**Q1 (Person and HashMaps)**
Part 1:
- ID should be final for HashMap to be able to use it
- Missing override of ‘equals’ and ‘hashCode’ methods - necessary for inserting Person into HashMap

Part 2:
- Missing main
- Initializing HashMap as HashMap<String,Integer> or HashMap<Person, int>
- Some people initialized HashMap inside person which was not the intention of the question.

**Q2 (Decorator and Singleton)**
Part 1:
- Decorator doesn’t save an inner Bird instance and\or doesn’t delegate actions to it
- No override of ‘fly’ method - implementation of other methods instead
- Using super.fly() instead of delegation

Part 2:
- Missing keywords in instance initialization (private static final)

**Q3 (Serialization and Reflection)**
Part 1:

Part 2:
- A lot of people thought reflection solves circular referencing

**Q4 (Regex)**

Part 3:
- A lot of people thought it would capture ‘abe’ and didn’t account for (Or gave a wrong explanation for) ‘?’ character.

**Q5 (Beastmaster)**
Part 1:
- Holding an Animal object instead of a generic T type
- Not holding an inner Animal at all

Part 2:

Part 3:
- Using <T extends Monkey> when it is already used (Extends Animal)
- Using a generic that was not used in the class
- Initializing an array of <? Extends Monkey> instead of beastmasters
- Initialized just one instance of BeastMaster
- Initialized a list of BeastMasters of type Monkey instead of <? Extends Monkey>

**Q6 (Erasure)**
Part 1:
- Missing explanation about what happens after deletion

Part 2:

- Examples with upcasting or downcasting unrelated to generics

**Part 2 Q1:**

1) מחלקה פרטית Node לא לעשות את

2) פרטי ולהשתמש בו כערך החזרה ממתודה פומבית Node לעשות את

3) גנרי MyLinkedList לא לעשות את

4) גנרי Comparable לשכוח שגם

**Part 2 Q2:**

**Part A:**

- Not mentioning a functional interface has exactly one abstract method.

**Part B:**

- Lambda / Comparator returned a double and not an int
- Casting a double to an int in the comparator (leading to edge cases being wrong)
- Calling compareTo on a double, need to use Double.compareTo() instead.

**Part C:**

- Using a hashMap with the Computable objects as a key. We don't know if Computable objects implement hashCode and equals so they can't be used as keys

**Part D:**

- Adding access modifiers to local classes

**Part E:**

- Mentioning a programming principle which wasn't actually used in the solution.
- If you only used functional programming and not OOP design principles there was no need to mention any design principles.