Elite Quiz 19

Question **1**

Correct

Mark 1.00 out of 1.00

Which of the following statements about the Callable call() and Runnable run() methods are correct? (Choose all that apply.)

Select one or more:

- [✓ a. Both can throw unchecked exceptions.] ✓
- b. Callable takes a generic method argument.
- [✓ c. Callable can throw a checked exception.] ✓
- [✓ d. Both can be implemented with lambda expressions.] ✓
- e. Runnable returns a generic type.
- [✓ f. Callable returns a generic type.] ✓

A, C, D, F. Runnable and Callable statements both take no method arguments as input, so B is incorrect. Runnable returns void and Callable returns a generic type, so F is correct, and E and G are incorrect. All methods are capable of throwing unchecked exceptions, so A is correct. Only Callable is capable of throwing checked exceptions, so C is correct. Both Runnable and Callable can be implemented with lambda expressions, so D is correct.

- g. Both methods return void.

Your answer is correct.

The correct answer is: Both can throw unchecked exceptions., Callable can throw a checked exception., Both can be implemented with lambda expressions., Callable returns a generic type.

Question **2**

Correct

Mark 1.00 out of 1.00

Which of these statements compile? (Choose all that apply.)

Select one or more:

- a. HashSet<Number> hs = new HashSet<Integer>();
- [✓ b. HashSet<? super ClassCastException> set = new HashSet<Exception>();] ✓
- [✓ c. List<String> list = new Vector<String>();] ✓
- d. List<Object> values = new HashSet<Object>();
- e. List<Object> objects = new ArrayList<? extends Object>();
- [✓ f. Map<String, ? extends Number> hm = new HashMap<String, Integer>();] ✓

B, C, F. Option A does not compile because the generic types are not compatible. We could say HashSet hs2 = new HashSet();. Option B uses a lower bound, so it allows superclass generic types. Option C is a traditional use of generics where the generic type is the same and the List type uses the interface as the type. Option D does not compile because a Set is not a List. Option E does not compile because upper bounds are not allowed when instantiating the type. Finally, Option F does compile because the upper bound is on the correct side of =.

Your answer is correct.

The correct answer is: HashSet<? super ClassCastException> set = new HashSet<Exception>();, List<String> list = new Vector<String>();, Map<String, ? extends Number> hm = new HashMap<String, Integer>();

Question 3

Correct

Mark 1.00 out of 1.00

Which happens when more tasks are submitted to a thread executor than available threads?

Select one or more:

- a. The thread executor will throw an exception when a task is submitted that is over its thread limit.
- [✓ b. The task will be added to an internal queue and completed when there is an available thread.]

✓

B. If a task is submitted to a thread executor, and the thread executor does not have any available threads, the call to the task will return immediately with the task being queued internally by the thread executor. For this reason, B is the only correct answer.

- c. The thread executor will discard any task over its thread limit.
- d. The call to submit the task to the thread executor will wait until there is a thread available before continuing.
- e. The thread executor creates new temporary threads to complete the additional tasks.

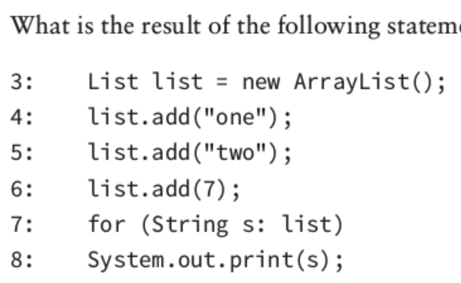
Your answer is correct.

The correct answer is: The task will be added to an internal queue and completed when there is an available thread.

Question 4

Correct

Mark 1.00 out of 1.00



What is the result of the following statements?

```
3:    List list = new ArrayList();
4:    list.add("one");
5:    list.add("two");
6:    list.add(7);
7:    for (String s: list)
8:    System.out.print(s);
```

Select one or more:

- a. onetwo
- b. onetwo7
- c. onetwo followed by an exception
- d. Compiler error on line 6
- [✓ e. Compiler error on line 7]

✓

E. The code does not compile. It attempts to mix generics and legacy code. Lines 3 through 7 create an ArrayList without generics. This means that we can put any objects in it. Line 7 should be looping through a list of Objects rather than Strings since we didn't use generics.

Your answer is correct.

The correct answer is: Compiler error on line 7

Question **5**

Correct

Mark 1.00 out of 1.00

What is the result of executing the following code snippet?

```
List<Integer> l1 = Arrays.asList(1,2,3);
List<Integer> l2 = new CopyOnWriteArrayList<>(l1);
Set<Integer> s3 = new ConcurrentSkipListSet<>();
s3.addAll(l1);
for(Integer item: l2) l2.add(4); // x1
for(Integer item: s3) s3.add(5); // x2
System.out.println(l1.size()+" "+l2.size()+" "+s3.size());
```

Select one or more:

[✓] a. It outputs 3 6 4. ✓

A. The code compiles without issue, so D is incorrect. The CopyOnWriteArrrayList class is designed to preserve the original list on iteration, so the first loop will be executed exactly three times and E is incorrect. The ConcurrentSkipListSet class allows modifications while iterating, so it is possible that the second loop could generate an infinite loop. In this case, though, the second loop executes exactly four times, since elements in a set are unique and 5 can be added only once. For these reasons, F and G are also incorrect. Finally, despite using the elements of l1 to populate the collections, l2 and s3 are not backed by the original list, so the size of l1 is 3. Likewise, the size of l2 is 6 and the size of s3 is 4, so A is the correct answer.

[ ] b. It outputs 6 6 6.

[ ] c. It outputs 6 3 4.

[ ] d. The code does not compile.

[ ] e. It compiles but throws an exception at runtime on line x1.

[ ] f. It compiles but throws an exception at runtime on line x2.

[ ] g. It compiles but enters an infinite loop at runtime.

Your answer is correct.

The correct answer is: It outputs 3 6 4.

Question **6**

Correct

Mark 1.00 out of 1.00

Which of the following properties of concurrency are true? (Choose all that apply.)

Select one or more:

[✓] a. By itself, concurrency does not guarantee which task will be completed first. ✓

[ ] b. Concurrency always improves the performance of an application.

[ ] c. Computers with a single processor do not benefit from concurrency.

[✓] d. Applications with many resource-heavy tasks tend to benefit more from concurrency
than ones with CPU-intensive tasks. ✓

A, D. By itself, concurrency does not guarantee which task will be completed first, so A is correct. Furthermore, applications with numerous resource requests will often be stuck waiting for a resource, which allows other tasks to run. Therefore, they tend to benefit more from concurrency than CPU-intensive tasks, so D is also correct. B is incorrect because concurrency may in fact make an application slower if it is truly single-threaded in nature. Keep in mind that there is a cost associated with allocating additional memory and CPU time to manage the concurrent process. C is incorrect because single-processor CPUs have been benefiting from concurrency for decades. Finally, E is incorrect; there are numerous examples in this chapter of concurrent tasks sharing memory.

[ ] e. Concurrent tasks do not share the same memory.

Your answer is correct.

The correct answer is: By itself, concurrency does not guarantee which task will be completed first., Applications with many resource-heavy tasks tend to benefit more from concurrency
than ones with CPU-intensive tasks.

Question 7

Correct

Mark 1.00 out of 1.00

What is the result of the following statements?

```
3. ArrayDeque<String> greetings = new ArrayDeque<String>();
4. greetings.push("hello");
5. greetings.push("hi");
6. greetings.push("ola");

7: greetings.pop();
8: greetings.peek();
9: while (greetings.peek() != null)
10: System.out.print(greetings.pop());
```

Select one or more:

- ☐ a. hello
- ☐ b. hellohi
- ☐ c. hellohiola
- ☐ d. hi
- ☑ e. hihello ✓

E. Since we call push() rather than offer(), we are treating the ArrayDeque as a LIFO (last-in, first-out) stack. On line 7, we remove the last element added, which is "ola". On line 8, we look at the new last element ("hi"), but don’t remove it. Lines 9 and 10, we remove each element in turn until none are left. Note that we don’t use an Iterator to loop through the ArrayDeque. The order in which the elements are stored internally is not part of the API contract.

- ☐ f. The code does not compile.
- ☐ g. An exception is thrown.

Your answer is correct.

The correct answer is: hihello

Question 8

Correct

Mark 1.00 out of 1.00

What is the result of executing the following application? (Choose all that apply.)

```
import java.util.concurrent.*;
public class CountNumbers extends RecursiveAction {
private int start;
private int end;
public CountNumbers(int start, int end) {
this.start = start;
this.end = end;
}
protected void compute() {
if (start<0) return;
else {
int middle = start + ((end—start) / 2);
invokeAll(new CountNumbers(start, middle),
new CountNumbers(middle, end)); // m1
}
}

public static void main(String[] args) {
ForkJoinTask<?> task = new CountNumbers(0, 4); // m2
ForkJoinPool pool = new ForkJoinPool();
Object result = pool.invoke(task); // m3
}
}
```

Select one or more:

- ☐ a. It compiles and runs without issue.
- ☐ b. The code will not compile because of m1.
- ☐ c. The code will not compile because of m2.
- ☐ d. The code will not compile because of m3.
- ☑ e. It compiles but throws an exception at runtime. ✓

  E. The program compiles without issue, so B, C, and D are incorrect. Lines m2 and m3 throw a compiler warning about generics but still compile. Notice that RecursiveAction, unlike RecursiveTask, does not return a value. However, since we used a generic ForkJoinTask reference, the code still compiles. The issue here is that the base condition is not reached since the numbers start/end are consistently positive. This causes an infinite loop, although since memory is finite, Java detects this and throws a StackOverflowError, so E is correct. In practice, this could also generate a locking exception before the StackOverflowError when the program runs out of memory, but in either circumstance, the program will exit.
- ☐ f. It compiles but hangs at runtime.

Your answer is correct.

The correct answer is: It compiles but throws an exception at runtime.

Question **9**

Correct

Mark 1.00 out of 1.00

What is the result of executing the following application? (Choose all that apply.)

```
import java.util.concurrent.*;
import java.util.stream.*;
public class PrintConstants {
   public static void main(String[] args) {
      ExecutorService service = Executors.newScheduledThreadPool(10);
      DoubleStream.of(3.14159,2.71828) // b1
         .forEach(c -> service.submit( // b2
            () -> System.out.println(10*c))); // b3
      service.execute(() -> System.out.println("Printed")); // b4
   }
}
```

Select one or more:

- a. It compiles and outputs the two numbers, followed by Printed.
- b. The code will not compile because of line b1.
- c. The code will not compile because of line b2.
- d. The code will not compile because of line b3.
- e. The code will not compile because of line b4.
- ✓ f. It compiles but the output cannot be determined ahead of time. ✓
- g. It compiles but throws an exception at runtime.
- ✓ h. It compiles but waits forever at runtime. ✓

F, H. The application compiles and does not throw an exception, so B, C, D, E, and G are incorrect. Even though the stream is processed in sequential order, the tasks are submitted to a thread executor, which may complete the tasks in any order. Therefore, the output cannot be determined ahead of time and F is correct, making A incorrect. Finally, the thread executor is never shut down; therefore the code will run but it will never terminate, making H also correct

Your answer is correct.

The correct answer is: It compiles but the output cannot be determined ahead of time., It compiles but waits forever at runtime.

Question **10**

Correct

Mark 1.00 out of 1.00

What is the result of the following code?

```
TreeSet<String> tree = new TreeSet<String>();
tree.add("one");
tree.add("One");
tree.add("ONE");
System.out.println(tree.ceiling("On"));
```

Select one or more:

- a. On
- b. one
- ✓ c. One ✓

C. TreeSet sorts the elements. Since uppercase letters sort before lowercase letters, the ordering is "ONE", "One", "one". The ceiling() method returns the smallest element greater than the specified one. "On" appears between "ONE" and "One". Therefore, the smallest element that is larger than the specified value is "One".

- d. ONE
- e. The code does not compile.
- f. An exception is thrown.

Your answer is correct.

The correct answer is: One

Question 11

Correct

Mark 1.00 out of 1.00

Assuming 100 milliseconds is enough time for the tasks submitted to the thread executor to complete, what is the result of executing the following program? (Choose all that apply.)

```
import java.util.concurrent.*;
public class SheepManager {
   private static AtomicInteger sheepCount1 = new AtomicInteger(0); // w1
   private static int sheepCount2 = 0;

   public static void main(String[] args) throws InterruptedException {
      ExecutorService service = null;
      try {
         service = Executors.newSingleThreadExecutor(); // w2

         for(int i=0; i<100; i++)
            service.execute(() ->
               {sheepCount1.getAndIncrement(); sheepCount2++;}); // w3
         Thread.sleep(100);
         System.out.println(sheepCount1+" "+sheepCount2);
      } finally {
         if(service != null) service.shutdown();
      }
   }
}
```

Select one or more:

- ☐ a. It outputs 100 99.
- [✓] b. It outputs 100 100. ✓

  B. The code compiles and runs without issue, so D, E, F, and G are incorrect. The key aspect to notice in the code is that a single-thread executor is used, meaning that no task will be executed concurrently. Therefore, the results are valid and predictable with 100 100 being the output, and B is the correct answer. If a pooled thread executor was used with at least two threads, then the sheepCount2++ operations could overwrite each other, making the second value indeterminate at the end of the program. In this case, C would be the correct answer.

- ☐ c. The output cannot be determined ahead of time.
- ☐ d. The code will not compile because of line w1.
- ☐ e. The code will not compile because of line w2.
- ☐ f. The code will not compile because of line w3.
- ☐ g. It compiles but throws an exception at runtime.

Your answer is correct.

The correct answer is: It outputs 100 100.

Question **12**

Correct

Mark 1.00 out of 1.00

What statements about the following class definition are true? (Choose all that apply.)

```
public class TicketManager {
   private TicketManager() { super(); }
   private static TicketManager instance;
   public static synchronized TicketManager getInstance() { // k1
      if (instance == null) instance = new TicketManager(); // k2
      return instance;
   }

   private int tickets;
   public int getTicketCount() { return tickets; }
   public void makeTicketsAvailable(int value) { tickets += value; } // k3
   public void sellTickets(int value) {
      synchronized (this) { // k4
         tickets -= value;
      }
   }
}
```

Select one or more:

[✓] a. It compiles without issue. ✓

[ ] b. The code will not compile because of line k2.

[ ] c. The code will not compile because of line k3.

[ ] d. The lock locks acquired on k1 and k4 are on the same object.

[ ] e. The class correctly prevents concurrency issues for the value of tickets when accessed by multiple threads.

[✓] f. At most one instance of TicketManager will be created in the application. ✓

A, F. The class compiles without issue so A is correct, and B and C are incorrect. The synchronized object on line k1 is TicketManager.class, while the synchronized object on line k4 is the instance of TicketManager. The class is not thread-safe because the makeTicketsAvailable() method is not synchronized, and E is incorrect. One thread could call sellTickets() while another thread has unblocked accessed to makeTicketsAvailable(), causing an invalid number of tickets to be reached as part of a race condition. Finally, F is correct because the class synchronizes using a static getInstance() method, preventing more than one instance from being created.

Your answer is correct.

The correct answer is: It compiles without issue., At most one instance of TicketManager will be created in the application.

Question 13

Correct

Mark 1.00 out of 1.00

What is the result of executing the following program?

```
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.*;

public class PrintCounter {
   static int counter = 0;
   public static void main(String[] args) throws InterruptedException,
   ExecutionException {
      ExecutorService service = Executors.newSingleThreadExecutor();
      List<Future<?>> results = new ArrayList<>();
      IntStream.iterate(0,i -> i+1).limit(5).forEach(
            i -> results.add(service.execute(() -> counter++)) // n1
      );
      for(Future<?> result : results) {
         System.out.print(result.get()+" "); // n2
      }
      service.shutdown();
   }
}
```

Select one or more:

- a. It prints 0 1 2 3 4
- b. It prints 1 2 3 4 5
- c. It prints null null null null null
- d. It hangs indefinitely at runtime.
- e. The output cannot be determined.
- [✓ f. The code will not compile because of line n1.] ✓

  F. The key to solving this question is to remember that the execute() method returns void, not a Future object. Therefore, line n1 does not compile and F is the correct answer. If the submit() method had been used instead of execute(), then C would have been the correct answer, as the output of submit(Runnable) task is a Future object which can only return null on its get() method.
- g. The code will not compile because of line n2.

Your answer is correct.

The correct answer is: The code will not compile because of line n1.

Question 14

Correct

Mark 1.00 out of 1.00

Fill in the blanks: _________ occur(s) when two or more threads are blocked forever but both appear active. _________occur(s) when two or more threads try to complete a related task at the same time.

Select one or more:

- a. Livelock, Deadlock
- b. Deadlock, Starvation
- c. Race conditions, Deadlock
- [✓ d. Livelock, Race conditions] ✓

  D. Livelock occurs when two or more threads are conceptually blocked forever, although they are each still active and trying to complete their task. A race condition is an undesirable result that occurs when two tasks are completed at the same time, which should have been completed sequentially. For these reasons, D is the only correct answer.
- e. Starvation, Race conditions
- f. Deadlock, Livelock

Your answer is correct.

The correct answer is: Livelock, Race conditions

Question 15

Correct

Mark 1.00 out of 1.00

Suppose that you have a collection of products for sale in a database and you need to display those products. The products are not unique. Which of the following collections classes in the java.util package best suit your needs for this scenario?

Select one or more:

- ☐ a. Arrays
- ☑ b. ArrayList ✓

  B. The answer needs to implement List because the scenario allows duplicates. Since you need a List, you can eliminate C, D, and E immediately. HashMap is a Map and HashSet is a Set. LinkedList is both a List and a Queue. You want a regular List. Option A, Arrays, is trying to distract you. It is a utility class rather than a Collection. An array is not a collection. By process of elimination, the answer is B.
- ☐ c. HashMap
- ☐ d. HashSet
- ☐ e. LinkedList

Your answer is correct.

The correct answer is: ArrayList