**CS-390**
**Fundamental Programming Practices**
**Final Exam Sample**

**Name: _____ ID: _____**

| I | II | | | III | | SCI |
|---|---|---|---|---|---|---|
| | | | | | | |

**Part I. Multiple Choice & True/False Questions.** (2 points each) For multiple choice, circle the best answer; circle *only one* answer in each problem. For True/False, mark it either 'T' or 'F'.

1. Which of the following statements is true?
   a. Use `ArrayList` when a lot of insertions and removals are needed.
   b. There is no need to shift elements when we remove elements from `ArrayList`.
   c. `LinkedList` implements `RandomAccess`.
   d. Resizing is not necessary for a `LinkedList` when a lot of insertions are done.

2. ____(True/False) Suppose you create a class `Key` in which you override `equals` and `hashCode`. Suppose that your way of overriding `hashCode` is the following:

```
hashCode() {
   return 1;
}
```

   If you use instances of `Key` as keys in a `Hashmap`, the `Hashmap` operations of `put`, `get`, `remove` will be no more efficient than the corresponding operations of adding, getting, and removing elements in a linked list.

3. ___(True/False) In-order traversal will visit nodes in a binary search tree in sorted order.

4. ___(True/False) The following code is a full implementation of an `Employee` class and includes an implementation, as an inner class, of the `Comparator` interface. Is the implementation shown consistent with equals?

```
public class Employee  {
   private String name;
   private double salary;
   public Employee(String name, double salary) {
      this.name = name;
      this.salary = salary;
   }
   class NameComparator implements Comparator<Employee> {
      @Override
      public int compare(Employee e1, Employee e2) {
         if(e1.name.equals(e2.name)) return 0;
         else return e1.name.compareTo(e2.name);
      }
   }
   public boolean equals(Object ob) {
      if(ob == null) return false;
      if(!(ob instanceof Employee)) return false;
      Employee e = (Employee)ob;
      Return e.name.equals(name) && e.salary == salary;
   }
}
```

5. The new `forEach` method that was introduced in Java 8 is an example of which of the following (circle the best answer)
    a. A static method in an interface
    b. A default method in an interface
    c. A new implemented method in the Iterator interface
    d. None of the above

6. What is the output when the `main` method of `Test` class is run? (You may safely assume that no compiler errors will occur.)

```
public class Test {
    public static void test() throws Exception{
        try {
             throw new Exception("Exception thrown in test method");
        }
        catch (Exception x){
           System.out.println(x.getMessage() + " and caught by Exception!");
        }
        finally {
           System.out.println("In finally block!");
        }
        System.out.println("In test method!");
    }

    public static void main(String[] args){
        try{
            test();
        }
        catch(Exception x){
            System.out.println("Caught by Exception In main method!");
       }
    }
}
```

**Explain and list the output:**

7. When the `main` method is run in the `Main` class (shown below), which of the following is output to the console? Circle only one answer.

   a. true
      001:data

   b. true
      null

   c. false
      001:data

   d. false
      null

```java
public class Main {
    HashMap<Key, Record> map = new HashMap<>();
    Key defaultKey = new Key("secret");
    public Main() {
        map.put(defaultKey, new Record("001", "data"));
    }
    public static void main(String[] args) {
        Main m = new Main();
        Key k = new Key("secret");
        System.out.println(k.equals(m.defaultKey));
        Record recFound = m.map.get(k);
        System.out.println(recFound);
    }
}
```

```java
public class Key {
    private String key;
    public Key(String k) {
        this.key = k;
    }

    @Override
    public boolean equals(Object ob) {
        if(ob == null) return false;
        if(!(ob instanceof Key)) return false;
        Key theKey = (Key)ob;
        return key.equals(theKey.key);
    }
}

public class Record {
    private String recordId;
    private String data;
    public Record(String id, String data) {
        this.recordId = id;
        this.data = data;
    }
    public String getRecordId() {
        return recordId;
    }
    public String getData() {
        return data;
    }

    @Override
    public String toString() {
        return recordId + ":" + data;
    }
}
```

**Part II. Short Answer**

1. [3 points] Draw the 10-item MyHashtable: keys 1, 4, 3, 8, 2, 9, 11, 19, 20, and 5 using the following hash and hashcode methods and assuming collisions are handled by chaining. value = 2* key.

```
tableSize = 10;
bigNum = hashCode();
index = hash(bigNum);

public int hashCode() {
   int result = 7;
   result += 11 * result + key;
   return result;
}
private int hash(int bigNum) {
   return (int)Math.abs(bigNum % tableSize);
}
```

2. [4 points] Draw the binary search tree obtained from successively adding the following integers to an initially empty BST: 5, 9, 2, 3, 1, 4, 8, 7

**Part III. Programming Questions.**

1. (12 points) Below is a skeleton of a `Stack` implementation based on `Node`. The `MyStack` class has a member node `top`. Your task is to implement the four unimplemented stack methods shown in the code below. If there is no element in stack, throw IllegalArgumentException for the peek and pop methods. Write your code in the space provided, below:

```
public class MyStack {
   private Node top;
   private class Node{
      Integer data;
      Node next;
   }
   public boolean isEmpty() {



   }
   public void push(Integer val) {









   }

   public Integer peek() {









   }

   public Integer pop() {









   }
}
```

2. (12 points) Fully implement the methods in the `SearchForString` class, shown below. The class `SearchForString` has one instance variable `String[] arr`, one constructor with signature

```
SearchForString (String[] arr)
```

and one instance method

```
public boolean search(String s)
```

The constructor should set its value in the instance variable of the class. The method `search` should be a recursive implementation of a search for the input argument `s` in the array `arr`; if `s` is found, the method should return `true`; `false` otherwise.

The method must implement the following recursive strategy:

> Compare `s` to `arr[len-1]` (where `len` is the length of `arr`). If they are equal, return `true`. Otherwise, (recursively) search for `s` in the rest of the array.

You may safely assume that `arr` contains only non-null `Strings` and that the argument `s` passed in to `search` is never null. You *must not* assume that the `Strings` in `arr` are in sorted order.

To complete the problem, complete the work in the class `SearchForString` that has already been partially coded. A `private` instance method `recurSearch`, having two arguments (`s` and an integer argument `upperIndex`) has been included in `SearchForString`; you must make use of this method to do the actual recursion.

```
//write your code on the next page
```

```java
public class SearchForString {
   private String[] arr;
   public SearchForString(String[] arr) {
      this.arr = arr;
   }

   public boolean search(String s){









   }
   private boolean recurSearch(String s, int upperIndex){

   }
}
```