Question #1

a) What is the output of the following code? Write your answer in the box.

```java
int[] myArray = {1, 2, 3, 4, 5};
System.out.println(myArray[4]);
myArray = new int[4];
System.out.println(myArray[3]);
```

b) Fill in the blanks below to indicate what is printed by running the main method of Mystery.java shown below. There are no compile-time or run-time errors in this program.

```java
public class Mystery {
    public static void mystery1(boolean [] bArray){
        boolean b;
        for (int i = 0; i< bArray.length; i++){
            b = bArray[i];
            b = !b;
        }
        Mystery.mystery2(bArray); // 1
        bArray[2] = false;
        Mystery.mystery2(bArray); // 2
        bArray = new boolean[4];
        Mystery.mystery2(bArray); // 3
    }
    public static void mystery2(boolean [] bArray){
        int i = bArray.length - 1;
        while (i > 0){
            System.out.print(bArray[i] + " ");
            i--;
        }
        System.out.println();
    }
    public static void main(String [] args){
        boolean [] bArray = {true, true, true, true, false};
        Mystery.mystery1(bArray);
        Mystery.mystery2(bArray); // 4
    }
}
```

<table>
<thead>
<tr>
<th>Write what is printed</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>// 1</td>
<td>false true true true</td>
<td>false true false true</td>
<td>false false false</td>
<td>false true false true</td>
</tr>
</tbody>
</table>
Question #2

public class ExceptionalStuff {
    public static void crazy(int i) {
        if (i == 0) {
            System.out.println("1");
            throw new NullPointerException();
            System.out.println("2");
        }
        try {
            System.out.println("3");
            throw new ExceptionA();
            System.out.println("4");
        } catch (Exception e) {
            System.out.println("5");
            throw new ExceptionB();
            System.out.println("6");
        } finally {
            System.out.println("7");
        }
    }
}

a) ExceptionA, and ExceptionB all extend Exception. For the code above to compile, what must be added to the blank above? (Circle 0 or more of the words below)

- throws
- ExceptionA
- ExceptionB
- NullPointerException

b) What is printed by ExceptionalStuff.crazy(0);? (You do not need to print anything for exceptions.)

1 3 5 7

- All ExceptionAs that are thrown are caught so it is not necessary to add to the blank
- This is thrown by the method and must be caught
- You don’t need to declare null pointer exceptions as thrown because they are unchecked.

6
Question #3

For each example of code, respond whether or not it will compile. If it compiles, please respond whether or not it will run without errors. If it runs without errors and has a return value, please write the return value.

```
// Code – Each group of lines is independent

(new Y()).method(“hi”);  // Compiles? NO
This method is private for Y

(new Z()).method();  // Compiles? YES
Runs without errors? YES
Return value? 0

((Z) (new Y())).method(“yo”);  // Compiles? YES
You promise it is a Z, but it isn’t so it has a runtime error

X x1 = new Z();
Y y1 = (Z) x1;
Z is a subclass of Y so you can cast to a Z and set equal to a Y reference.

X[] xarr = {new Y(), new X()};  // Compiles? NO
You can’t make a new X() because it is an interface.

Y[] yarr = {new Y(), new Z()};  // Compiles? YES
Works – Z is a subclass of Y so you can put it in a Y array

((Y) (new Z())).method(“hey”);  // Compiles? NO
Y’s do not have a public method that takes a string.

X x2 = new Z();
Z z2 = (Y) x2;
z2.method();  // Compiles? NO
you can’t cast to a Y and then set it to a Z. You must cast to a Z.

X x3 = new Z();
x3.method(“hello”);  // Compiles? NO
X doesn’t have a method that takes in a String.
```
Question #4

Fill in the blanks below with legal Java to produce the output indicated in each comment. If it is impossible to write “IMPOSSIBLE” in the blank. **You may not create any additional objects!**

```java
class Parent {
    public void feed(Parent p) {
        System.out.println("Parent feed Parent");
    }
    public void feed(Child c) {
        System.out.println("Parent feed Child");
    }
}
class Child extends Parent {
    public void feed(Parent p) {
        System.out.println("Child feed Parent");
    }
    public void feed(Child c) {
        System.out.println("Child feed Child");
    }
    public static void main(String[] args) {
        Parent p = new Child();
        // Child feed Child
        p.feed((Child) p);
        // Child feed Parent
        p.feed(p);
        // Parent feed Child
        Impossible
        // Parent feed Parent
        Impossible
        p = new Parent();
        // Child feed Child
        Impossible
        // Child feed Parent
        Impossible
        // Parent feed Child
        Impossible
        // Parent feed Parent
        p.feed(p);
    }
}
```
Question #5 (continued on next page)

Below is a modification of code from the Account class. Read the syntactically valid code provided and debug the method removePoorParents(). This method should remove any parent from the chain of parents that has a balance less than 1,000. An Account that has their parent Account removed should still be able to access the parent of their former parent Account (Assuming that parent Account has a balance of 1000 or greater.)

a) Fill in the main method below with code to demonstrate the logical error in removePoorParents(). Also fill in the blanks to explain the error.

```java
public class Account {
    private Account myParent;
    private int myBalance;
    public Account(int balance, Account parent) {
        this.myBalance = balance;
        this.myParent = parent;
    }
    public void removePoorParents() {
        if (this.myParent != null) {
            if (this.myParent.myBalance < 1000) {
                this.myParent = this.myParent.myParent;
                if(this.myParent == null){
                    return;
                }
            }
            this.myParent.removePoorParents();
        }
    }
    public static void main(String[] args) {
        Account a1 = new Account(10, null);
        Account a2  = new Account(10, a1);
        Account a3 = new Account (10, a2);
        a3.removePoorParents();
        /* At this point ___________ is ____________________
        * but it should be ____________________
        */
    }
}
```

It keeps your parent even if your parent is poor.
Question #5 (continued from previous page)

b) Modify the `removePoorParents()` method below to fix the bug you demonstrated in part a).

```java
public void removePoorParents() {

    if (this.myParent != null) {

        if (this.myParent.myBalance < 1000) {

            this.myParent = this.myParent.myParent;
            this.removePoorParents();
            if (this.myParent == null) {

                return;

            }

        }

    }

    this.myParent.removePoorParents();

}
```

This is one of about 5 solutions that we saw or came up with. There are probably many more.