*CSE 102*

**LinkedList, HashSet, and HashMap**

**LinkedList:**

* LinkedList is yet another implementation of the List interface (ArrayList and Vector were too).
* The main difference between LinkedList and ArrayList is that the latter is implemented using arrays (a contiguous memory block) while the former is not. Each element of a LinkedList contains references to the next and previous elements in the list (this variant is sometimes called doubly-linked list) and each element may reside in arbitrary locations in memory.
* We prefer ArrayList when we want to keep items in an order and want to access them randomly using their indices. We use LinkedList when we want to keep items in an order and add or delete elements at random positions. Deletion or insertion requires shifting of many elements (comparable to the size of the whole list) when we use ArrayList. LinkedList has the big advantage of flexible structure for these operations and it doesn’t require any shifting.

**HashMap:**

* Conceptually, HashMap is a generalization of ArrayList in the sense that ArrayList of type T can be thought of a map from nonnegative integer values (indices) to values of type T.
* Maps make it possible to use data types other than integers to be used as indices (in the map terminology they are called keys) to access the values.
* We use HashMap when we need to store an association between two kinds of entities and want to access one of them by querying the other.

**HashSet:**

* We use HashSet when we are not interested in any kind of ordering for our collection but we want our elements to be unique.
* The main operation supported by sets is the check of membership, whether an element is a member of the set or not.

**Exercises**:

1. “We have a list consisting of all numbers from 1 to 100. Each time; we remove two numbers randomly from the list, add them up, subtract 2 from the sum and add the result back in the list. We continue this process until there is exactly one number left in the list. What is that last number?”

Suppose that you want to solve this problem by simulating the whole process using your computer. You decided to use Java language. Which of the following would be more appropriate to store the numbers?

1. ArrayList
2. LinkedList
3. HashSet
4. HashMap
5. “Write a function which takes a string representing an English sentence and returns the number of occurrences of each word in the string. You can assume that all the words are separated with a single space and there is no punctuation mark.”

Which of the following is more appropriate to use when you want to solve this question?

1. HashSet
2. LinkedList
3. HashMap
4. ArrayList
5. What is wrong with the following statement?

HashMap<String> map = **new** HashMap<>();

1. HashMaps cannot be created empty.
2. The right-hand side of assignment must be **new** HashMap<String>();
3. HashMaps must be specified with two data types, one for key and the other for value.
4. You have to specify an initial size while creating a HashMap.
5. The following piece of code is supposed to take integers from user and store them in an ArrayList until the user enters 0. Which one should be replaced with question marks?

Scanner sc = **new** Scanner(System.***in***);

ArrayList<Integer> list = **new** ArrayList<>();

**int** num;

**while**(??)

list.add(num);

1. (num = sc.nextInt()) != 0
2. num = (sc.nextInt() != 0)
3. **true**
4. num != 0
5. The following function is supposed to convert an array into a set, to get rid of any repetitions. What should come in place of the question marks?

**static** HashSet<String> setOf(String[] arr){

HashSet<String> set = **new** HashSet<>();

**for**(String s:??) {

set.add(s);

}

**return** set;

}

1. set
2. arr
3. arr[i]
4. set[i]
5. Consider the following function which is supposed to do something similar to what get() method of a HashMap<String, Boolean> object does. What is the main difference between the original method and this replica?

**static** Boolean myMapGet(String s, String[] strArr, Boolean[] boolArr) {

**int** i;

**for**(i=0; i<strArr.length; i++)

**if**(strArr[i].equals(s))

**break**;

// nothing found

**if**(i==strArr.length)

**return** **null**;

**else**

**return** boolArr[i];

}

1. The original get() method does not return null in any case.
2. This method returns an instance of the wrapper class Boolean while the original get() method would return a primitive boolean value.
3. Everything is different. It is not possible to write a method which maps different data types without using a real HashMap instance.
4. This method searches for the key by traversing all keys and returns the corresponding value, while the original get() method uses a completely different technique. There is a significant performance difference between the two.
5. Write a program which solves the problem quoted in Question 1.
6. Write the function quoted in Question 2.
7. Write a function which takes a HashMap<String, Integer> *map1* and a HashMap<Integer, String> *map2* as parameters and returns their composite mapping, which is of type HashMap<String, String>. The composition operation is the same with function composition in mathematics.

**BONUS:** Solve the problem in Question 1 without simulating the process.

**ANSWERS:**

1. B
2. C
3. C
4. A
5. B
6. D
7. **public** **static** **void** main(String[] args) {

LinkedList<Integer> list = **new** LinkedList<>();

Random r = **new** Random();

// initialize

**for**(**int** i=1; i<=100; i++)

list.add(i);

**while**(list.size() > 1) {

// draw first

**int** randX1 = r.nextInt(list.size());

**int** num1 = list.remove(randX1);

// draw second

**int** randX2 = r.nextInt(list.size());

**int** num2 = list.remove(randX2);

// add their sum - 2

list.add(num1+num2-2);

}

System.***out***.println(list);

}

1. **static** HashMap<String, Integer> countWords(String sentence){

HashMap<String, Integer> counts = **new** HashMap<>();

**for**(String s: sentence.split(" ")) {

**if**(counts.containsKey(s))

counts.put(s, counts.get(s) + 1);

**else**

counts.put(s, 1);

}

**return** counts;

}

1. **static** HashMap<String, String> compose(

HashMap<String, Integer> map1,

HashMap<Integer, String> map2)

{

HashMap<String, String> res = **new** HashMap<>();

**for**(String key: map1.keySet()) {

Integer n = map1.get(key);

**if**(map2.containsKey(n))

res.put(key, map2.get(n));

}

**return** res;

}