguenza: ovvero all'interno della directory **`test/gen/tipo`** e chiamandola, ad esempio, **`gen.tipo.GialloTest`**), con test-case *minimali* per verificare il corretto funzionamento dei metodi che si occupano della scelta della prossimo obiettivo. Ripetere l'esercizio per quante piu' specie 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 specie per favorirne la *testabilita`*.

DOMANDA 5 (SOLO PER STUDENTI CHE SOSTENGONO L'ESAME DA 9 CFU +10%)
Il programma termina quando viene premuto il tasto ESCape. Questo dovrebbe causare anche la stampa del messaggio "`Richiesta la terminazione della simulazione`" che invece spesso non appare perche' il programma termina prima ancora che il thread adibito alla stampa di tale messaggio arrivi ad eseguirla.
Cambiare il codice del metodo **`simula()`** all'interno della classe **`Simulatore`** affinche' la stampa del messaggio avvenga sempre e comunque nel corretto ordine, ovvero prima delle stampe finali delle statistiche (Non e` possibile cambiare il codice delle altre classi per rispondere a questa domanda).