CS61BL Midterm 2 Review

1 Problem Solving with Abstract Data Types

Write `hasDuplicates` which returns true if the input collection of `elements` has duplicates and false otherwise.

```
public static <E> boolean hasDuplicates(Collection<E> elements) {




}
```

Write `isPermutation` which returns true if the first string, `s1`, is a permutation of the second string, `s2`. The permutations of `cat`, for example, are: `cat, cta, act, atc, tca, and tac`.

```
public static boolean isPermutation(String s1, String s2) {




}
```

2 Generic Question

Consider the `NumericSet` class below.

```
class NumericSet<T extends Number> extends HashSet<T> {
    /** implementation goes here */
}
```

(a) What is the purpose of the `extends` keyword?

(b) What types or classes of objects can you add to a `NumericSet`?

(c) Why is the generic type for `HashSet <T>` and not `<T extends Number>`?

Now, consider the class header for the `BinarySearchTree` we introduced in lab.

```
class BinarySearchTree<T extends Comparable<T>> implements Comparable<T> {
    /** implementation goes here */
}
```

(a) What type of elements does an instance of `BinarySearchTree` hold?

(b) What method do all elements stored in a `BinarySearchTree` instance have in common?

(c) Why not define the generic type as `<?  extends Comparable>`?

3 VoteIterator

Write an iterator that takes in an `Integer` array of vote counts and iterates over the votes. The input array contains the number of votes each selection received. For example, if the input array contained the following:

```
[0, 2, 1, 0, 1, 0]
```

then calls to `next()` would eventually return 1 twice, 2 once, and 4 once. After that, `hasNext()` would return false. Provide code for the `VoteIterator` class below. Make sure your iterator adheres to standard iterator rules.

```
public class VoteIterator implements Iterator<Integer> {



    public VoteIterator(Integer[] votes) {




    }









}
```

4 db61bl

Let's implement a relational database management system! In this system, we manage *tables* consisting of several organizing *columns* and *rows* containing the entries for those columns.

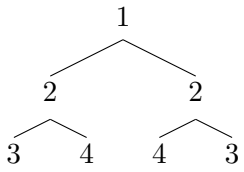
titles	SID	Lastname	Firstname	Major
rows	101	Yao	Alan	Food Systems
	102	Chen	Antares	Literature
	103	Nguyen	Daniel	Parrot Biology
	104	Lee	Maurice	Chemical Engineering
	105	Jian	Lisa	Cosmology
	106	Kim	Sarah	Parrot Biology

Fill out the implementations for the overloaded methods `select`, `export`, and `exportBy` on the next page. You may find it helpful to write the most abstract version of each method (the one that consumes the most arguments) first before writing the other convenience methods.

```
class Table {
    String name;
    String[] titles;
    Set<Row> rows;

    Table(String name, String... titles) {
        this.name = name;
        this.titles = titles;
        this.rows = new HashSet<>();
    }

    class Row {
        String[] data;

        Row(String... data) {
            this.data = data;
        }

        /** Return a new row with the entries specified by COLUMNS. */
        Row getRow(String... columns) {
            String[] newData = new String[columns.length];
            for (int i = 0; i < columns.length; i++) {
                int rowIndex = getColumnIndex(columns[i]);
                newData[i] = data[rowIndex];
            }
            return new Row(newData);
        }
    }

    /** Add multiple ROWSTOADD to this table and return this table. */
    Table addAll(Stream<Row> rowsToAdd) {
        rowsToAdd.forEach(rows::add);
        return this;
    }
```

```
    /** Return a table containing new rows where PRED is true. */
    Table select(Predicate<Row> pred) {
        return new Table(name, titles).
    }

    /** Return a table of rows consisting of COLUMNS from this table. */
    Table select(String... columns) {
        return
    }

    /** Return a table containing new rows from first applying PRED and
     *  then selecting the remaining COLUMNS. */
    Table select(Predicate<Row> pred, String... columns) {
        return new Table(name, columns).

    }

    /** Return a stream of rows applying PRED then SELECTOR to each. */
    Stream<Row> select(Predicate<Row> pred, Function<Row,Row> selector) {
        return rows.stream().

    }

    /** Return a list of strings by applying GETTER to each row. */
    List<String> export(Function<Row,String> getter) {
        return
    }

    /** Return a list of strings from rows applying PRED then GETTER. */
    List<String> export(Predicate<Row> pred,Function<Row,String> getter) {
        return rows.stream().



    }

    /** Return a map of strings to results grouped according to GROUPER
     *  and with values selected by GETTER. */
    Map<String, List<String>> exportBy(
            Function<Row,String> grouper, Function<Row,String> getter) {
        return
    }

    /** Return a map of strings to results where the row satisfies PRED,
     *  grouped according to GROUPER, and values selected by GETTER. */
    Map<String, List<String>> exportBy(Predicate<Row> pred,
            Function<Row,String> grouper, Function<Row,String> getter) {
        return rows.stream().



    }
}
```

5 Trees



Define `isSymmetric` which checks whether the binary tree is a mirror of itself.

```
public class BinaryTree<T> {
    TreeNode root;

    public class TreeNode {
        T item;
        TreeNode left;
        TreeNode right;
    }

    public boolean isSymmetric()









}
```

(a) Insert the following numbers into a 2-3 tree: `[20, 10, 35, 40, 50, 5, 25, 15, 30, 60]`

(b) Draw the corresponding left-leaning red-black tree.

6 Asymptotic Analysis

Provide the tightest asymptotic runtime bounds for the following methods: give a $\Theta(\cdot)$ bound if possible, otherwise give both the $\Omega(\cdot)$ and $O(\cdot)$ bound.

(a) Insertion of N distinct elements at the back of the following data structures.

- `ArrayList`
- `LinkedList`
- `TreeSet`
- `HashSet`

(b) Insertion of N **identical** elements in sequence into the following data structures.

- `ArrayList`
- `LinkedList`
- `TreeSet`
- `HashSet`

(c) Check that a single element is contained in a collection of N distinct elements.

- `ArrayList`
- `LinkedList`
- `TreeSet`
- `HashSet`

(d) Suppose we have a `HashMap` implementation that has an initial capacity of 1, and a load factor of 1. In this hash map implementation, each resizing of the hash map increases the number of buckets by 1. Assume we are using a good hash function, one that (miraculously) always divides our data sets evenly between the buckets. Give a $\Theta(\cdot)$ worst-case bound for the running times of the following operations.

- Inserting N distinct key-value pairs.
- Looking up a single key after the N insertions above.

(e) Now, suppose we have the same `HashMap` implementation, except resizing doubles the capacity, and we use a hash function that maps all keys to 0. Give a $\Theta(\cdot)$ bound for the average runtime of the following operations.

- Inserting N distinct key-value pairs.
- Looking up a single key after the N insertions above.

7 HashBrowns & HashMaps

Describe what would happen in each of the following scenarios.

(a) Define `String.hashCode` to return the length of the string.

(b) Define `String.hashCode` to return a random, uniformly-distributed number each time.

(c) Override the `equals` method without overriding the `hashCode` method.

(d) Override the `hashCode` method without overriding the `equals` method.

(e) Modify the key of an entry inserted into a `HashMap`.

(f) Modify the value of an entry inserted into a `HashMap`.

(g) Instead of a linked list, we use an array, BST, or hash table as the external chain.