CPT204 Hyflex 2021
Resit Exam

Overview

This document contains questions for Resit Exam. However, the resit 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:
```
int[] arr = new int[] { 2, 5, 10 };
arr[2] = "4";
```
Which kind of checking will it fail? …

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

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

4. A method that throws a checked exception must advertise the exception in the method signature using the keyword ….

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

6. Write one line of code to call the empty constructor 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(1, 5);
```
What is the output of `parent(7)` ? …

8. If we want to define a class that implements the interface `Iterator<Integer>`, 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, where `intValue()` simply returns the integer value of the Integer object:
```
public int hashCode() {
    return intValue() * intValue();
}
```
   This is a poor hash code because collisions occur when two integers have the same ….
   For example, 5 and -5 will have the same hash code.

10. Consider the following code:
```
synchronized (this) {
    this.balance -= withdrawAmt;
}
```
   Access to the shared memory `balance` is protected by a/an … mechanism.

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

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

It inserts n copies of item at the given index.

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

The index is valid position to insert within the deque.

Test case 1:

```
LLDeque<String> deque = new LLDeque<>();
deque.addLast("a");
deque.addLast("c");
deque.insertN("b", 1, 2);
deque.printDeque();              →  a b b c
deque.insertN("d", 4, 3);
deque.printDeque();              →  a b b c d d d
```

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

Consider the following ADT DeptCourses:

```
1  /**
2  * Represents a list of set of Course objects
3  * that belong to the same department,
4  * e.g. {CPT105, CPT204, CPT403}.
5  * Each set is considered fixed,
6  * so it never changes once created,
7  * but new sets may be added to the list.
8  */
9  public class DeptCourses {
10    public final List<Set<Course>> courses;
11
12    /** Make an empty DeptCourses */
13    public DeptCourses() {
14        this.courses = new ArrayList<Set<Course>>();
15    }
16
17    /** Make DeptCourses from an existing DeptCourses
18      * @param other */
19    public DeptCourses(DeptCourses courses) {
20        this.courses = other.courses;
21    }
22
23    /** Add a new set of courses
24      * @param newCourses set of courses in a new department */
25    public void addCourses(Set<Course> newCourses) {
26        this.courses.add(newCourses);
27    }
28
29    /** Get all courses of this DeptCourses object
30      * @return the courses in this object */
31    public List<Set<Course>> getCourses() {
32        return this.courses;
33    }
34 }
```

Fill in the box with the answer.

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

```
DeptCourses()
```

```
DeptCourses(DeptCourses other)
```

```
void addCourses(Set<Course> newCourses)
```

```
List<Set<Course>> getCourses()
```

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

Line … breaks encapsulation, the instance variable can be accessed by other classes. To fix it, we have to make the instance variable ….

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

Line 32 exposes the representation by returning a … to `courses` out of the ADT, making it modifiable by the method caller. What we should do instead is to return a/an … of the `courses` list.

Question 4. Equality, Inheritance (20 points)

Complete the method **`boolean equals(Object that)`** of `LLDeque` 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 `LLDeque<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 DemonSlayer class:

```
public class DemonSlayer {
    private final String name;
    private final Set<DemonSlayer> siblings;
    // Invariant:
    // name, siblings are not null
    // a is sibling of b if and only b is sibling of a
    // i.e. for all s in siblings, s.siblings contains this

    public DemonSlayer(String name) {
        this.name = name;
        this.siblings = new HashSet<>();
    }

    public synchronized boolean isSiblingOf(DemonSlayer that) {
        return this.siblings.contains(that);
    }

    public synchronized void addSibling(DemonSlayer that) {
        if (this.siblings.add(that)) {
            that.addSibling(this);
        }
    }

    public synchronized void remSibling(DemonSlayer that) {
        if (this.siblings.remove(that)) {
            that.remSibling(this);
        }
    }
}
```

Fill in the box with the answer.

a) Given two DemonSlayer objects:
```
DemonSlayer tanjiro = new DemonSlayer("Tanjiro");
DemonSlayer nezuko = new DemonSlayer("Nezuko");
```
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 ----------