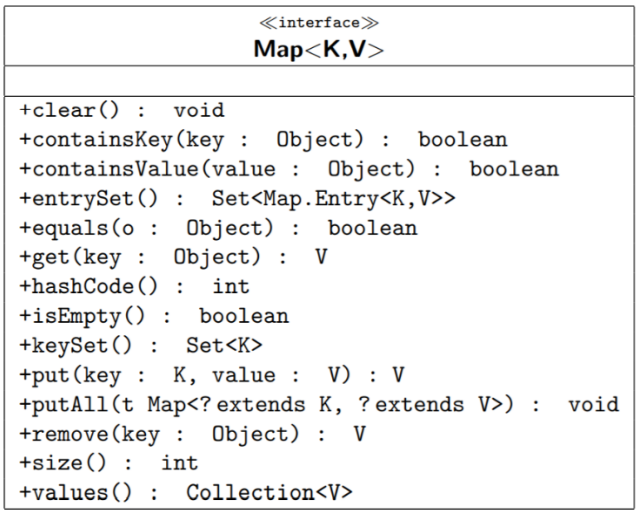
Data Structures and Algorithms: Assignment 3

Due: Monday 22nd June, 11:59pm

50 Marks Possible – worth 15% of final grade

Question 1) Map implementation – 25 marks

A *map* is an ADT (Abstract Data Type) for fast key lookups, where an identifier key (K) is used to retrieve an associated content value (V). A map can be defined by the following interface – available in the java collections API.



Create an implementing class called *LinkedHashMapWithChaining*. This class should store key-entry values in your **own** underlying *hashtable*, where each key-value entry is stored in an index determined by the *hashcode* of the key. Your class should maintain a *load factor* of <75%, otherwise it will have to expand capacity. Your class should resolve any collisions with *chaining*. An advantage of your class is that it also maintains a doubly-linked structure (your own), so the *toString*, *keySet, entrySet* and *values* methods should give the *key* or *value* pairs in the **order in which they were put into the map.** Make sure all methods are safe from unexpected events using appropriate exception handling (example, if some user code tries to remove something from the map which does not exist). Feel free to look at the *Map* Java documentation to get a full understanding of how these methods should operate. Create a test class with a suitable *main* method which effectively tests most of the operations of your *LinkedHashMapWithChaining* class.

Question 2) Luxury Cruises – 25 marks

Obtain the class called *CruiseShip* (available from Blackboard) which represents a single trip for a cruise ship travelling between two different ports and dates. The ship has a cost and unique ship name and leaves from the departure port on the *departDate*, arriving at the arrival port on the *arriveDate*. All this information is passed into the constructor. The class has getter methods for each and a *toString* to obtain a useful string representation for the object. A *CruiseShip* is considered equal to another if the arrival/departure date, port, and ship names are the same.

Using the following UML diagram below create a class called *CruiseJourney* that represents a journey comprised of one or more *CruiseShip* objects between multiple ports. The class encapsulates a *List* data structure of *CruiseShip* objects, keeping the dated order of the necessary trips which comprise a complete journey. There are two constructors, firstly a default constructor creating a journey with no initial trips, secondly a constructor that adds an existing list of *CruiseShip* objects to the underlying data structure.

The *addCruise* method should add a *CruiseShip* to the journey, returning true and only adding the trip to the current journey if it meets the following criteria:

- The journey's end port is equal to the newly added *CruiseShip* parameter departure port (so the new cruise departs from the same port as journey's current end port).
- The journey's end date is earlier in time to the newly added CruiseShip parameter's departure date (so the new cruise cannot be added if its departure date is earlier than the journey's current end date) – You may assume that all departures are in the afternoon, and all arrivals in the morning (so dates that are the same are valid).
- The journey so far does not already contain the parameters arrival port somewhere in its shipList (meaning the same port is never visited twice – no closed paths).

The *containsPort* method should return true if the given port is in the journey. The getter methods return the start and end ports and dates (or null) of the Journey (so start port and date are the first *CruiseShip's* departure values, whereas end port and date are the last CruiseShip's arrival values). The *removeLastTrip* method removes the lastly added *CruiseShip* from the current journey (if any). The *getTotalCost* returns the total cost of all the *CruiseShip* trips in this journey. The *getNumberOfTrips* returns the number of CruiseShips which comprise the journey. The *toString* method should print out a string representation of all trips in this journey and the total cost in a nicely formatted way. The *cloneJourney* method returns a new *CruiseJourney* object, passing its *shipList* to the new instance by calling the second constructor.

CruiseJourney
- shipList : List<CruiseShip>
+ CruiseJourney() + CruiseJourney(list : List<CruiseShip>) + addCruise(ship : CruiseShip) : boolean + removeLastTrip() : boolean + containsPort(port : String) : boolean + getStartPort() : String + getEndPort() : String + getStartDate() : Calendar + getEndDate() : Calendar + cloneJourney() : CruiseJourney + getNumberOfTrips() : int + getTotalCost() : double + toString() : String

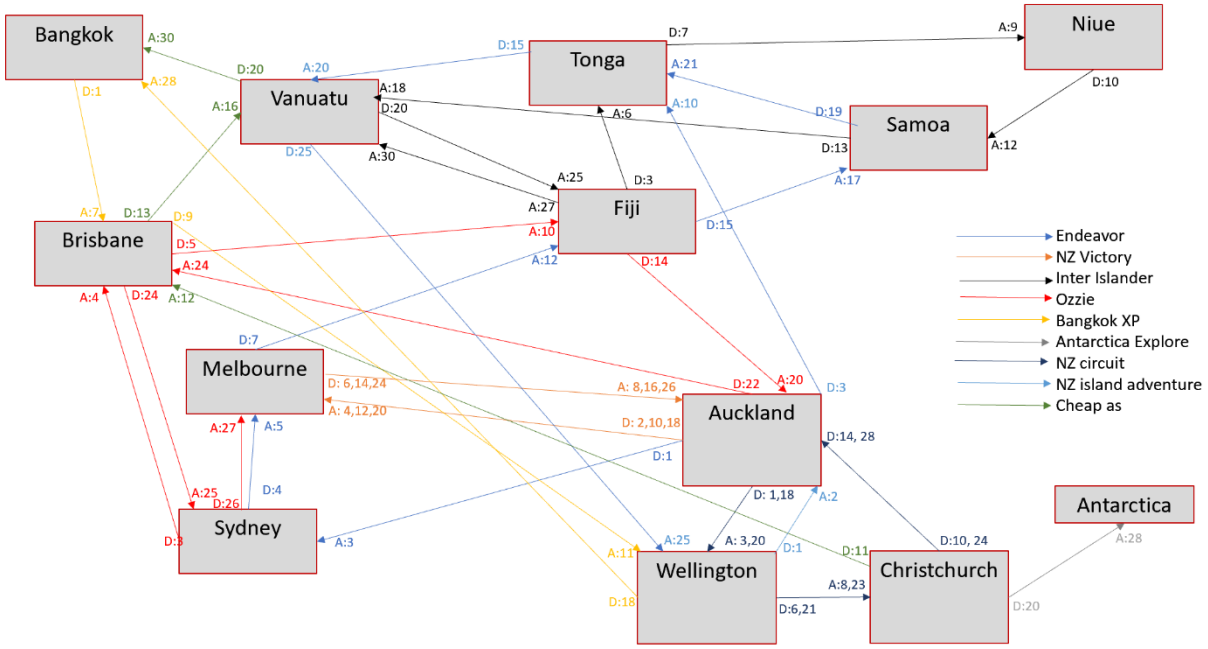
Using the following UML, create a class called *LuxuryCruiseCentre*. The class holds an efficient *Map* data structure of unique ports as key and a **set** of *CruiseShip* instances which depart that port. The add method is used to add a unique *CruiseShip* to this map (care taken when the departure port already exists in the portMap).

The *getPossibleJourneys* method is used to return list of all the uniquely possible routes from the start port and date to the end port. It does this by calling the recursive method *findPaths* which uses

a *depth first search* technique to build up all the possible journeys that can be undertaken between the start and end port (if any). It does this by using the *currentJourney* parameter and when the target port is found it will "clone" the current journey and add it to the List of *CruiseJourney* values found.

LuxuryCruiseCentre
- portMap : Map<String, Set<CruiseShip>>
+ LuxuryCruiseCentre() + add(ship: CruiseShip) : boolean + getPossibleJourneys(startPort:String, startDate:Calendar, endPort:String) : List<CruiseJourney> -findPaths(departPort:String, departDate: Calendar, endPort:String, current:Journey, journeyList: List<CruiseJourney>):void

Once developed, a suitable test class (available on Blackboard) can be used to find all *CruiseJourney's* between any two ports. The test class adds numerous *CruiseShip* objects to the *LuxuryCruiseCentre*. The graph below provides a visual representation showing the different cruise ships travelling between ports with depart and arrival dates (all in June 2020). Use this test class to test different scenarios of travel. If you are having problems and for simple debugging, try using only a **small subset** of these trips and build your own smaller graph before attempting this one.



Example: Leaving Auckland to Bangkok on the 1st June: There are 5 possible journeys:

Journey 1: Total Cost $16299.0!!!
- NZ Island Adventure: LEAVING Auckland (3-June) and ARRIVING Tonga (10-June)
- NZ Island Adventure: LEAVING Tonga (15-June) and ARRIVING Vanuatu (20-June)
- Cheap As Chips: LEAVING Vanuatu (20-June) and ARRIVING Bangkok (30-June)

Journey 2: Total Cost $3887.0!!!
- NZ Circuit: LEAVING Auckland (01-Jun) and ARRIVING Wellington (04-Jun)
- NZ Circuit: LEAVING Wellington (06-Jun) and ARRIVING Christchurch AT (08-Jun)
- Cheap As Chips: LEAVING Christchurch (11-Jun) and ARRIVING Brisbane (12-Jun)
- Cheap As Chips: LEAVING Brisbane (13-Jun) and ARRIVING Vanuatu (16-Jun)
- Cheap As Chips: LEAVING Vanuatu (20-Jun) and ARRIVING Bangkok (30-Jun)

Journey 3: Total Cost $9200.0!!!
- NZ Circuit: LEAVING Auckland (01-Jun) and ARRIVING Wellington (04-Jun)
- Bangkok XP: LEAVING Wellington (18-Jun) and ARRIVING Bangkok (28-Jun)

Journey 4: Total Cost $13170.0!!!
- Endeavour: LEAVING Auckland (01-Jun) and ARRIVING Sydney (03-Jun)
- Ozzie: LEAVING Sydney (03-Jun) and ARRIVING Brisbane (04-Jun)
- Bangkok XP: LEAVING Brisbane (09-Jun) and ARRIVING Wellington (11-Jun)
- Bangkok XP: LEAVING Wellington (18-Jun) and ARRIVING Bangkok (28-Jun)

Journey 5: Total Cost $4368.0!!!
- Endeavour: LEAVING Auckland (01-Jun) and ARRIVING Sydney (03-Jun)
- Ozzie: LEAVING Sydney (03-Jun) and ARRIVING Brisbane (04-Jun)
- Cheap As Chips: LEAVING Brisbane (13-Jun) and ARRIVING Vanuatu (16-Jun)
- Cheap As Chips: LEAVING Vanuatu (20-Jun) and ARRIVING Bangkok (30-Jun)

Example: Leaving Brisbane to Antarctica from the 8th June: There is 1 possible journey:

Journey 1: Total Cost $4850.0!!!
- Bangkok XP: LEAVING Brisbane (09-Jun) and ARRIVING Wellington (11-Jun)
- NZ Circuit: LEAVING Wellington (21-Jun) and ARRIVING Christchurch AT (23-Jun)
- Antarctic Explorer: LEAVING Christchurch (25-Jun) and ARRIVING Antarctica AT (28-Jun)

Example: Leaving Auckland to Melbourne from the 10th June: There is 3 possible journeys:

Journey 1: Total Cost $1925.0!!!
- Ozzie: LEAVING Auckland (22-Jun) and ARRIVING Brisbane (24-Jun)
- Ozzie: LEAVING Brisbane (24-Jun) and ARRIVING Sydney (25-Jun)
- Ozzie: LEAVING Sydney (26-Jun) and ARRIVING Melbourne (27-Jun)

Journey 2: Total Cost $990.0!!!
- NZ Victory: LEAVING Auckland (10-Jun) and ARRIVING Melbourne (12-Jun)

Journey 3: Total Cost $1340.0
- NZ Victory: LEAVING Auckland (18-Jun) and ARRIVING Melbourne (20-Jun)

Note: The above journey 2 and 3, although the same boat leave on different days so is valid. Also note in all examples each journey is unique and has no closed paths (not visiting the same port twice) and each cruise ship in the journey departs after the previous trips arrival (or on same day).