Chapter 20
Java Concurrency

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

- ✓**Java Concurrency**
- Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
- Identify potential threading problems among deadlock, starvation, livelock, and race conditions
- Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution
- Use java.util.concurrent collections and classes including CyclicBarrier and CopyOnWriteArrayList
- Use parallel Fork/Join Framework
- Use parallel Streams including reduction, decomposition, merging processes, pipelines and performance.

1. Which of the following methods is not available on an `ExecutorService` instance?

1. `execute(Callable)`
2. `execute(Runnable)`
3. `submit(Callable)`
4. `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();
   }
}
```

2. The first number printed is consistently `10`.
3. The second number printed is consistently `5`.

1. I only
2. I and II
3. I, II, and III
4. None of the above

3. Which of the following is a recommended way to define an asynchronous task?

1. Create a `Callable` expression and pass it to an instance of `Executors`.
2. Create a class that extends `Thread` and overrides the `start()` method.
3. Create a `Runnable` expression and pass it to a `Thread` constructor.
4. 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?

1. `newSingleThreadExecutor()`
2. `newSingleThreadScheduledExecutor()`
3. `newCachedThreadPool()`
4. 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);
```

1. Zero
2. One
3. Two
4. 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.

1. Deadlock, livelock
2. Deadlock, resource starvation
3. Livelock, resource starvation
4. 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: }
```

---

1. `null`
2. The code does not compile.
3. Line 7 throws an exception at runtime.
4. Line 8 throws an exception at runtime.

8. Which of the following `static` methods does not exist in the `Executors` class?

1. `newFixedScheduledThreadPool()`
2. `newFixedThreadPool()`
3. `newSingleThreadExecutor()`
4. `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")));
   }
}
```

---

1. Zero
2. One
3. Three
4. The code does not compile.

10. Which thread-safe class would you use to add elements to the front and back of an ordered data structure and includes methods for waiting a specified amount of time to do so?

1. `BlockingDeque`
2. `ConcurrentLinkedDeque`
3. `ConcurrentSkipListSet`
4. `LinkedBlockingDeque`

11. Three of the four methods below always produce the same result whether they are executed on a serial or parallel ordered stream. Which one does not?

1. `findAny()`
2. `findFirst()`
3. `limit()`
4. `skip()`

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);
   }
}
```

1. Only the first array is printed in the same order every time.
2. Only the second array is printed in the same order every time.
3. Both arrays are printed in the same order every time.
4. None of the above

13. Fill in the blanks: In the fork/join framework, using the _____ class requires overriding an abstract `compute()` method containing a generic return type, while using the _____ class requires overriding an abstract `compute()` method containing a `void` return type.

1. `ForkJoinTask`, `RecursiveAction`
2. `RecursiveAction`, `RecursiveTask`
3. `RecursiveTask`, `ForkJoinTask`
4. `RecursiveTask`, `RecursiveAction`

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));
   }
}
```

1. `(a,b) -> (a-b)`
2. `(a,b) -> 5`
3. `(a,b) -> i++`
4. None of the above are appropriate.

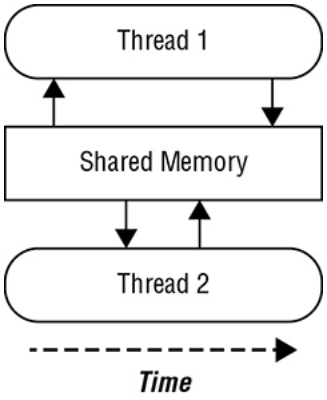
15. What is the output of the following code snippet?

```
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!");
```

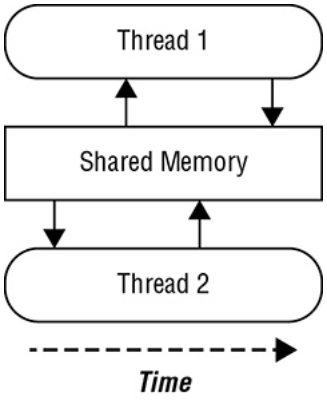
1. `Done!`
2. The code does not compile.
3. The code hangs indefinitely at runtime.
4. The code throws an exception at runtime.

16. The following diagrams represent the order of read/write operations of two threads sharing a common variable. Each thread first reads the value of the variable from memory and then writes a new value of the variable back to memory. Which diagram demonstrates proper synchronization?

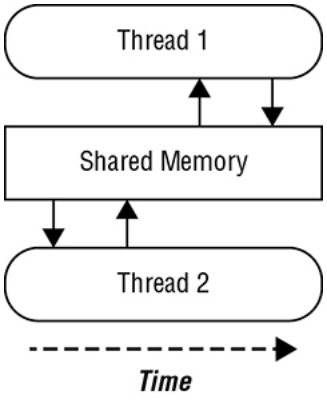
1.


Thread 1
Shared Memory
Thread 2
Time

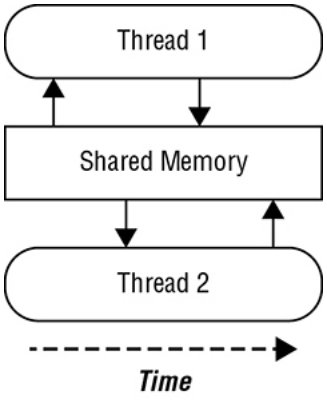
2.


Thread 1
Shared Memory
Thread 2
Time

3.


Thread 1
Shared Memory
Thread 2
Time

4.


Thread 1
Shared Memory
Thread 2
Time

17. What is the output of the following application?

```
package story;
import java.util.*;
import java.util.concurrent.*;
public class Race {
   static ExecutorService service = Executors.newFixedThreadPool(8)
   public static int sleep() {
      try {
         Thread.sleep(1000);
      } catch (Exception e) {}
      return 1;
   }
   public static void hare() {
      try {
         Callable c = () -> sleep();
         final Collection<Callable<Integer>> r = Arrays.asList(c,c,
         List<Future<Integer>> results = service.invokeAll(r);
         System.out.println("Hare won the race!");
      } catch (Exception e) {e.printStackTrace();}
   }
   public static void tortoise() {
      try {
         Callable c = () -> sleep();
         final Collection<Callable<Integer>> r = Arrays.asList(c,c,
         Integer result = service.invokeAny(r);
         System.out.println("Tortoise won the race!");
      } catch (Exception e) {e.printStackTrace();}
   }
   public static void main(String[] p) throws Exception {
      service.execute(() -> hare());
      service.execute(() -> tortoise());
   }
}
```

1. `Hare won the race!` is printed first.
2. `Tortoise won the race!` is printed first.
3. The code does not compile.
4. The result is unknown until runtime.

18. Which of the following concurrent collections is sorted?

1. `ConcurrentLinkedQueue`
2. `ConcurrentSkipListMap`
3. `CopyOnWriteArrayList`
4. `LinkedBlockingQueue`

19. What is the most likely result of executing the following application?

```
package unknown;
import java.util.concurrent.*;
public class Riddle {
   public void sleep() {
      try {
         Thread.sleep(5000);
      } catch (Exception e) {}
   }
   public String getQuestion(Riddle r) {
      synchronized {
         sleep();
         if(r != null) r.getAnswer(null);
         return "How many programmers does it take "
               + "to change a light bulb?";
      }
   }
   public synchronized String getAnswer(Riddle r) {
      sleep();
      if(r != null) r.getAnswer(null);
      return "None, that's a hardware problem";
   }

   public static void main(String... ununused) {
      final Riddle r1 = new Riddle();
      final Riddle r2 = new Riddle();
      ExecutorService s = Executors.newFixedThreadPool(2);
      s.submit(() -> r1.getQuestion(r2));
      s.execute(() -> r2.getAnswer(r1));
      s.shutdown();
   }
}
```

1. A deadlock is produced at runtime.
2. A livelock is produced at runtime.
3. The application completes successfully.
4. The code does not compile.

20. Which `ScheduledExecutorService` method can result in the same action being executed by two threads at the same time?

    1. `scheduleAtFixedDelay()`
    2. `scheduleAtFixedRate()`
    3. `scheduleWithFixedDelay()`
    4. There is no such method in `ScheduledExecutorService`.

21. What is the output of the following application?

```
package olympics;
import java.util.concurrent.*;
public class Athlete {
   int stroke = 0;
   public synchronized void swimming() {
      stroke++;
   }
   public static void main(String... laps) {
      ExecutorService s = Executors.newFixedThreadPool(10);
      Athlete a = new Athlete();
      for(int i=0; i<1000; i++) {
         s.execute(() -> a.swimming());
      }
      s.shutdown();
      System.out.print(a.stroke);
   }
}
```

    1. `1000`
    2. The code does not compile.
    3. The result is unknown until runtime because `stroke` is not accessed in a thread-safe manner and a write may be lost.
    4. The result is unknown until runtime for some other reason.

22. Which of the following is most likely to be caused by a race condition?

    1. A thread perpetually denied access to a resource
    2. An `int` variable incorrectly reporting the number of times an operation was performed
    3. Two threads actively trying to restart a blocked process that is guaranteed to always end the same way
    4. Two threads endlessly waiting on each other to release shared locks

23. What is the output of the following application?

```
package farm;
import java.util.concurrent.*;
public class CountSheep extends RecursiveAction {
   static int[] sheep = new int[] {1,2,3,4};
   final int start;
   final int end;
   int count = 0;
   public CountSheep(int start, int end) {
      this.start = start;
      this.end = end;
   }
   public void compute() {
      if(end-start<2) {
         count+=sheep[start];
         return;
      } else {
         int middle = start + (end-start)/2;
         invokeAll(new CountSheep(start,middle),
               new CountSheep(middle,end));
      }
   }
   public static void main(String[] night) {
      ForkJoinPool pool = new ForkJoinPool();
      CountSheep action = new CountSheep(0,sheep.length);
      pool.invoke(action);
      pool.shutdown();
      System.out.print(action.count);
   }
}
```

    1. `0`
    2. `10`
    3. The code does not compile.
    4. None of the above.

24. Which statement about parallel streams is correct?

1. A parallel stream always executes all stream operations faster than a serial stream.
2. A parallel stream always executes certain stream operations faster than a serial stream.
3. A parallel stream synchronizes its operations so that they are atomic.
4. All streams can be converted to a parallel stream.

25. What is a possible output of the following application?

```
package salvage;
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.*;
public class Car {
   private String model;
   private int year;
   public Car(String name, int year) {
      this.model = name; this.year = year;
   }
   public int getYear() {return year;}
   @Override public String toString() {return model;}
   public static void main(String... make) {
      List<Car> cars = new ArrayList<>();
      cars.add(new Car("Mustang",1967));
      cars.add(new Car("Thunderbird",1967));
      cars.add(new Car("Escort",1975));
      ConcurrentMap<Integer, List<Car>> map = cars
         .stream()
         .collect(Collectors.groupingByConcurrent(Car::getYear));
      System.out.print(map);
   }
}
```

1. `{1975=[Escort], 1967=[Thunderbird, Mustang]}`
2. `{Escort=[1975], Thunderbird=[1967], Mustang=[1967]}`
3. The code does not compile.
4. The application throws an exception at runtime because the stream is not parallel.

26. What is the output of the following application?

```
package exercise;
import java.util.*;
public class Concat {
   public String concat1(List<String> values) {
      return values.parallelStream()
            .reduce("a",
                  (x,y)->x+y,
                  String::concat);
   }
   public String concat2(List<String> values) {
      return values.parallelStream()
            .reduce((w,z)->z+w).get();
   }
   public static void main(String... questions) {
      Concat c = new Concat();
      List<String> list = Arrays.asList("Cat","Hat");
      String x = c.concat1(list);
      String y = c.concat2(list);
      System.out.print(x+" "+y);
   }
}
```

1. `aCataHat HatCat`
2. `CatHat CatHat`
3. The code does not compile because `concat1()` returns an `Optional`.
4. The code does not compile for a different reason.

27. What is the output of the following application?

```
package taxes;
import java.util.concurrent.*;
public class Accountant {
   public static void completePaperwork() {
      System.out.print("[Filing]");
   }
   public static double getPi() {
      return 3.14159;
   }
   public static void main(String[] args) throws Exception {
      ExecutorService x = Executors.newSingleThreadExecutor();
      Future<?> f1 = x.submit(() -> completePaperwork());
      Future<Object> f2 = x.submit(() -> getPi());
      System.out.print(f1.get()+" "+f2.get());
```

```
      x.shutdown();
    }
  }
```

1. `[Filing]null 3.14159`
2. The declaration of `f1` does not compile.
3. The declaration of `f2` does not compile.
4. An exception is thrown at runtime.

28. Which statement about the following class is correct?

```
package my;
import java.util.*;
public class ThreadSafeList {
   private List<Integer> data = new ArrayList<>();
   public synchronized void addValue(int value) {
      data.add(value);
   }
   public int getValue(int index) {
      return data.get(index);
   }
   public int size() {
      synchronized(ThreadSafeList.class) {
         return data.size();
      }
   }
}
```

1. The code does not compile because of the `size()` method.
2. The code compiles and is thread-safe.
3. The code compiles and is not thread-safe.
4. The code does not compile for another reason.

29. Which two method names, when filled into the `print2()` method, produce the same output as the `print1()` method? Assume the input arguments for each represent the same non-`null` numeric value, only accessible by a single thread at a time.

```
public static void print1(int value) {
   System.out.println(value--);
   System.out.println(++value);
}
public static void print2(AtomicInteger value) {
   System.out.println(value._________);
   System.out.println(value._________);
}
```

1. `decrementAndGet()` and `getAndIncrement()`
2. `decrementAndGet()` and `incrementAndGet()`
3. `getAndDecrement()` and `getAndIncrement()`
4. `getAndDecrement()` and `incrementAndGet()`

30. How many times does the following application print `1` at runtime?

```
package crew;
import java.util.concurrent.*;
import java.util.stream.*;
public class Boat {
   private void waitTillFinished(CyclicBarrier c) {
      try {
         c.await();
         System.out.print("1");
      } catch (Exception e) {}
   }
   public void row(ExecutorService service) {
      final CyclicBarrier cb = new CyclicBarrier(5);
      IntStream.iterate(1, i-> i+1)
         .limit(12)
         .forEach(i -> service.submit(() -> waitTillFinished(cb)));
   }
   public static void main(String[] oars) {
      ExecutorService service = null;
      try {
         service = Executors.newCachedThreadPool();
         new Boat().row(service);
      } finally {
         service.isShutdown();
      }
   }
}
```

1. 0

2. 10
3. 12
4. None of the above

31. Using the `Boat` class from the previous question, what is the final state of the application?

1. The application produces an exception at runtime.
2. The application terminates successfully.
3. The application hangs indefinitely because the `ExecutorService` is never shut down.
4. The application produces a deadlock at runtime.

32. What is the expected output of the following application?

```
package store;
import java.util.concurrent.*;
import java.util.stream.*;
public class Line {
   static BlockingDeque<Integer> queue = new LinkedBlockingDeque<>(
   public static void main(String[] participants) throws Exception
      IntStream.iterate(1, i -> i+1).limit(5)
         .parallel()
         .forEach(s -> queue.offerLast(s,10000,TimeUnit.MILLISECOND
      IntStream.iterate(1, i -> 5).limit(10)
         .parallel()
         .forEach(s -> queue.pollFirst(10,TimeUnit.SECONDS));
      System.out.print(queue.size());
   }
}
```

1. `0`
2. A number from `0` to `5`
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

33. Given the `original` array, how many of the following `for` statements result in an infinite loop at runtime, assuming each is executed independently?

```
List<Integer> original = new ArrayList<>(Arrays.asList(1,2,3));

List<Integer> copy1 = new ArrayList<>(original);
for(Integer q : copy1)
   copy1.add(1);

List<Integer> copy2 = new CopyOnWriteArrayList<>(original);
for(Integer q : copy2)
   copy2.add(2);

Deque<Integer> copy3 = new ConcurrentLinkedDeque<>(original);
for(Integer q : copy3)
   copy3.push(3);
List<Integer> copy4 = Collections.synchronizedList(original);
for(Integer q : copy4)
   copy4.add(4);
```

1. Zero
2. One
3. Two
4. Three

34. Three of the four following options make up the requirements for performing a parallel reduction with the `collect()` method, which takes a `Collector` argument. Choose the one that is not a requirement.

1. The `Collector` argument is marked concurrent.
2. The elements of the stream implement the `Comparable` interface.
3. The stream is parallel.
4. The stream or `Collector` is marked unordered.

35. Which statement about the following application is true?

```
package math;
import java.util.concurrent.*;
public class Fun extends RecursiveTask<Integer> {
   final int value;
   public Fun(int value) {
      this.value = value;
   }
   @Override protected Integer compute() {  // w1
      if(value<1) {
         return 1;
      }
      final Fun f1 = new Fun(value-1);
      final Fun f2 = new Fun(value-2);
      return f1.compute() * f2.compute();
   }
   public static void main(String... data) {
      ForkJoinPool pool = new ForkJoinPool();
      try {
         System.out.print(pool.invoke(new Fun(10)));
      } finally {
         pool.shutdown();
      }
   }
}
```

1. The class does not compile due to line `w1`.
2. The class does not compile for another reason.
3. The application compiles and uses the fork/join framework correctly.
4. The application compiles but does not use the fork/join framework correctly.

36. Which `ExecutorService` method guarantees all running tasks are stopped in an orderly fashion?

1. `shutdown()`
2. `shutdownNow()`
3. `halt()`
4. None of the above

37. Given the following code snippet, what statement about the values printed on lines `p1` and `p2` is correct?

```
List<Integer> db = Collections.synchronizedList(new ArrayList<>());
IntStream.iterate(1, i -> i+1).limit(5)
   .parallel()
   .map(i -> {db.add(i); return i;})
   .forEachOrdered(System.out::print);  // p1
System.out.println();
db.forEach(System.out::print);  // p2
```

1. They are always the same.
2. They are sometimes the same.
3. They are never the same.
4. The code will produce a `ConcurrentModificationException` at runtime.

38. Assuming 10 seconds is enough time for all of the tasks to finish, what is the output of the following application?

```
package finance;
import java.util.concurrent.*;
public class Bank {
   static int cookies = 0;
   public synchronized void deposit(int amount) {
      cookies += amount;
   }
   public static synchronized void withdrawal(int amount) {
      cookies -= amount;
   }
   public static void main(String[] amount) throws Exception {
      ExecutorService teller = Executors.newScheduledThreadPool(50)
      Bank bank = new Bank();
      for(int i=0; i<25; i++) {
         teller.submit(() -> bank.deposit(5));
         teller.submit(() -> bank.withdrawal(5));
      }
      teller.shutdown();
      teller.awaitTermination(10, TimeUnit.SECONDS);
      System.out.print(bank.cookies);
   }
}
```

1. `0`
2. The code does not compile.
3. The result is unknown until runtime.
4. An exception is thrown at runtime.

39. What is the output of the following application?

```
package util;
import java.util.*;
public class SearchList<T> {
   private List<T> data;
   private boolean foundMatch = false;
   public SearchList(List<T> list) {
      this.data = list;
   }
   public void exists(T value,int start, int end) {
      if(end-start<=1) {
         foundMatch = foundMatch || value.equals(data.get(start));
      } else {
         final int middle = start + (end-start)/2;
         new Thread(() -> exists(value,start,middle)).run();
         new Thread(() -> exists(value,middle,end)).run();
      }
   }
   public static void main(String[] a) throws Exception {
      List<Integer> data = Arrays.asList(1,2,3,4,5,6);
      SearchList<Integer> t = new SearchList<Integer>(data);
      t.exists(5, 0, data.size());
      System.out.print(t.foundMatch);
   }
}
```

1. `true`
2. `false`
3. The code does not compile.
4. The result is unknown until runtime.

40. How many lines of the following code snippet contain compilation errors?

```
11: ScheduledExecutorService t = Executors
12:    .newSingleThreadScheduledExecutor();
13: Future result = t.execute(System.out::println);
14: t.invokeAll(null);
15: t.scheduleAtFixedRate(() -> {return;}, 5, TimeUnit.MINUTES);
```

1. None. The code compiles as is.
2. One
3. Two
4. Three

PREV
Chapter 19 Java File I/O (NIO.2)

NEXT
Chapter 21 Building Database Applications with JDBC