Java Threads

1. Which of the following methods is not available on an ExecutorService instance?
A. execute(Callable)
B. execute(Runnable)
C. submit(Callable)
D. submit(Runnable)

2. Which statements about executing the following TicketTaker application multiple times are true?

```
package performance;
import java.util.concurrent.atomic.*;
import java.util.stream.*;
public class TicketTaker {
long ticketsSold;
final AtomicInteger ticketsTaken;
public TicketTaker() {
ticketsSold = 0;
ticketsTaken = new AtomicInteger(0);
}
public void performJob() {
IntStream.iterate(1, p -> p+1)
.parallel()
.limit(10)
.forEach(i -> ticketsTaken.getAndIncrement());
IntStream.iterate(1, q -> q+1)
.limit(5)
.parallel()
.forEach(i -> ++ticketsSold);
System.out.print(ticketsTaken+" "+ticketsSold);
}
public static void main(String[] matinee) {
new TicketTaker().performJob();
}
}
```

I. The class compiles and runs without throwing an exception.
II. The first number printed is consistently 10.
III. The second number printed is consistently 5.
A. I only
B. I and II
C. I, II, and III
D. None of the above

3. Which of the following is a recommended way to define an asynchronous task?
A. Create a Callable expression and pass it to an instance of Executors.
B. Create a class that extends Thread and overrides the start() method.
C. Create a Runnable expression and pass it to a Thread constructor.
D. All of the above

4. Let's say you needed a thread executor to create tasks for a CyclicBarrier that has a barrier limit of five threads. Which static method in ExecutorService should you use to obtain it?
A. newSingleThreadExecutor()
B. newSingleThreadScheduledExecutor()
C. newCachedThreadPool()
D. None of these would work.

5. Given the original array, how many of the following for statements result in an exception at runtime, assuming each is executed independently?

```
List<Integer> original = new ArrayList<>(Arrays.asList(1,2,3,4,5));
List<Integer> copy1 = new CopyOnWriteArrayList<>(original);
for(Integer w : copy1)
copy1.remove(w);
List<Integer> copy2 = Collections.synchronizedList(original);
for(Integer w : copy2)
copy2.remove(w);
List<Integer> copy3 = new ArrayList<>(original);
for(Integer w : copy3)
copy3.remove(w);
Queue<Integer> copy4 = new ConcurrentLinkedQueue<>(original);
for(Integer w : copy4)
copy4.remove(w);
```

A. Zero
B. One
C. Two
D. Three

6. Fill in the blanks: is a special case of , in which two or more active threads try to acquire the same set of locks and are repeatedly unsuccessful.
A. Deadlock, livelock
B. Deadlock, resource starvation
C. Livelock, resource starvation
D. Resource starvation, race conditions

7. What is the output of the following application?

```
1: package office;
2: import java.util.concurrent.*;
3: public class TpsReport {
4: public void submitReports() {
5: ExecutorService service = Executors.newCachedThreadPool();
6: Future bosses = service.submit(() -> System.out.print(""));
7: service.shutdown();
8: System.out.print(bosses.get());
9: }
10: public static void main(String[] memo) {
11: new TpsReport().submitReports();
12: }
13: }
```

A. null
B. The code does not compile.
C. Line 7 throws an exception at runtime.

D. Line 8 throws an exception at runtime.

8. Which of the following static methods does not exist in the Executors class?
A. newFixedScheduledThreadPool()
B. newFixedThreadPool()
C. newSingleThreadExecutor()
D. newSingleThreadScheduledExecutor()

9. How many times does the following application print Ready at runtime?

```
package parade;
import java.util.concurrent.*;
public class CartoonCat {
private void await(CyclicBarrier c) {
try {
c.await();
} catch (Exception e) {}
}
public void march(CyclicBarrier c) {
ExecutorService s = Executors.newSingleThreadExecutor();
for(int i=0; i<12; i++)
s.execute(() -> await(c));
s.shutdown();
}
public static void main(String... strings) {
new CartoonCat().march(new CyclicBarrier(4,
() -> System.out.println("Ready")));
}
}
```

A. Zero
B. One
C. Three
D. The code does not compile

12. What is the result of executing the following application multiple times?

```
package bears;
import java.util.*;
public class Bounce {
public static void main(String... legend) {
Arrays.asList(1,2,3,4).stream()
.forEach(System.out::println);
Arrays.asList(1,2,3,4).parallel()
.forEachOrdered(System.out::println);
}
}
```

A. Only the first array is printed in the same order every time.
B. Only the second array is printed in the same order every time.
C. Both arrays are print
D. None of the above

14. Given the following code snippet, which lambda expression is the best choice for the accumulator, based on the rules for applying a parallel reduction?

```
public class GoodAccumulator {
int i;
public void test() {
BiFunction<Integer,Integer,Integer> accumulator = ;
System.out.print(Arrays.asList(1,2,3,4,5)
.parallelStream()
.reduce(0,accumulator,(s1, s2) -> s1 + s2));
}
}
```

A. (a,b) -> (a-b)
B. (a,b) -> 5
C. (a,b) -> i++
D. None of the above are appropriate.

15. What is the output of the following code snippet?

```
Callable c = new Callable() {
public Object run() {return 10;}
};
ExecutorService s = Executors.newScheduledThreadPool(1);
for(int i=0; i<10; i++) {
Future f = s.submit(c);
f.get();
}
s.shutdown();
System.out.print("Done!");
```

A. Done!
B. The code does not compile.
C. The code hangs indefinitely at runtime.
D. The code throws an exception at runtime.

Answers

1. A. The ExecutorService interface defines the two submit() methods shown in Options C and D. Because ExecutorService extends Executor, it inherits the execute(Runnable) method presented in Option B. That leaves us with the correct answer, Option A, because ExecutorService does not define nor inherit an overloaded method execute() that takes a Callable parameter.

2. B. The class compiles and runs without throwing an exception, making the first statement true. The class defines two values that are incremented by multiple threads in parallel. The first IntStream statement uses an atomic class to update a variable. Since updating an atomic numeric instance is thread-safe by design, the first number printed is always 10, and the second statement is true. The second IntStream statement uses an int with the preincrement operator (++), which is not thread-safe. It is possible two threads could update and set the same value at the same time, a form of race condition, resulting in a value less than 5. For this reason, the third statement is not true. Since only the first two statements are true, Option B is the correct answer.

3. C. Option A is incorrect, although it would be correct if Executors was replaced with ExecutorService in the sentence. While an instance of ExecutorService can be obtained from the Executors class, there is no method in the Executors class that performs a task directly. Option B is also incorrect, but it would be correct if start() was replaced with run() in the sentence. It is recommended that you override the run() method, not the start() method, to execute a task using a custom Thread class. Option C is correct, and one of the most common ways to define an asynchronous task. Finally, Option D is incorrect because Options A and B are incorrect.

4. D. Trick question! ExecutorService does not contain any of these methods. In order to obtain an instance of a thread executor, you need to use the Executors factory class. For this reason, Option D is the correct answer. If the question had instead asked which Executors method to use, then the correct answer would be Option C. Options A and B do not allow concurrent processes and should not be used with a CyclicBarrier expecting to reach a limit of five concurrent threads. Option C, on the other hand, will create threads as needed and is appropriate for use with a CyclicBarrier.

5. C. CopyOnWriteArrayList makes a copy of the array every time it is modified, preserving the original list of values the iterator is using, even as the array is modified. For this reason, the for loop using copy1 does not throw an exception at runtime. On the other hand, the for loops using copy2 and copy3 both throw ConcurrentModificationException at runtime since neither allows modification while they are being iterated upon. Finally, the ConcurrentLinkedQueue used in copy4 completes without throwing an exception at runtime. For the exam, remember that the Concurrent classes order read/write access such that access to the class is consistent across all threads and processes, while the synchronized classes do not. Because exactly two of the for statements produce exceptions at runtime, Option C is the correct answer.

6. C. Resource starvation is when a single active thread is perpetually unable to gain access to a shared resource. Livelock is a special case of resource starvation, in which two or more active threads are unable to gain access to shared resources, repeating the process over and over again. For these reasons, Option C is the correct answer. Deadlock and livelock are similar, although in a deadlock situation the threads are stuck waiting, rather than being active or performing any work. Finally, a race condition is an undesirable result when two tasks that should be completed sequentially are completed at the same time.

7. B. The class does not compile because the Future.get() on line 8 throws a checked InterruptedException and ExecutionException, neither of which is handled nor declared by the submitReports() method. If the submitReports() and accompanying main() methods were both updated to declare these exceptions, then the application would print null at runtime, and Option A would be the correct answer. For the exam, remember that Future can be used with Runnable lambda expressions that do not have a return value but that the return value is always null when completed.

8. A. Options B and C are both proper ways to obtain instances of ExecutorService. Remember that newSingleThreadExecutor() is equivalent to calling newFixedThreadPool() with a value of 1. Option D is the correct way to request a single-threaded ScheduledExecutorService instance. The correct answer is Option A. The method newFixedScheduledThreadPool() does not exist in the Executors class, although there is one called newScheduledThreadPool().

9. A. The code compiles without issue but hangs indefinitely at runtime. The application defines a thread executor with a single thread and 12 submitted tasks. Because only one thread is available to work at a time, the first thread will wait endlessly on the call to await(). Since the CyclicBarrier requires four threads to release it, the application waits endlessly in a frozen condition. Since the barrier is never reached and the code hangs, the application will never output Ready, making Option A the correct answer. If newCachedThreadPool() had been used instead of newSingleThreadExecutor(), then the barrier would be reached

12. D. The static method Array.asList() returns a List instance, which inherits the Collection interface. While the Collection interface defines a stream() and parallelStream() method, it does not contain a parallel() method. For this reason, the second stream statement does not compile, and Option D is the correct answer. If the code was corrected to use parallelStream(), then the arrays would be consistently printed in the same order, and Option C would be the correct answer. Remember that the forEachOrdered() method forces parallel streams to run in sequential order.

14. B. An accumulator in a serial or parallel reduction must be associative and stateless. In a parallel reduction, invalid accumulators tend to produce more visible errors, where the result may be processed in an unexpected order. Option A is not associative, since (a-b)-c is not the same as a-(b-c) for all values a, b, and c. For example, using values of 1, 2, and 3 results in two different values, -4 and 2. Option C is not stateless, since a class or instance variable i is modified each time the accumulator runs. That leaves us with Option B, which is the correct answer since it is both stateless and associative. Even though it ignores the input parameters, it meets the qualifications for performing a reduction.

15. B. The code does not compile because Callable must define a call() method, not a run() method, so Option B is the correct answer. If the code was fixed to use the correct method name, then it would complete without issue, printing Done! at runtime, and Option A would be the correct answer.