Un'azienda che gestisce un servizio di *bike sharing* sta sviluppando un software per la simulazione delle attivita` del proprio parco bici: 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 biciclette (quelle *bianche* sul display di visualizzazione) ad una tariffa più elevata ma con totale libertà di scelta della destinazione, ed uno stesso numero di biciclette ad un prezzo ridotto (quelle *gialle* sul display di visualizzazione), ma con un insieme di destinazioni molto più ridotto e prefissato (vedi costante **`bici.sim.CostantiSimulazione.N_DESTINAZIONI`**) , e comune a tutte le biciclette di questo colore.

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

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

(N.B. Per una piu` agevole comprensione della descrizione che segue, si consiglia di provare ad eseguire il metodo **`main()`** della classe **`bici.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 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 tipologia di bicicletta, modellata dalla classe **`Bianca`** del package **`bici.tipo`**, ma successivamente sarà necessario introdurre anche la classe **`Gialla`** che finisce per differenziarsi dalla prima solamente per la politica di scelta della destinazione.

La simulazione comincia con la creazione dello stesso numero di mezzi per ciascuna tipologia; ciascuna ha una posizione di partenza ed una di destinazione ed in ogni passo della simulazione (vedi anche il metodo **`bici.tipo.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 percorso coperto viene registrato e ciascun mezzo decide una nuova destinazione secondo la politica della propria tipologia (prendendo una destinazione casuale qualsiasi se bianca, ovvero scegliendo una destinazione comunque casuale ma selezionata tra quelle comuni a tutti mezzi dello stesso colore, se di altro colore). 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 **`bici.tipo.Bici`** che accumuni tutte le tipologie di biciclette, sia presenti sia future.

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

- **(20%)** Introdurre una classe astratta **`bici.tipo.Bici`** per generalizzare le biciclette di tipo **`Bianca`** ed **`Gialla`**. 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 **`bici.sim.Simulatore.simula()`**, **`bici.gui.GUI.disegnaBici()`**, ove alcune operazioni vanno opportunamente generalizzate per funzionare indipendentemente dal tipo dinamico delle biciclette coinvolte. Ciascuna 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 **`bici.tipo.BiciTest`** (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 biciclette 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 **`bici.sim.Simulatore.creaBiciclette()`**) di modo che anche i nuovi tipi di bicicletta entrino a pieno titolo nella simulazione.
  (Sugg.: sfruttare anche i servizi del metodo **`bici.sim.GeneratoreCasuale.generaNposizioniCasuali()`**)

Le domande che seguono richiedono il completamento di alcuni corpi dei metodi nella classe **`bici.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 **`bici.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< Bici, List< Percorso > > percorsoPerBici(Set< Percorso >)`** nella classe **`Statistiche`**. In particolare questo metodo deve scandire l'insieme degli oggetti di tipo **`Percorso`** che riceve come parametro, e raggrupparli sulla base della bicicletta che li ha percorsi, associandoli al corrispondente oggetto **`Bici`**.
  E' possibile, ma solo se ritenuto necessario, modificare anche il codice della classe **`Percorso`** e della classe **`Bici`**.
  (Sugg.: usare il metodo **`bici.sim.Percorso.getBici()`**; si consideri anche che alcune bici possono ripetere lo stesso percorso più volte e che il risultato di questo metodo alimenta il metodo **`utilizzi()`** oggetto della prossima domanda).

- completare il corrispondente test-case **`testPercorsoPerBici()`** all'interno di **`StatistricheTest`**

DOMANDA 4 (25%)
Completare il metodo **`SortedMap< Coordinate,Integer > utilizzi( Map< Bici, List< Percorso > >)`** presente nella classe **`Statistiche`**.
Riceve la mappa dei percorsi effettuati da ciascuna bici (come restituito dal metodo **`percorsoPerBici()`** scritto in risposta alla domanda precedente) e restituisce la classifica (modellata dalla **`SortedMap`**) delle posizioni più frequentemente utilizzate come destinazione e/o origine di uno dei percorsi 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 bicicletta che abbia participato alla simulazione. Se una stessa bici ripete lo stesso percorso diverse volte, ciascun percorso va contato separatamente. Utilizzare un criterio di ordinamento *esterno* alla classe **`Coordinate`**.
(Sugg.: Assicurarsi che oggetti rappresentanti coordinate distinte non finiscano per essere esclusi dalle chiavi del risultato solo perché aventi lo stesso numero di utilizzi di altri).

DOMANDA 5 (20%)
Scrivere una nuova classe di test **`bici.tipo.GiallaTest`** (posizionandola corrispondentemente alla classe sotto test, ovvero all'interno della directory **`test/bici/tipo`**) con almeno due test-case *minimali* per verificare il corretto funzionamento del metodo che si occupa della scelta della prossima destinazione. E' possibile, ma solo se ritenuto conveniente e senza compromettere il resto del progetto, modificare anche il codice della classe **`Gialla`** per favorirne la *testabilita`*.

DOMANDA 6 (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 sua stampa arrivi ad eseguirla.
Cambiare il codice del metodo **`simula()`** all'interno della classe **`Simulatore`** affinché le stampa del messaggio avvenga sempre (Non e` possibile cambiare il codice delle altre classi per rispondere a questa domanda).