CS-390
Fundamental Programming Practices
Final Exam

Name: ____________________ ID: _________________

I (16)	II (16)			III (24)		SCI (3)

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. Suppose a list mylist contains objects that do not have a natural ordering. (Note: Integers and Strings have a natural ordering but Persons and Employees do not.) Assume the following line of code compiles and runs without error:
   ```
   Collections.sort(mylist)
   ```
   Which of the following statements is *not necessarily* true?
   a. The type of `mylist` is a subtype of `List`
   b. The type of `mylist` is a subtype of `Iterable`
   c. The type of `mylist` is a subtype of `Comparable`
   d. The type of each element of `mylist` is a subtype of `Comparable`

3. Suppose you create your own kind of `List` class, based on an array, in which you plan to store `String`s in sorted order. Which Java interface(s) *must* you implement in order to be sure that the method call
   ```
   Collections.binarySearch(<your list>, <your string>)
   ```
   will run properly and use the fast binary search algorithm on your `List` in searching for your test `String`? Circle the best answer.

   a. The `List` interface is enough
   b. The `Iterable` interface is enough
   c. Both the `List` interface and the `RandomAccess` interface
   d. The `RandomAccess` interface is enough

4. ____(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.

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

6. ___(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;
   }
}
```

7. 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

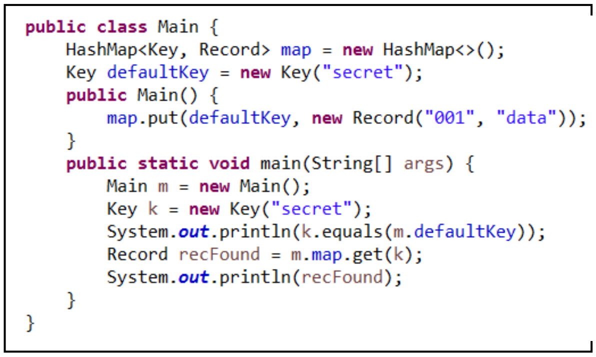
8. When the `main` method is run in the `Main` class (shown below), which of the following is output to the console? Circle only one answer.

a. true
001:data

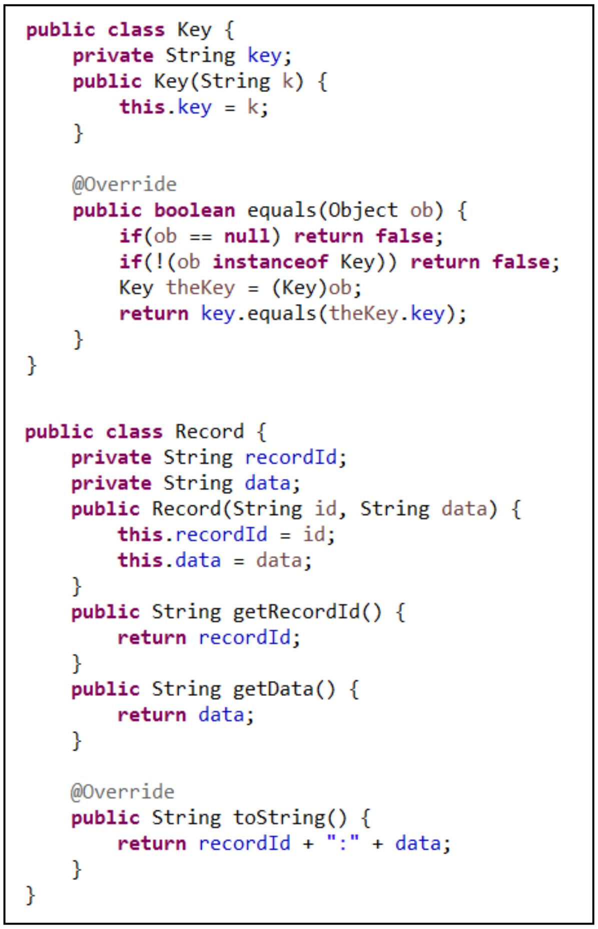
b. true
null

c. false
001:data

d. false
null



```
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);
    }
}
```



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

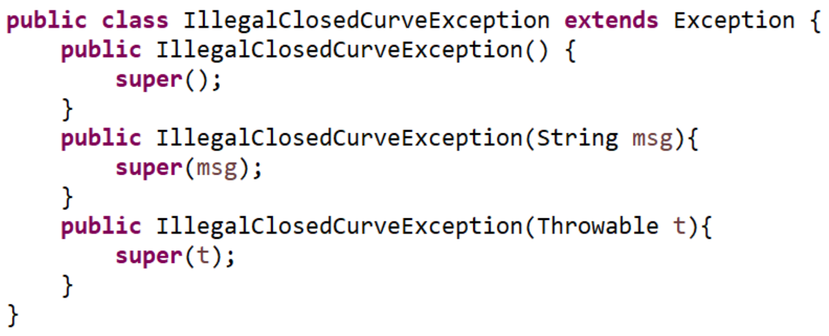
```
class Test {
   public static void test() throws Exception {
        try {
             throw new Exception("Exception thrown");
        }
        catch (Exception x){
            System.out.println(x.getMessage());
        }
        finally {
            System.out.println("In finally block!");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

   public static void main(String[] args){
        try{
            test();
        }
        catch(Exception x){
            System.out.println(x.getMessage());
      }
    }
}
```

2. [5 points] Many data structures are implemented using the composition pattern, relying on some kind of background data structure to perform its operations. Below, several such data structures are listed (a – e). Match the data structure to the type of structure most often used as its background data structure (choose from 1 – 5).

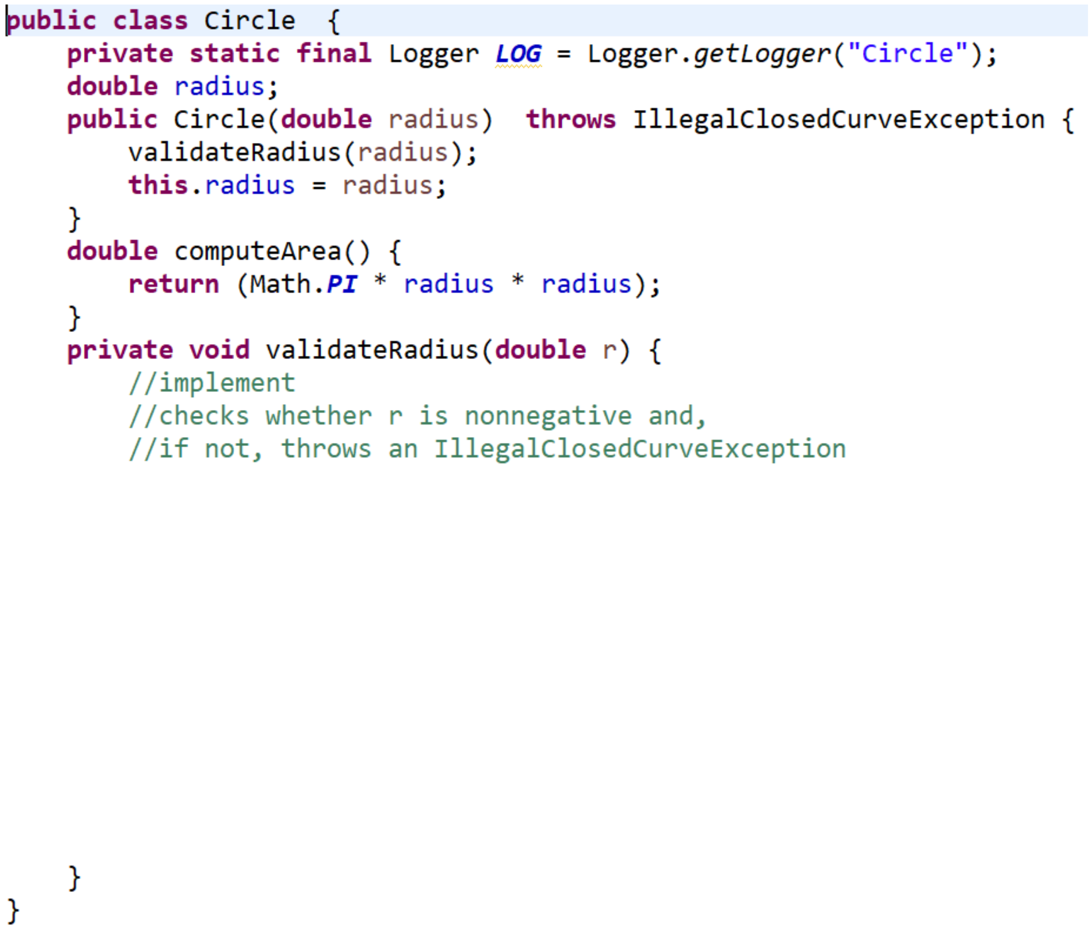
   Note. It is possible to use data structures in the right column more than once.

__ a. Binary Search Tree	1. Node
__b. HashMap	2. Entry
__ c. TreeSet (from Java library)	3. Binary Search Tree
__ d. LinkedList	4. Linked List
__ e. HashSet	5. HashMap

3. [4 points] Below is code for a `Circle` class. The constructor accepts input for the size of the circle's radius and intends to validate that the input value for the radius is non-negative. To validate the input, the method `validateRadius` is called. Write the code for the `validateRadius` method. This method should throw an `IllegalClosedCurveException` if the input value of the radius is a negative number. *Hint.* You may need to modify the declaration of the method.



```
public class IllegalClosedCurveException extends Exception {
    public IllegalClosedCurveException() {
        super();
    }
    public IllegalClosedCurveException(String msg){
        super(msg);
    }
    public IllegalClosedCurveException(Throwable t){
        super(t);
    }
}
```



```
public class Circle  {
    private static final Logger LOG = Logger.getLogger("Circle");
    double radius;
    public Circle(double radius)  throws IllegalClosedCurveException {
        validateRadius(radius);
        this.radius = radius;
    }
    double computeArea() {
        return (Math.PI * radius * radius);
    }
    private void validateRadius(double r) {
        //implement
        //checks whether r is nonnegative and,
        //if not, throws an IllegalClosedCurveException













    }
}
```

4. [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 `Nodes`. The `NodeStack` class has a member inner class `Node` that has already been implemented, and has an instance variable `topNode`. Your task is to implement the three unimplemented stack methods shown in the code below. To implement `pop`, you must replace `topNode` with the next `Node` in the stack, and return the value contained in the original `topNode`. For `peek`, you must return the value stored in `topNode`, but you will not remove it. And for `push`, you will create a new `Node` and set it as the new `topNode`. All changes made by `push`, `pop`, and `peek` must ensure that links from `Node` to `Node` have been defined properly. Write your code in the space provided, below:

```
public class NodeStack {
    private Node topNode = null;
    public void push(String val) {



    }

    public String peek() {



    }

    public String pop() {



    }
```

```
   class Node {
      private String data;
      private Node next;
      Node(String data, Node next) {
         this.data = data;
         this.next = next;
      }
   }
}
```

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
```

```
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){



   }
}
```

Part IV. SCI (3 points)

Describe a parallel between principles of SCI and principles of computer science that have been discussed in the course. Richer content will be awarded more credit.