CPT204 Hyflex 2021
Final Exam

Overview

This document contains questions for Final Exam. However, the final exam will be given as a Quiz in Learning Mall.

There are five questions with total marks 100 points. All questions are to be answered and submitted within 2 hours.

Question 1. Short Answer Questions (20 points)

Fill in the box with the answer.

1. Consider the following code:
   ```
   String name = null;
   System.out.println(name.toLowerCase());
   ```
   Which kind of checking will it fail? …

2. Consider the following method to square a list.
   ```
   public static MyList squareList(MyList list) {
       MyList p = list;
       while (p != null) {
           p.value *= p.value;
           p = p.next;
       }
   return list;
   }
   ```
   The method will also … the input list.

3. In a Javadoc comment, we use **@param** tag to denote the … in a specification of a method.

4. To create our own unchecked exception, we must subclass and extend the class ….

5. To allow types such as Integer, String, and user-defined types to be a parameter to methods, classes, and interfaces, we use ….
   Using it, we can create classes that work with different data types.

6. Write one line of code to call the overridden `toString()` method of the superclass:
   …

7. Assume we have 9 items, represented by integers 0 through 8 in a `WeightedQuickUnion` data structure. All items are initially unconnected to each other. Given the following series of `connect(p, q)` operations, where we break ties by connecting p's root to q's root :
   ```
   connect(2, 3);
   connect(1, 2);
   ```

```
connect(5, 7);
connect(8, 4);
connect(7, 2);
connect(0, 6);
connect(0, 8);
connect(6, 7);
```
What is the output of `parent(4)` ? …

8. If we want to define a class that implements the interface `Iterator<Double>`, we need to define two methods: method `public boolean hasNext()` and what method? Write the method signature.
   …

9. Consider the following implementations of the `Integer`'s `hashCode()` method:
```
public int hashCode() {
    return 25;
}
```
   This is a poor hash code because … will occur all the time.

10. Consider the following code:
```
synchronized (this) {
    this.balance += depositAmt;
}
```
   We use … to ensure thread safety of the shared memory.

Question 2. Linked Data Structure, Generics (20 points)

Complete the method `void insertN(T item, int index, int n)` of `SLList<T>`.

It inserts n copies of item at the given index.

If index is after the end of the list, insert the n new items at the end.

The index is valid position to insert within the list.

Test case 1:

```
SLList<String> list = new SLList<>();
list.addLast("a");
list.addLast("c");
list.insertN("b", 1, 2);
list.print();                  →  a b b c
list.insertN("d", 4, 3);
list.print();                  →  a b b c d d d
```

Question 3. ADT, Encapsulation, Deep Copy (20 points)

Consider the following ADT SynCollection:

```
1  /**
2  * Represents a list of collections of strings
3  * that are synonymous in English,
4  * e.g. {"large", "big", "sizeable"}.
5  * Each collection is considered fixed,
6  * so it never changes once created,
7  * but new collections may be added to this list.
8  */
9  public class SynCollection {
10    public final List<Set<String>> collections;
11
12    /** Make an empty SynCollection */
13    public SynCollection() {
14        this.collections = new ArrayList<Set<String>>();
15    }
16
17    /** Make SynCollection from an existing SynCollection
18    * @param other */
19    public SynCollection(SynCollection other) {
20        this.collections = other.collections;
21    }
22
23    /** Add a new collection of strings
24    * @param newColl set of strings that are synonymous */
25    public void addCollection(Set<String> newColl) {
26        this.collections.add(newColl);
27    }
28
29   /** Get all collections that share a particular word
30   * @param word String to look for
31   * @param result list that receives the collections found
32   * Adds all collections that contain word to the result list. */
33   public void filter(String word, List<Set<String>> result) {
```

```
34        for (Set<String> coll : this.collections) {
35            if (coll.contains(word))
36                result.add(coll);
37            }
38        }
39    }
40 }
```

Fill in the box with the answer.

What type of *ADT operation* is the following method ? **(2 points each)**

```
SynCollection()
SynCollection(SynCollection other)
void addCollection(Set<String> newColl)
void filter(String word, List<Set<String>> result)
```

Complete the statements below regarding the concepts of encapsulation and representation exposure **(2 points each)**

Line 10 breaks …, the `collections` representation field is publicly-accessible. To fix it, we have to make it ….

Line 20 exposes the representation by sharing a … list instance between two `SynCollection` objects. What we should do instead it to make a/an … of the `collections` list.

Line … exposes the representation by directly adding a set to the result list. What should we do it to … the set before that.

Question 4. Equality, Inheritance (20 points)

Complete the method **`boolean equals(Object that)`** of an iterable ARDeque that overrides the one in `Object`.

To be the same, they must be of the same size, and must share at most 2 different items at the same position.

Suppose `[]` is an `ARDeque<Integer>`. Then:

```
[1, 2, 3, 4].equals([1, 2, 5, 6])   →  true
[1, 2, 3, 4].equals([1, 4, 2, 3])   →  false
```

Question 5. Invariant, Concurrency, Deadlock (20 points)

Given the following `Devil` class:

```
public class Devil {
    private final String name;
    private final String type;
    private final Set<Devil> friends;
    // Invariant:
    // name, type, friends are not null
    // a is friend with b if and only b is friend with a
    // i.e. for all f in friends, f.friends contains this

    public Devil(String name, String type) {
        this.name = name;
        this.type = type;
        this.friends = new HashSet<>();
    }

    public synchronized boolean isFriendWith(Devil that) {
        return this.friends.contains(that);
    }

    public synchronized void addFriend(Devil that) {
        if (this.friends.add(that)) {
            that.addFriend(this);
        }
    }

    public synchronized void remFriend(Devil that) {
        if (this.friends.remove(that)) {
            that.remFriend(this);
        }
    }
}
```

Fill in the box with the answer.

a) Given two `Devil` objects:
   `Devil denji = new Devil("Denji", "Chainsaw");`
   `Devil power = new Devil("Power", "Blood");`
   and two threads, Thread A and Thread B.

Assume that currently the invariant holds.
Give two examples of thread operations that may cause deadlock. **(10 points)**

b) Give two strategies that would fix the deadlock problem above. **(10 points)**

---------- THIS IS THE END OF THE DOCUMENT ----------