**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# **Introduction**

A **Map** is a type of collection that associates a key with a value. The mapping of keys to values can be accomplished using different underlying data structures. In this three-part assignment, you will be working with two such data structures and then using them to implement a **Map**. One is called a **Linked List** and the other is called a **Tree**.

In part II, you will be implementing a Map that uses your Linked List to store its entries. It's important to remember that the LinkedList should handle its own nodes and data. **Your Map should only use the LinkedList by calling its methods.** It should not touch inner variables like head or tail or create node objects itself. That is the LinkedList's job to do internally.

Since the key-value pairs stored in the map could be of any type, we will be using [Generics](https://www.geeksforgeeks.org/generics-in-java/) to implement all our classes. For example, we could have < “Andrew”, 1> as a key-value pair (which is of type <String, Integer>) and we could also have < 432, 34> as a key-value pair (which is of type <Integer, Integer>). Hence, it is important that we provide a way to reuse the same code with different inputs. **Generics** enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

**Note:** You are **NOT** allowed to use any built-in Java data structures for any part of this assignment.

## **What is a Map?**

A map associates **keys** with **values**, treating them as a **key/value pair**. Keys must be unique within the map, so there should be no duplicate keys in a map at any time. Values may or may not be unique. Information is stored in the map by providing a key/value pair, and is looked up by providing the key. There is no concept of indexing in the map and the data's ordering is not guaranteed by the definition of a map. When the same key is given for a new value in the map, it may change the value associated with that key or simply not store the new pair, based on the method that was used.

# **Program and Starter Code**

You are being provided with starter code which is an interface representing a **Map**. You can find it in the course public folder ([public/5P](https://cs.unh.edu/~cs416/public/5P)). It contains the signatures of the methods that you need to implement. It also contains an [inner class](https://www.geeksforgeeks.org/inner-class-java/) to represent an **Entry** in the **Map**. Do not change **Map.java**.

## **LinkedList.java**

**IMPORTANT: You may need to modify your LinkedList methods such as get(Object), contains(Object), and remove(Object) to compare the node's value to the object parameter, and not the other way around.** This will allow your map to look things up by using only the key portion of the data. So if you have a variable node that you are comparing to the parameter o, you should call node.value.equals(o) rather than o.equals(node.value).

## **LinkedMap.java inner class Entry**

You will need to create **LinkedMap.java**. It will declare a class **LinkedMap**, (declared as **public class LinkedMap<K, V> implements Map<K, V>** .) When you begin writing the LinkedMap class, you will first need to define its own inner class **Entry** (declared as **public class Entry<K, V>** **implements Map.Entry<K, V>** .) It will need to implement the following methods:

**public Entry( K key, V value )**

The constructor for Entry. It should initialize any necessary fields.

**public K getKey()**

Returns this entry's key.

**public V getValue()**

Returns this entry's value.

**public String toString()**

Returns a string representation of this Entry. It should follow this format: (key, value)

**public boolean equals( Object o )**

Compares an object o to this entry. If the object is an Entry (you can detect this using o instanceof Entry), first cast o to a variable of type Entry, then return whether its key and value both match this entry's key and value. Otherwise, return whether o equals this entry's key.

## **LinkedMap.java**

Your LinkedMap class should use an instance variable of type LinkedList to store its entries. Note that LinkedMap should only call methods from LinkedList, **LinkedMap should not edit the LinkedList's nodes directly**. You have to implement the following methods for the **LinkedMap** class:

**public LinkedMap( )**

The constructor for LinkedMap. It should initialize any necessary fields.

**public LinkedList<Entry<K,V>> getList()**

Returns the LinkedList being used by this map. Mostly used for testing purposes.

**public V put( K key, V value )**

Associates the given key with the given value in the map. Should not add duplicate items to the map (items with the same key.) If the map previously contained a mapping for the key, the old value is replaced by the specified value.

**public V putIfAbsent( K key, V value )**

If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current value.

**public String toString()**

Returns a string representation of this LinkedMap. It should be in this format:

[(key1, val1), (key2, val2), (key3, val3)]

**public void clear( )**

Removes all of the elements from this map.

**public V get( K key )**

Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

**public boolean containsKey( K key )**

Returns true if this map contains a mapping for the specified key.

**public boolean isEmpty( )**

Returns true if this map contains no elements, false otherwise.

**public V remove( K key )**

Removes the mapping for a key from this map if it is present. Returns the value to which this map previously associated the key, or null if the map contained no mapping for the key.

**public int size( )**

Returns the number of elements in this map.

# Interactive Test Driver

Provides a user interface for creating and manipulating a LinkedList of strings. Includes a command for each of the methods above. **It should exit when a user enters the command x.**

| **Command** | **Method** |
| --- | --- |
| p | put( K key, V value ) |
| P | putIfAbsent( K key, V value ) |
| c | containsKey( K key ) |
| C | clear() |
| g | get( K key ) |
| s | size() |
| e | isEmpty() |
| r | remove( K key ) |

Example transcript:

| p word 100  [(word, 100)]  ------------------  p word 100  [(word, 100)]  ------------------  p another 2  [(word, 100), (another, 2)]  ------------------  P word -1  [(word, 100), (another, 2)]  ------------------  p word -1  [(word, -1), (another, 2)]  ------------------  c word  true  ------------------  c not  false  ------------------  e  false  ------------------  s  2  ------------------  g word  -1  ------------------  g not  null  ------------------  r word  [(another, 2)]  ------------------  C  []  ------------------  e  true  ------------------  s  0  ------------------  x |
| --- |