Un'azienda che gestisce un servizio di *car sharing* sta sviluppando un software per la simulazione delle attivita` del proprio parco vetture: queste simulazioni hanno l'obiettivo di collezionare alcuni dati statistici per supportare future decisioni aziendali tese a diversificare l'offerta e rendere il servizio più efficiente.

L'azienda è interessata a studiare cosa accadrebbe se offrisse una parte delle proprie vetture (auto *bianche* sul display di visualizzazione) ad una tariffa più elevata ma con totale libertà di scelta della destinazione, ed uno stesso numero di vetture ad un prezzo ridotto (auto *gialle* sul display di visualizzazione), ma con un insieme di destinazioni molto più ridotto e prefissato (vedi costante **`car.sim.CostantiSimulazione.N_DESTINAZIONI_GIALLE`**) , e comune a tutte le vetture gialle.

Le vetture si muovono all'interno di una zona di copertura ben delimitata (modellata dalla classe **`car.sim.Zona`**), occupando posizioni di un piano cartesiano modellate dalla classe **`car.sim.Coordinate`**. La direzione degli spostamenti, che sono incrementali, orizzontali o verticali, viene modellata dall'enum **`car.sim.Direzione`**.

DOMANDA 1 (5%)
Modificare il codice della classe **`car.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 **`car.Main`**.

(N.B. Per una piu` agevole comprensione della descrizione che segue, si consiglia di provare ad eseguire il metodo **`main()`** della classe **`car.Main`** ed osservare l'animazione della simulazione gia` dopo aver risposto a questa prima domanda. La simulazione stampa, a fine esecuzione, alcune statistiche raccolte durante l'esecuzione. Queste 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.

Inizialmente si studia una sola tipologia di vettura, modellata dalla classe **`Bianca`** del package **`car.auto`**, ma successivamente sarà necessario introdurre anche la classe **`Gialla`** che finisce per differenziarsi dalla prima solamente la politica di scelta della destinazione.

La simulazione comincia con la creazione dello stesso numero di vetture per ciascuna tipologia; ciascuna ha una posizione di partenza ed una di destinazione ed in ogni passo della simulazione (vedi anche il metodo **`car.auto.Bianca.simula()`**) si avvicina incrementalmente verso la propria destinazione spostandosi sul piano cartesiano di una singola cella, orizzontalmente o verticalmente.

All'arrivo a destinazione il tragitto percorso viene registrato e ciascuna vettura deve decidere una nuova destinazione secondo la politica della propria tipologia (ovvero prendendo una destinazione casuale qualsiasi se bianca, mentre le vetture gialle devono scegliere una destinazione comunque casuale ma selezionata tra quelle comuni a tutte le auto gialle). Tale scelta nella classe **`Bianca`** è delegata al metodo **`decidiProssimaDestinazione()`**.

Un progettista esperto fa notare al programmatore che esisterebbero molte linee di codice in comune tra la classe **`Bianca`** (già esistente) e la classe **`Gialla`** ancora da creare e suggerisce, visto che prevedibilmente altre offerte potrebbero essere studiate in futuro, di ristruttare il codice introducendo un nuovo tipo astratto **`car.auto.Auto`** che accumuni tutte le tipologie di vetture, presenti e future.

DOMANDA 2 (35%)
Pertanto il progettista esperto suggerisce di ristrutturare l'applicazione come segue:

- **(20%)** Introdurre una classe astratta **`car.auto.Auto`** per generalizzare le vetture di tipo **`Bianca`** ed **`Gialla`**. Nelle classi che facevano uso del primo tipo, generalizzare il codice usando solo il nuovo tipo astratto al posto dei precedenti tipi concreti. Si segnalano *almeno* questi utilizzi: metodi **`car.sim.Simulatore.simula()`**, **`car.gui.GUI.disegnaAuto()`**, ove alcune operazioni vanno opportunamente generalizzate per funzionare indipendentemente dal tipo dinamico delle vetture coinvolte. Ciascuna vettura possiede un intero identificatore ("**`id`**") progressivo base 0 assegnato sulla base della propria tipologia, ed incrementato ogni qualvolta un nuovo esemplare di quella tipologia di vettura viene creato: a supporto e precisazione di questo requisito completare e ricorrere all'esecuzione dei test di unita` presenti nella classe **`car.auto.AutoTest`** (test-case **`testIdProgressivi...()`**). 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.

- **(15%)** Creare la classe associata alla nuova tipologia di offerta aggiungendo la classe **`Gialla`**: tutte le vetture di questo tipo possiedono un riferimento verso lo stesso elenco di destinazioni possibili e possono spostarsi solo tra queste. Cambiare il codice (in particolare il corpo del metodo **`car.sim.Simulatore.creaVetture()`**) di modo che anche i nuovi tipi di vettura entrino a pieno titolo nella simulazione.
(Sugg.: sfruttare anche i servizi del metodo **`car.sim.GeneratoreCasuale.generaNposizioniCasuali()`**)

Le domande che seguono richiedono il completamento dei metodi nella classe **`car.sim.Statistiche`** che calcolano le statistiche stampate a video al termine di ciascuna simulazione. Gli scheletri dei metodi sono gia` presenti nel codice fornito e vanno opportunamente completati. Sono anche gia` forniti a supporto dei metodi di stampa dei loro risultati per facilitarne l'ispezione. Si suggerisce di studiare il sorgente della classe **`car.sim.Statistiche`** ed in particolare il metodo **`stampaFinale()`** per i dettagli.

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

- completare il codice del metodo **`Map< Auto, Set< Tragitto > > tragittoPerAuto(List< Tragitto >)`** nella classe **`Statistiche`**. In particolare questo metodo deve scandire l'elenco degli oggetti di tipo **`Tragitto`** che riceve come parametro, e raggrupparli sulla base della vettura che li ha percorsi, associandoli al corrispondente oggetto **`Auto`**
(Sugg.: si consideri il metodo **`car.sim.Tragitto.getAuto()`**).

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

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

DOMANDA 4 (20%)
Completare il metodo **`SortedMap< Coordinate,Integer > utilizzi(List< Tragitto > percorsi)`** presente nella classe **`Statistiche`**.
Restituisce la classifica (modellata dalla **`SortedMap`**) delle posizioni più frequentemente utilizzate come destinazione e/o origine di uno dei tragitti ricevuti nella lista passata come parametro. Ad ogni posizione che figura come chiave nella mappa corrisponde come valore un oggetto **`Integer`** pari al numero complessivo di utilizzi, da parte di una qualsiasi vettura che abbia participato alla simulazione. Utilizzare un criterio di ordinamento *esterno* alla classe **`Coordinate`**.

DOMANDA 5 (20%)
Scrivere una nuova classe di test **`car.sim.ZonaTest`** (posizionandola corrispondentemente alla classe sotto test **`Zona`**, ovvero all'interno della directory **`car/test/car/sim`**) con almeno tre test-case *minimali* per verificare il corretto funzionamento del metodo:

`Set< Direzione > car.sim.Zona.getPossibiliDirezioni(Coordinate riferimento)`

metodo gia` presente all'interno della classe **`Zona`**. E' possibile, ma solo se ritenuto conveniente e senza compromettere il resto del progetto, modificare anche il codice della classe **`Zona`** per favorirne la *testabilita`*.

DOMANDA 6 (SOLO PER STUDENTI CHE SOSTENGONO L'ESAME DA 9 CFU +20%)

Sia data una matrice quadrata **`int[][] utilizzi`** che riporta ad ogni riga e colonna di indice **`i,j`** il numero di tragitti che hanno avuto, durante la simulazionte, origine e/o destinazione nella posizione di coordinate **`i,j`**.
Utilizzando il metodo per il calcolo del massimo su matrici di enormi dimensioni già presente nella classe **`car.stats.CalcolatoreMaxSeriale`** ed avente segnatura:
`int max(int[][] utilizzi, int colonnaIniziale, int colonnaFinale)`
implementare all'interno della classe **`CalcolatoreMaxParallelo`** il metodo di segnatura:
`int max(int[][] utilizzi)`
che restituisce sempre il max di una matrice quadrata, ma tramite un algoritmo di decomposizione parallela teso a sfruttare l'eventuale disponibilità di un'architettura multi-core.