Birthday: Month: _____ Day: _____

Question #1

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

When you create the new array the values are set to 0

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

5

0

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.

```
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
   }
}
```

B is a local variable so setting it doesn't affect the array

Set to a new array with default values false and has length of 4 – not 5

Doesn't print the last element. This wasn't intended to be tricky but rather to try to test your understanding of loops.

Printing Backwards.

Shouldn't be changed by modifying the local variable bArray in the method mystery1. In the main we still point to the old array

	Write what is printed
// 1	false true true true
// 2	false true false true
// 3	false false false
// 4	false true false true

6

Birthday: Month: _____ Day: _____

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");
        }
    }
```

Once the nullpointer exception is thrown – no other code executes. Because it is not inside of a try. So the 2 is never printed.

4 and 6 are never printed because they come after an exception that is thrown. The finally is always executed last.

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

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.

c) What is printed by `ExceptionalStuff.crazy(1);` ? (You do not need to print anything for exceptions.)

3

5

7

_____

6

Birthday: Month: _____ Day: _____

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.

```
interface X {
        int method();
}
class Y implements X {
        int method(){
                return 0;
        }
        private double method(String arg){
                return 12.20;
        }
}
class Z extends Y{
        double method(String arg){
                return 3.14;
        }
}
public class testXYZ{
        public static void main(String[] args) {
```

// Code – Each group of lines is independent	Compiles?	Runs without errors?	Return value?
`(new Y()).method("hi");` This method is private for Y	**NO**		
`(new Z()).method();` Works	**YES**	**YES**	**0**
`((Z) (new Y())).method("yo");` You promise it is a Z, but it isn't so it has a runtime error	**YES**	**NO**	

`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.
	YES	**YES**	

`X[] xarr = {new Y(), new X()};` You can't make a new X() because it is an interface.	**NO**		
`Y[] yarr = {new Y(), new Z()};` Works –Z is a subclass of Y so you can put it in a Y array	**YES**	**YES**	
`((Y) (new Z())).method("hey");` Y's do not have a public method that takes a string.	**NO**		

`X x2 = new Z();`
`Z z2 = (Y) x2;`
`z2.method();`
you can't cast to a Y and then set it to a z. You must cast to a Z.
	NO		

`X x3 = new Z();`
`x3.method("hello");`
X doesn't have a method that takes in a String.
	NO		

```
    }
}
```

9

Birthday: Month: _____ Day: _____

Question #4

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

```
public class Parent {
	public void feed(Parent p){
		System.out.println("Parent feed Parent");
	}
	public void feed(Child c){
		System.out.println("Parent feed Child");
	}
}
public 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();
```

p.feed((Child) p)	// Child feed Child
p.feed(p)	// Child feed Parent
Impossible	// Parent feed Child
Impossible	// Parent feed Parent

```
		p = new Parent();
```

Impossible	// Child feed Child
Impossible	// Child feed Parent
Impossible	// Parent feed Child
p.feed(p)	// Parent feed Parent

```
	}
}
```

Birthday: Month: _____ Day: _____

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.

```
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) {
```

It keeps your parent even if your parent is poor.

```
      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 _____

 */
   }
}
```

Birthday: Month: _____ Day: _____

Question #5 (continued from previous page)

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

```
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.

4