Sets and Maps

Write a set interface with:

- creating an empty set,
- adding an element to a set,
- checking if a set is non-empty,
- checking if an element is in a set,
- set union,
- set intersection

and a key-value map interface with:

- creating an empty map,
- adding a key-value pair to a map,
- finding a value of a key,
- checking if a map is non-empty,
- checking if a key is in a map,
- union of two maps that throws an exception when a key occurs in both maps but with different values.

Provide an ordered linked list based implementations for both interfaces (i.e. write a class implementing sets and a class implementing maps). Use a common abstract superclass for ordered linked lists. Also, provide the `clone()` method for the implementations. Write as much common code as reasonably possible in the superclass.

For bonus points, do one of the following exercises. It would be better if different people select different exercises.

Exercise 1. For both the set and the map class, implement the iterator pattern using an inner class. (Could it be implemented in the abstract superclass?)

Exercise 2. Write set and map implementations using binary search trees (built out of nodes with links to other nodes), also with a common binary search tree superclass. You do not need to write the linked list implementations if you choose this exercise.

Exercise 3. Write an adapter that takes any map implementation and turns it into a set implementation (actually, turns a map object into a set object).

Exercise 4. Write your favorite (nontrivial) algorithm that uses sets and/or maps; use the set/map interfaces.