Question 4 (10 marks total)

Consider the `Bit` enum and `BigBinary` class from Assignment 4 where the bits of the binary number are stored in a list:

```
public enum Bit {
    ZERO, ONE;
    // rest of enum not shown
}

public class BigBinary {
    private List<Bit> bits;

    public BigBinary(List<Bit> bits) {
        // NOT SHOWN
    }

    public int numberOfBits() {
        return this.bits.size();
    }
}
```

Part A (2 marks)

Consider only the implementation shown above (and not how the class was supposed to be implemented on the assignment). Which of the following relationships exist between `Bit` and `BigBinary`?

- dependency
- association
- aggregation
- composition
- inheritance

Part B (2 marks)

Suppose that you were using the `BigBinary` class, but you could not see the source code or its documentation. What (few lines of) code could you write to determine if a `BigBinary` object has exclusive access (owns) its list of bits?

Continued on next page.

Part C (2 marks)

Suppose that `BigBinary` has the following method:

```
public Bit getBit(int index) {
    // check if index is valid here; not shown
    return this.bits.get(index);
}
```

Does `getBit` have a privacy leak? Briefly explain why or why not.

Part D (4 marks)

Suppose that two `BigInteger` objects are equal if and only if their patterns of bits are the equal (both numbers have the same number of bits and the corresponding bits of both numbers are equal). A programmer overrides `equals` as follows:

```
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return false;
    }
    if (this.getClass() != obj.getClass()) {
        return false;
    }
    BigBinary other = (BigBinary) obj;
    for (int i = 0; i < this.bits.size(); i++) {
        if (this.bits.get(i) != other.bits.get(i)) {
            return false;
        }
    }
    return true;
}
```

There are some errors in the implementation of `equals`. For each of the following four cases, provide an example of `BigBinary` numbers `a` and `b` such that:

1. `a.equals(b)` incorrectly returns `false`
2. `a.equals(b)` throws an exception
3. `a.equals(b)` throws an exception of a different type than your answer to the previous case #2
4. `a.equals(b)` incorrectly returns `true`