I. Fill in the Blank (30 pts)

1. A **self-referential** structure is a class definition which contains an instance variable whose type is the class definition.

2. A **Set** is used to hold a unique collection of objects.

3. We use the keyword **transient** to indicate that the value of an instance variable won’t be included when the instance of the class is serialized.

4. Java implements the **termination** model of exception handling.

5. **Throwable** is at the top of the hierarchy of classes that deal with exceptions that occur during execution.

6. We know that we are done reading from a BufferedReader when the method `readLine()` returns null.

7. The keyword **try** precedes a block of code that might cause an exception to occur so that we can catch and handle that exception.

8. The **DataOutputStream** class is used to write primitive types to an underlying stream in a platform independent way.

9. We use the method **retainAll()** to determine the intersection of two Collections.

10. We use the method **seek** to advance the file pointer in a RandomAccessFile.

11. A **ArithmeticException** occurs if we have an integer division with the denominator 0.

12. A queue is known as a **FIFO** structure because the last thing placed in the queue is the last thing taken out.

13. A **NullPointerException** occurs if we try to reference an instance method on an object reference which does not refer to an actual object.
14. **Serialization** is the process of converting an object into a form that can be written to a stream.

15. If a primitive numeric type is signed, the first bit in its bit representation is set to 1.

16. The variable `serialVersionUID` is created for a Serializable class if it doesn’t already contain one.

17. The **InputStreamReader** class is used as a bridge between byte and character streams so that we can read characters from the stream.

18. In order to write an instance of a class to a file, the class definition must implement the **Serializable** interface.

19. The major difference between the Vector and ArrayList classes is that the methods in the Vector class are **synchronized**.

20. The keyword **assert** can be used as a debugging mechanism.

21. **Error** represents exceptions that occur during execution that a programmer is not expected to recover from.

22. Java uses **UNICODE** to encode characters.

23. The keyword **throw** is used to cause an exception to occur.

24. We use the keyword **throws** on a method header to indicate that a method might cause an exception to occur.

25. A **TreeSet** contains a unique collection of objects sorted according to their natural order or based on a Comparator.

26. A stack is known as a **LIFO** structure because the first thing placed in the stack is the last thing taken out.

27. The write methods in the **PrintStream** class for writing byte streams do not cause IOExceptions.

28. The **FileOutputStream** is the class we use to establish a basic byte connection to send output to a file.

29. 8 bits are used to store a byte.

30. We normally wrap a FileReader in a **BufferedReader** for efficiency.
II. Matching (10 pts)

31. unchecked exception A 
A. Error, RuntimeException, and their subclasses.

32. stack trace Q 
B. HashSet<String> set = new HashSet<String>();

33. push J 
C. The operation on a stack that removes an element from the stack.

34. get(int) R 
D. TreeSet<String> set = new HashSet<String>();

35. finally W 
E. The operation on a stack that examines the first element in the stack.

36. checked exceptions G 
F. Set<String> set = new HashSet<String>();

37. “rw” M 
G. Throwable, Exception, and subclasses of Exception that are not subclasses of RuntimeException.

38. Programming to an interface F 
H. Error, Throwable, ClassCastException, and IOException and their subclasses. and not an interpretation

39. NumberFormatException T 
I. The mode used to open a RandomAccessFile for reading.

40. ClassCastException L 
J. The operation on a stack that places an element on the stack.

K. Occurs if we cast a reference type to Object.

L. Occurs when we attempt to cast a reference to a subclass reference, but the object is not an instance of that subclass.

M. The mode used to open a RandomAccessFile for both reading and writing.

N. A block that must always be included with a try block.

O. A method for retrieving an element from a certain position in a set.

P. Error and Exception.

Q. Contains a list of the methods called up to a certain point in a program.

R. The method used to retrieve an element from a certain position in a list.

S. An abbreviation for the MTV Show “The Real World”.

T. Occurs if we try to convert a String to a primitive type when that String doesn’t contain that primitive type.

U. A block of code associated with a try block that will only be executed if an exception occurs during the try block.

V. Occurs if we try to divide by an integer 0.

W. A block that is associated with a try block that will execute regardless of whether or not an exception occurs in the try block.

X. Occurs at compile-time if we cast a reference to a final class to an interface that class doesn’t implement.

Y. A list of all methods in a class definition that haven’t been called.
III. **Short Answer** (20 pts)

41. (2 pts) If a line of code within a block executes, what can we guarantee has not happened when previous lines in the block were executed?

   **We know that no exception was thrown.**

42. (4 pts) If we inherit a method from a superclass, what is the restriction on the checked exceptions we can list on the method in the subclass? What is the restriction on the unchecked exceptions that can be listed in the subclass.

   **We can only include the checked exceptions or subclasses of the checked exceptions listed in the superclass. We can add any unchecked exceptions we want.**

43. (2 pts) Are the byte and char primitive types signed or unsigned? What is the implication?

   **byte is signed, and char is unsigned. If a numeric type is signed, it can represent both positive and negative values.**

44. (2 pts) What is the difference between writing character data and binary data to a stream?

   **characters must be encoded before they are written and decoded after they are read. bytes are written and read directly.**
45. (4 pts) Suppose we have an array with the contents 15 45 20 17 18 5 10 17 14 5. If we use the first element as the pivot element, what are the contents of the array after one pass of QuickSort?

5 5 14 10 15 18 17 17 20 45

46. (2 pts) Give an example of using the for each construct to iterate through the elements of a Set named set that contains Strings and print each one.

```java
for (String string : set)
    System.out.println(string);
```

47. (2 pts) What method do we use to find the union of two Collections?

`addAll`

48. (2 pts) What are the first and last nodes in a list called?

`head` and `tail`
IV. Discussion (10 pts)

49. (4 pts) What are the three criteria of a well-defined recursive method? Give an example of a well-defined recursive method and point out how the criteria are satisfied.

1. The method must call itself either directly or indirectly.

2. There must be an exit criterion.

3. Each time the method is called, it must work on a “smaller” problem to work toward the exit criterion.

   public int factorial(int n) {
       if (n == 0)
           return(1); -> the exit criterion.
       else
           return(n * factorial(n-1)); -> the method calls itself and works on a “smaller” problem each time it is called.
   }

50. (3 pts) What are the key points in the hashCode() contract? What is the implication of not honoring the hashCode() contract?

1. During a single execution of a program, the hashCode() method should return consistent values on an object.

2. If the equals method returns true between two objects, then the objects must have the same hashCode() value.

3. If the equals method returns false between two objects, it is not necessary that the objects have different hashCode() values.

   If a class definition does not honor the hashCode() contract, then a Set is not guaranteed to be able to determine that two objects are the same.

51. (3 pts) What is important about the relationship between the way the add and remove methods work for a structure that implements a queue?

   We must add and remove from the same end.
V. Problem Solving and Coding (30 pts)

52. (5 pts) Point out the three major errors in the following code.

```java
public class Question52 extends Question1 {
    public void print(int number) throws java.io.IOException {
        try {
            System.out.println(number);
            double d = 1/0.0;
            System.out.println(number);
        } catch (ArithmeticException ae) {
            System.out.println(ae);
        } catch (NullPointerException npe) {
            System.out.println(npe);
        } catch (ClassCastException cce) {
            System.out.println(cce);
        } catch (Exception e) {
            System.out.println(e);
        } catch (java.io.IOException ie) {
            System.out.println(ie);
        }
    }
}

class Question1 {
    public void print(int number) throws InterruptedException, ClassCastException {
    }
}
```

1. You can’t list a catch block for a superclass, Exception, before a catch block for one of its subclasses, java.io.Exception.

2. Since java.io.IOException is a checked exception and is not listed on the throws clause of the method in the superclass that is overridden, then we can’t list it on the method in the subclass.

3. Since java.io.IOException is a checked exception, and there was no line in the try block that could potentially cause an IOException to occur, we can’t have a catch block for java.io.IOException.
53. (5 pts) Suppose we have the following class definition. What is the output of this program? Does this agree with what you would intuitively think it should print. What are two reasons for the discrepancy?

```java
import java.util.*;
public class Question53 {
    private int num;
    public Question53(int num) {
        this.num = num;
    }
    public boolean equals(Question53 question53) {
        return(num == question53.num);
    }
    public int hashCode() {
        return(1);
    }
    public static void main(String[] args) {
        Question53 question531 = new Question53(1);
        Question53 question532 = new Question53(1);
        Set<Question53> set = new HashSet<Question53>();
        set.add(question531);
        set.add(question532);
        System.out.println(set.size());
    }
}
```

The program will print 2.

**We would expect the program to print 1, since we have two Question53 objects which should be equal. However, because the parameter in the equals method is of type Question53 and not Object, we are not overriding the equals method. Instead we are overloading it.**

**Also the program does not correctly override the hashCode() method because the c is lowercase.**
54. (5 pts) What will the following class definition print? Explain the output.

```java
public class Question54 {
    public class LinkedList {
        private Node head;
        public class Node {
            Object datum;
            Node next;
            public Node(Object datum, Node next) {
                this.datum = datum;
                this.next = next;
            }
        }
        public LinkedList() {
            head = null;
        }
        public void add(Object datum) {
            Node current = head;
            while (current.next != null)
                current = current.next;
            current.next = new Node(datum,null);
        }
        public void traverse(Node head) {
            if (head != null) {
                System.out.println(head.datum);
                traverse(head.next);
            }
        }
        public void traverse() {
            traverse(head);
        }
    }
    public static void main(String[] args) {
        LinkedList list = new Question54().new LinkedList();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        list.traverse();
    }
}
```

The program will break with a NullPointerException. In the add method, we don’t check to see if the head is null before we try to access the value of next associated with it.
Choose two of the following problems. If you work more than two correctly, then it will count as extra credit.

55. (7.5 pts) Show how you would implement the enqueue and dequeue methods of a queue by using a linked list?

```java
public class Question55 {
    public class Queue {
        LinkedList list;
        public class LinkedList {
            private Node head;
            public class Node {
                Object datum;
                Node next;
                public Node(Object datum, Node next) {
                    this.datum = datum;
                    this.next = next;
                }
            }
            public LinkedList() {
                head = null;
            }
            public void add(Object datum) {
                if (head == null)
                    head = new Node(datum, head);
                else {
                    Node current = head;
                    while (current.next != null)
                        current = current.next;
                    current.next = new Node(datum, null);
                }
            }
            public Object remove() {
                if (head == null)
                    return(null);
                else {
                    Object datum = head.datum;
                    head = head.next;
                    return(datum);
                }
            }
            public Queue() {
                list = new LinkedList();
            }
            public void enqueue(Object datum) {
                list.add(datum);
            }
            public Object dequeue() {
                return(list.remove());
            }
        }
    }
}
```
56. (7.5 pts) Write a class definition in which 20 random digits less than 50 and 20 random lowercase characters are generated. You should add these to a collection where no digit or character is repeated and the contents are arranged so that the characters arranged in ascending order appear first and the digits sorted in descending order appear after the characters.

```java
import java.util.*;
public class Question56 {
    public static void main(String[] args) {
        Set<Character> set1 = new TreeSet<Character>();
        while (set1.size() < 20)
            set1.add((char) (97 + (int) (26*Math.random())));
        Comparator<Integer> comparator = new Comparator<Integer>() {
            public int compare(Integer integer1, Integer integer2) {
                if (integer1 > integer2)
                    return(-1);
                else if (integer1 < integer2)
                    return(1);
                else
                    return(0);
            }
        };
        Set<Integer> set2 = new TreeSet<Integer>(comparator);
        while (set2.size() < 20)
            set2.add((int) (50*Math.random()));
        Set<Object> set = new LinkedHashSet<Object>();
        set.addAll(set1);
        set.addAll(set2);
    }
}
```
import java.io.*;

public class Question57 implements Serializable {
    private String name;
    private double amountOfChickenEaten;

    public Question57(String name, double amountOfChickenEaten) {
        this.name = name;
        this.amountOfChickenEaten = amountOfChickenEaten;
    }

    public void store(String fileName) {
        try {
            FileOutputStream file = new FileOutputStream(fileName);
            ObjectOutputStream output = new ObjectOutputStream(file);
            output.writeObject(this);
            output.close();
        } catch (IOException ie) {
            System.out.println(ie);
        }
    }

    public static Question57 get(String fileName) {
        Question57 question57 = null;
        try {
            FileInputStream file = new FileInputStream(fileName);
            ObjectInputStream input = new ObjectInputStream(file);
            question57 = (Question57)input.readObject();
            input.close();
        } catch (ClassNotFoundException cnfe) {
            System.out.println(cnfe);
        } catch (IOException ie) {
            System.out.println(ie);
        }
        return(question57);
    }
}
import java.io.*;

public class Question58 {
    private String name;
    public Question58(String name) {
        this.name = name;
    }

    public void write(RandomAccessFile file) throws IOException {
        file.seek(0);
        if (name == null)
            file.writeInt(-1);
        else {
            file.writeInt(name.length());
            file.writeChars(name);
        }
    }

    public static Question58 get(RandomAccessFile file) throws IOException {
        Question58 question58 = null;
        try {
            file.seek(0);
            int lengthOfString = file.readInt();
            if (lengthOfString == -1)
                question58 = new Question58(null);
            else {
                String string = "";
                for (int counter=0;counter<lengthOfString;counter++)
                    string += file.readChar();
                question58 = new Question58(string);
            }
        } catch (IOException ie) {
            System.out.println(ie);
        }
        return(question58);
    }
}