OOPJ CCEE Practice Quiz - 4

Total points 29/40

Duration: 60 Minutes

The respondent's email (**sanket.mandavgane.cmaug25@gmail.com**) was recorded on submission of this form.

0 of 0 points

Name *

Sanket Mandavgane

PRN *

250840320174

MCQ

29 of 40 points

✓ List<String> list = new ArrayList<>(); * 1/1

```
list.add("A");
list.add("B");
list.add("C");
Iterator<String> it = list.iterator();
while(it.hasNext()) {
   String s = it.next();
   if(s.equals("B")) {
      list.add("D");
   }
}
```

What will happen when this code runs?

- ○ A) Prints A, B, C, D
- ◉ B) ConcurrentModificationException ✓
- ○ C) Compilation error
- ○ D) Infinite loop

✓ Map<String, Integer> map = new HashMap<>(); * 1/1

map.put("A", 1);

map.put("B", 2);

map.put("A", 3);

map.put(null, 4);

map.put("C", null);

System.out.println(map.size() + " " + map.get("A") + " " + map.get(null));

What is the output?

- A) 5 1 4
- B) 4 3 4 ✓
- C) 4 1 null
- D) Compilation error

✕ Set<StringBuilder> set = new HashSet<>(); * 0/1

StringBuilder sb1 = new StringBuilder("Hello");

StringBuilder sb2 = new StringBuilder("Hello");

set.add(sb1);

set.add(sb2);

sb1.append(" World");

System.out.println(set.size() + " " + set.contains(sb1));

What is the output?

- ○ A) 1 true
- ◉ B) 2 true ✕
- ○ C) 2 false
- ○ D) 1 false

Correct answer

- ◉ C) 2 false

✕ List<Integer> list = Arrays.asList(1, 2, 3); * 0/1

Collections.reverse(list);

list.add(4);

System.out.println(list);

What happens?

- (●) A) Prints [3, 2, 1, 4] ✕
- ( ) B) Prints [4, 3, 2, 1]
- ( ) C) UnsupportedOperationException
- ( ) D) Compilation error

Correct answer

- (●) C) UnsupportedOperationException

✓ TreeSet<String> set = new TreeSet<>(); * 1/1

```
set.add("banana");
set.add("apple");
set.add("cherry");
set.add(null);
for(String s : set) {
    System.out.print(s + " ");
}
```

What happens?

- ○ A) Prints apple banana cherry null
- ○ B) Prints null apple banana cherry
- ◉ C) NullPointerException ✓
- ○ D) Prints banana apple cherry null

✗ What is the fundamental difference between fail-fast and fail-safe iterators in Java Collections? *0/1

- ○ A) Fail-fast works on original collection, fail-safe works on copy
- ◉ B) Fail-fast throws exceptions, fail-safe never throws exceptions ✗
- ○ C) Fail-fast is faster, fail-safe is safer
- ○ D) Fail-fast is for Lists, fail-safe is for Sets

Correct answer

- ◉ A) Fail-fast works on original collection, fail-safe works on copy

✓ Why does HashMap use both hashCode() and equals() methods, and what happens if they're not consistent? *1/1

- ○ A) hashCode() for insertion, equals() for retrieval
- ◉ B) hashCode() determines bucket, equals() resolves collisions ✓
- ○ C) They serve the same purpose as backup
- ○ D) hashCode() is for performance, equals() is for correctness

✓ What is the significance of load factor in HashMap and how does it affect performance? *1/1

- ○ A) Higher load factor = better performance always
- ◉ B) Load factor determines when to rehash for balance of space vs time ✓
- ○ C) Load factor only affects memory usage, not time complexity
- ○ D) Load factor is fixed and cannot be changed

✓ What is the difference between Comparable and Comparator, and when should each be used? *1/1

- ○ A) Comparable is for primitives, Comparator is for objects
- ◉ B) Comparable provides natural ordering, Comparator provides custom ordering ✓
- ○ C) Comparable is faster, Comparator is more flexible
- ○ D) They are interchangeable and serve the same purpose

✗ System.out.println(new ArrayList<>().add("test")); * 0/1

What does this print?

- A) test
- B) true
- C) [test] ✗
- D) Compilation error

Correct answer

- B) true

✓ System.out.println(Arrays.asList(1,2,3).getClass().getSimpleName()); * 1/1

What is the output?

- A) ArrayList
- B) List
- C) Arrays$ArrayList ✓
- D) AbstractList

✗ System.out.println(Collections.emptyList() == Collections.emptyList()); * 0/1

What does this print?

- A) true
- B) false
- C) Compilation error ✗
- D) Depends on JVM

Correct answer

- A) true

✗ System.out.println(new TreeSet<>(Arrays.asList(3,1,4,1,5)).size()); * 0/1

What is the output?

- A) 5 ✗
- B) 4
- C) 3
- D) Compilation error

Correct answer

- B) 4

✗ System.out.println(new LinkedHashMap<>().put("key", "value")); * 0/1

What does this print?

- A) value
- B) key
- C) null
- D) {key=value} ✗

Correct answer

- C) null

✓ **A shopping cart system needs to:** * 1/1

- Allow duplicate items
- Maintain the order items were added
- Provide fast access by index

Which collection is most suitable?

- A) HashSet
- B) LinkedHashSet
- C) ArrayList ✓
- D) TreeSet

✓ **An online voting system needs to ensure:** * 1/1

- No duplicate votes from same voter ID
- Fast lookup to check if voter has already voted
- No specific ordering required

Which collection should be used to store voter IDs?

- ○ A) ArrayList
- ○ B) LinkedList
- ◉ C) HashSet ✓
- ○ D) TreeMap

✗ **A student grade system needs to:** * ···/1

- Map student names to their grades
- Maintain alphabetical order of students
- Allow grade updates

Which collection is most appropriate?

- ○ A) HashMap
- ○ B) LinkedHashMap
- ○ C) TreeMap
- ○ D) None of these
- ◉ All of these ✗

No correct answers

✓ Which collection allows duplicate elements but maintains insertion order? *1/1

- A) HashSet
- B) TreeSet
- (●) C) ArrayList ✓
- D) HashMap

✗ Which interface does TreeMap implement for automatic sorting? * 0/1

- A) Comparable
- B) Comparator
- (●) C) SortedMap ✗
- D) NavigableMap

Correct answer

- (●) D) NavigableMap

✓ What happens when you add a null value to a TreeSet? * 1/1

- A) It gets added at the beginning
- B) It gets added at the end
- (●) C) NullPointerException is thrown ✓
- D) It gets ignored

✓ Which collection maintains insertion order and allows fast insertion/deletion at both ends? *1/1

- ○ A) ArrayList
- ◉ B) LinkedList ✓
- ○ C) Vector
- ○ D) Stack

✓ What is the difference between HashMap and LinkedHashMap? * 1/1

- ○ A) HashMap is synchronized, LinkedHashMap is not
- ◉ B) LinkedHashMap maintains insertion order, HashMap doesn't ✓
- ○ C) HashMap allows null keys, LinkedHashMap doesn't
- ○ D) LinkedHashMap is faster than HashMap

✓ Which method is used to safely remove elements while iterating through a collection? *1/1

- ○ A) collection.remove()
- ○ B) iterator.delete()
- ◉ C) iterator.remove() ✓
- ○ D) collection.delete()

✓ Set<String> set = new HashSet<>(); * 1/1

set.add("A");

set.add("B");

set.add("A");

System.out.println(set.size());

What will be the output of this code?

- A) 1
- B) 2 ✓
- C) 3
- D) Compilation Error

✓ Which collection class is best for implementing a LIFO (Last In First Out) structure? *1/1

- A) ArrayList
- B) LinkedList
- C) Stack ✓
- D) Queue

✗ What is the default initial capacity of ArrayList? * 0/1

- ○ A) 8
- ○ B) 10
- ◉ C) 16 ✗
- ○ D) 32

Correct answer

- ◉ B) 10

✓ Which interface should a class implement to be stored in a TreeSet without providing a Comparator? *1/1

- ○ A) Serializable
- ○ B) Cloneable
- ◉ C) Comparable ✓
- ○ D) Iterator

✓ Which collection allows duplicate keys? * 1/1

- ○ A) HashMap
- ○ B) TreeMap
- ○ C) LinkedHashMap
- ◉ D) None of the above ✓
- ○ All of these

✓ What will happen if you try to add elements to a collection while iterating using enhanced for loop? *1/1

- ○ A) Elements will be added successfully
- ◉ B) ConcurrentModificationException ✓
- ○ C) Compilation error
- ○ D) Elements will be ignored

✓ Which method is used to convert a Collection to Array? * 1/1

- ◉ A) toArray() ✓
- ○ B) convertToArray()
- ○ C) getArray()
- ○ D) asArray()

✓ What is the load factor of HashMap by default? * 1/1

- A) 0.5
- B) 0.75 ✓
- C) 1.0
- D) 0.25

✓ Which collection is synchronized by default? * 1/1

- A) ArrayList
- B) HashMap
- C) Vector ✓
- D) HashSet

✓ Which method is used to check if a Map contains a specific key? * 1/1

- A) hasKey()
- B) containsKey() ✓
- C) keyExists()
- D) findKey()

✓ What will be the iteration order of elements in a TreeSet? * 1/1

- ○ A) Insertion order
- ○ B) Random order
- ◉ C) Sorted order ✓
- ○ D) Reverse insertion order

✓ What are the different ways to create threads in Java and what are their advantages/disadvantages? *1/1

- ○ A) Only by extending Thread class
- ○ B) Extending Thread class or implementing Runnable interface
- ◉ C) Extending Thread, implementing Runnable, using Callable with ExecutorService ✓
- ○ D) Only by implementing Runnable interface

✓ class MyThread extends Thread { * 1/1

```
    public void run() {
        for(int i = 0; i < 3; i++) {
            System.out.println(Thread.currentThread().getName() + ": " + i);
            try { Thread.sleep(100); } catch(InterruptedException e) {}
        }
    }
}

public class Test {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();
        t1.run();
        t2.start();
    }
}
```

What is the difference in execution between t1.run() and t2.start()?

- ○ A) Both create new threads and run concurrently
- ◉ B) t1.run() executes in main thread, t2.start() creates new thread ✓
- ○ C) Both execute in main thread sequentially
- ○ D) t1.run() creates thread, t2.start() doesn't

✓ class Counter implements Runnable { * 1/1

```
    private static int count = 0;

    public void run() {
        for(int i = 0; i < 1000; i++) {
            count++;
        }
        System.out.println("Final count: " + count);
    }
}

public class Test {
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();
        Thread t1 = new Thread(counter);
        Thread t2 = new Thread(counter);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
    }
}
```

What is the most likely outcome and why?

A) Always prints "Final count: 2000" twice

B) Prints unpredictable values less than or equal to 2000 ✓

C) Compilation error

D) Always prints "Final count: 1000" twice

✓ public class Example { * 1/1

```
public static void main(String[] args) throws InterruptedException {
    Thread t1 = new Thread(() -> {
        try { Thread.sleep(2000); } catch(InterruptedException e) {}
        System.out.println("Thread 1 finished");
    });
    Thread t2 = new Thread(() -> {
        try { Thread.sleep(1000); } catch(InterruptedException e) {}
        System.out.println("Thread 2 finished");
    });
    t1.start();
    t2.start();
    t1.join();
    System.out.println("Main finished");
  }
}
```

What is the execution order?

- ○ A) Thread 1 → Thread 2 → Main
- ◉ B) Thread 2 → Thread 1 → Main ✓
- ○ C) Thread 2 → Main → Thread 1
- ○ D) Unpredictable order

✓ new Thread(() -> System.out.println("Hello")).start(); * 1/1

What does this line demonstrate?

- (●) A) Anonymous thread creation using lambda expression ✓
- ( ) B) Compilation error due to lambda syntax
- ( ) C) Creates thread but doesn't start it
- ( ) D) Synchronous execution in main thread

✗ System.out.println(Thread.currentThread().getState()); * 0/1

If this executes in main method, what will it print?

- ( ) A) NEW
- ( ) B) RUNNABLE
- (●) C) RUNNING ✗
- ( ) D) ACTIVE

Correct answer

- (●) B) RUNNABLE

Experience Zone - No formality 0 of 0 points

Level of Exam * Dropdown

Moderate

I will keep revising all the concepts time to time.
(Without revision all progress will be diminished with time). Now, I will also focus in Communicating in English as well. *

Dropdown

Koi kuch bhi bole - *60% leaao bas****... I will achieve 100% in CCEE.***

I Promise