Java Concurrency - 0

Given code of Test.java fil

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

class MyThread implements Runnable {
    private String str;

    MyThread(String str) {
        this.str = str;
    }

    public void run() {
        System.out.println(str.toUpperCase());
    }
}

public class Test {
    public static void main(String[] args) throws
         ExecutionException, InterruptedException{
        ExecutorService es = Executors.newSingleThreadExecutor();
        MyThread thread = new MyThread("ocp");
        Future future = es.submit(thread);
        Integer tmp = (Integer) future.get(); //Line 22
        System.out.println(tmp);
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A - Compilation error is caused by Line 22

B - ClassCastException is thrown at runtime by Line 22

C -

```
ocp
null
```

D -

```
OCP
null
```

E -

```
OCP
OCP
```

Java Concurrency - 1

Given code of Test.java file:

```
import java.util.stream.IntStream;

public class Test {
    public static void main(String[] args) {
        IntStream.rangeClosed(1,
         10).parallel().forEachOrdered(System.out::println);
    }
}
```

What will be the result of compiling and executing Test class?

A - It will print numbers form 1 to 10 in descending order

B - It will print numbers from 1 to 10 in ascending order

C - It will print numbers form 1 to 10 but not in any specific order

Java Concurrency - 2

Given code of Test.java file:

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

class Printer implements Callable<String> {
    public String call() {
        System.out.println("DONE");
        return null;
    }
}

public class Test {
    public static void main(String[] args) {
        ExecutorService es = Executors.newFixedThreadPool(1);
        es.submit(new Printer());
        System.out.println("HELLO");
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A - HELLO and DONE will be printed but printing order is not fixed.

B - HELLO will always be printed before DONE.

C - DONE will always be printed before HELLO.

D - HELLO will never be printed.

Java Concurrency - 3

Given code of Test.java file:

```
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.*;

public class Test {
    public static void main(String[] args) throws
         InterruptedException, ExecutionException {
        Callable<String> c = new Callable<String>() {
            @Override
            public String call() throws Exception {
                try {
                    Thread.sleep(3000);
                } catch(InterruptedException e) {}
                return "HELLO";
            }
        };

        ExecutorService es = Executors.newFixedThreadPool(10);
        List<Callable<String>> list = Arrays.asList(c,c,c,c,c);
        List<Future<String>> futures = es.invokeAll(list);
        System.out.println(futures.size());
        es.shutdown();
    }
}
```

Which of the following statement is correct about above code?

A - Program will print 5 in any time greater than or equal to 3 secs.

B - Program will print 5 in less than 3 secs.

C - As each Thread sleeps for 3 secs, so program will take at least 15 secs. to print 5.

D - Program will always print 5 in exactly 3 secs.

Java Concurrency - 4

Given code of Test.java file:

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

class Player extends Thread {
    CyclicBarrier cb;

    public Player(){
        super();
    }

    public Player(CyclicBarrier cb) {
        this.cb = cb;
        this.start();
    }

    public void run() {
        try {
            cb.await();
        } catch (InterruptedException | BrokenBarrierException e)
         {}
    }
}

class Match implements Runnable {
    public void run() {
        System.out.println("Match is starting...");
    }
}

public class Test {
    public static void main(String[] args) {
        Match match = new Match();
        CyclicBarrier cb = new CyclicBarrier(2, match);
        Player p1 = new Player(cb);
        /*INSERT*/
    }
}
```

Which of the following statement, if used to replace /*INSERT*/, will print 'Match is starting…' on to the console and will successfully terminate the program?

A - cb.await();

B - new Player();

C - new Player(cb);

D - new Player(cb).start();

E - p1.start();

Java Concurrency - 5

Given code of Test.java file:

```
import java.util.concurrent.atomic.AtomicInteger;

class Counter implements Runnable {
    private static AtomicInteger ai = new AtomicInteger(3);

    public void run() {
        System.out.print(ai.getAndDecrement());
    }
}

public class Test {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Counter());
        Thread t2 = new Thread(new Counter());
        Thread t3 = new Thread(new Counter());
        Thread[] threads = {t1, t2, t3};
        for(Thread thread : threads) {
            thread.start();
        }
    }
}
```

What will be the result of compiling and executing Test class?

A - It will always print 210

B - None of the other options

C - It will always print 321

D - It will print three digits 321 but order can be different

E - It will print three digits 210 but order can be different

Java Concurrency - 6

Given code of Test.java file:

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

class Printer implements Runnable {
    public void run() {
        System.out.println("Printing");
    }
}

public class Test {
    public static void main(String[] args) {
        ExecutorService es = Executors.newFixedThreadPool(1);
        /*INSERT*/
        es.shutdown();
    }
}
```

Which of the following statements, if used to replace /*INSERT*/, will print ‘Printing’ on to the console? Select 2 options.

A - es.submit(new Printer());

B - es.run(new Printer());

C - es.start(new Printer());

D - es.execute(new Printer());

Java Concurrency - 7

Fill in the blanks:

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

public class Task extends _________________ {
    @Override
    protected Long compute() {
        return null;
    }
}
```

Select All that apply.

A - RecursiveAction

B - RecursiveAction<Object>

C - RecursiveTask<Object>

D - RecursiveAction<Long>

E - RecursiveTask<Long>

F - RecursiveTask

Java Concurrency - 8

Given code of Test.java file:

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

class MyCallable implements Callable<Integer> {
    private Integer i;

    public MyCallable(Integer i) {
        this.i = i;
    }

    public Integer call() throws Exception {
        return --i;
    }
}

public class Test {
    public static void main(String[] args) throws
         InterruptedException, ExecutionException {
        ExecutorService es = Executors.newSingleThreadExecutor();
        MyCallable callable = new MyCallable(100);
        System.out.println(es.submit(callable).get());
        System.out.println(es.submit(callable).get());
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A -

```
100
99
```

B -

```
99
99
```

C -

```
99
98
```

D -

```
100
100
```

E -

98

Java Concurrency - 9

Given code of Test.java file:

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

public class Test {
    public static void main(String[] args) {
        List<String> list1 = new ArrayList<>();
        list1.add("Melon");
        list1.add("Apple");
        list1.add("Banana");
        list1.add("Mango");
        List<String> list2 = new CopyOnWriteArrayList<>(list1);
        for(String s : list2) {
            if(s.startsWith("M")){
                list2.remove(s);
            }
        }
        System.out.println(list1);
        System.out.println(list2);
    }
}
```

What will be the result of compiling and executing Test class?

A -

```
[Melon, Apple, Banana, Mango]
[Apple, Banana]
```

B -

```
[Melon, Apple, Banana, Mango]
[Melon, Apple, Banana, Mango]
```

C - An exception is thrown at runtime

D -

```
[Apple, Banana]
[Apple, Banana]
```

Java Concurrency - 10

Given code of Test.java file:

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;

class Accumulator {
    private List<Integer> list = new ArrayList<>();

    public synchronized void accumulate(int i) {
        list.add(i);
    }

    public List<Integer> getList() {
        return list;
    }
}

public class Test {
    public static void main(String [] args) {
        ExecutorService s = Executors.newFixedThreadPool(1000);
        Accumulator a = new Accumulator();
        for(int i=1; i<=1000; i++) {
            int x = i;
            s.execute(() -> a.accumulate(x));
        }
        s.shutdown();
        System.out.println(a.getList().size());
    }
}
```

What will be the result of compiling and executing Test class?

A - It can print any number between 0 and 1000

B - The program will wait indefinitely

C - It will always print 1000 on to the console

Java Concurrency - 11

Fill in the blanks:

The states of the threads involved in _____ constantly change with regard to one another, with no overall progress made.

A - synchronization

B - CyclicBarrier

C - deadlock

D - livelock

Java Concurrency - 12

Given code of Test.java file:

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

class Player extends Thread {
    CyclicBarrier cb;

    public Player(CyclicBarrier cb) {
        this.cb = cb;
    }

    public void run() {
        try {
            cb.await();
        } catch (InterruptedException | BrokenBarrierException e)
        {}
    }
}

class Match implements Runnable {
    public void run() {
        System.out.println("Match is starting...");
    }
}

public class Test {
    public static void main(String[] args) {
        Match match = new Match();
        CyclicBarrier cb = new CyclicBarrier(2, match);
        ExecutorService es = Executors.newFixedThreadPool(1);
        es.execute(new Player(cb));
        es.execute(new Player(cb));
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A - “Match is starting…” is printed on to the console and program terminates successfully.

B - “Match is starting…” is printed on to the console and program doesn’t terminate.

C - “Match is Starting…” is never printed on to the console and program waits indefinitely.

Java Concurrency - 13

Given code of Test.java file:

```
class Counter implements Runnable {
    private static int i = 3;

    public void run() {
        System.out.print(i--);
    }
}

public class Test {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Counter());
        Thread t2 = new Thread(new Counter());
        Thread t3 = new Thread(new Counter());
        Thread[] threads = {t1, t2, t3};
        for(Thread thread : threads) {
            thread.start();
        }
    }
}
```

What will be the result of compiling and executing Test class?

A - It will always print 321

B - It will print three digits 210 but order can be different

C - It will always print 210

D - It will print three digits 321 but order can be different

E - None of the other options

Java Concurrency - 14

Given code of Test.java file:

```
import java.util.stream.IntStream;

public class Test {
    public static void main(String[] args) {
        int res = IntStream.rangeClosed(1, 1000).parallel()
                .filter(i -> i > 50).findFirst().getAsInt();
        System.out.println(res);
    }
}
```

What will be the result of compiling and executing Test class?

A - It will print any number between 1 and 50

B - It will always print 51

C - It will print any number between 51 and 1000

D - It will always print 50

Java Concurrency - 15

Given code of Test.java file:

```
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

public class Test {
    public static void main(String[] args) {
        List<String> list = new CopyOnWriteArrayList<>();
        list.add("Melon");
        list.add("Apple");
        list.add("Banana");
        list.add("Mango");
        for(String s : list) {
            list.removeIf(str -> str.startsWith("M"));
            System.out.println(s);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A - Compilation error

B -

```
Apple
Banana
```

C - An exception is thrown at runtime

D -

```
Melon
Apple
Banana
Mango
```

Java Concurrency - 16

Given code of Test.java file:

```
import java.util.stream.IntStream;

public class Test {
    public static void main(String[] args) {
        IntStream.rangeClosed(1,
         10).parallel().forEach(System.out::println);
    }
}
```

What will be the result of compiling and executing Test class?

A - It will print numbers form 1 to 10 in descending order

B - It will print numbers form 1 to 10 but not in any specific order

C - It will print numbers from 1 to 10 in ascending order

Java Concurrency - 17

Given code of Test.java file:

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

class Adder extends RecursiveAction {
    private int from;
    private int to;
    int total = 0;

    Adder(int from, int to) {
        this.from = from;
        this.to = to;
    }

    @Override
    protected void compute() {
        if ((to - from) <= 4) {
            int sum = 0;
            for(int i = from; i <= to; i++) {
                sum += i;
            }
            total+=sum;
        } else {
            int mid = (from + to) / 2;
            Adder first = new Adder(from, mid);
            Adder second = new Adder(mid + 1, to);
            invokeAll(first, second);
        }
    }
}

public class Test {
    public static void main(String[] args) {
        Adder adder = new Adder(1, 5); //Line 34
        ForkJoinPool pool = new ForkJoinPool(4);
        pool.invoke(adder);
        System.out.println(adder.total);
    }
}
```

What will be the result of compiling and executing Test class?

A - None of the other options

B - It will print 0 on to the console

C - It will print 15 on to the console

D - It can print any number between 0 and 15

Java Concurrency - 18

Given code of Test.java file:

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

class MyCallable implements Callable<Integer> {
    private Integer i;

    public MyCallable(Integer i) {
        this.i = i;
    }

    public Integer call() throws Exception {
        return --i;
    }
}

public class Test {
    public static void main(String[] args) {
        ExecutorService es = Executors.newSingleThreadExecutor();
        MyCallable callable = new MyCallable(1);
        System.out.println(es.submit(callable).get());
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A - 0

B - Compilation error

C - 1

D - An exception is thrown at runtime

Java Concurrency - 19

Given code of Test.java file:

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

public class Test {
    public static void main(String[] args) throws
        InterruptedException, ExecutionException {
        ExecutorService es = Executors.newSingleThreadExecutor();
        Future<String> f = es.submit(() -> "HELLO");
        System.out.println(f.get());
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A - An exception is thrown at runtime

B - null

C - HELLO

D - Compilation error

Java Concurrency - 20

Given code of Test.java file:

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Test {
    private static void print() {
        System.out.println("PRINT");
    }

    private static Integer get() {
        return 10;
    }

    public static void main(String [] args) throws
         InterruptedException, ExecutionException {
        ExecutorService es = Executors.newFixedThreadPool(10);
        Future<?> future1 = es.submit(Test::print);
        Future<?> future2 = es.submit(Test::get);
        System.out.println(future1.get());
        System.out.println(future2.get());
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A -

```
PRINT
10
null
```

B -

```
PRINT
null
10
```

C -

```
null
10
PRINT
```

D -

```
null
PRINT
```

```
10
```

E - Compilation error

Java Concurrency - 21

Performance with parallel stream is always better than sequential streams.

A - false

B - true

Java Concurrency - 22

Can all streams be converted to parallel stream?

A - Yes

B - No

Java Concurrency - 23

Given code of Test.java file:

```
import java.util.Arrays;
import java.util.List;

public class Test {
    private static StringBuilder RES = new StringBuilder();

    public static void main(String[] args) {
        List<String> list = Arrays.asList("A", "B", "C", "D", "E",
        "F",
                                                          "G", "H",
        "I", "J");
        list.parallelStream().forEach(RES::append);
        System.out.println(RES);
    }
}
```

What will be the result of compiling and executing Test class?

A - It will always print ABCDEFGHIJ

B - Compilation error

C - Output cannot be predicted

Java Concurrency - 24

Given code of Test.java file:

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

public class Test {
    public static void main(String[] args) throws
        InterruptedException, ExecutionException {
        ExecutorService es = Executors.newSingleThreadExecutor();
        Future<String> f = es.execute(() -> "HELLO");
        System.out.println(f.get());
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A - HELLO

B - Compilation error

C - null

D - An exception is thrown at runtime

Java Concurrency - 25

Given code of Test.java file:

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

class Caller implements Callable<Void> {
    String str;

    public Caller(String s) {
        this.str = s;
    }

    public Void call() throws Exception {
        System.out.println(str.toUpperCase());
        return null;
    }
}

public class Test {
    public static void main(String[] args) throws
         InterruptedException,

         ExecutionException {
        ExecutorService es = Executors.newSingleThreadExecutor();
        Future<Void> future = es.submit(new Caller("Call"));
        System.out.println(future.get());
    }
}
```

What will be the result of compiling and executing Test class?

A - The program doesn’t terminate but prints following:

```
CALL
CALL
```

B - The program doesn’t terminate but prints following:

```
CALL
null
```

C - The program doesn’t terminate but prints following:

```
null
null
```

D - The program terminates after printing:

```
CALL
null
```

Java Concurrency - 26

Which of the following instances can be passed to invoke() method of ForkJoinPool class?

Select ALL that apply.

A - ForkJoinTask

B - RecursiveAction

C - RecursiveTask

D - Callable

E - Runnable

Java Concurrency - 27

Given code of Test.java file:

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

class Adder extends RecursiveAction {
    private int from;
    private int to;
    int total = 0;

    Adder(int from, int to) {
        this.from = from;
        this.to = to;
    }

    @Override
    protected void compute() {
        if ((to - from) <= 4) {
            int sum = 0;
            for(int i = from; i <= to; i++) {
                sum += i;
            }
            total+=sum;
        } else {
            int mid = (from + to) / 2;
            Adder first = new Adder(from, mid);
            Adder second = new Adder(mid + 1, to);
            invokeAll(first, second);
        }
    }
}

public class Test {
    public static void main(String[] args) {
        Adder adder = new Adder(1, 20); //Line 34
        ForkJoinPool pool = new ForkJoinPool(4);
        pool.invoke(adder);
        System.out.println(adder.total);
    }
}
```

What will be the result of compiling and executing Test class?

A - None of the other options.

B - It will print 210 on to the console.

C - It will print 0 on to the console.

D - It can print any number between 0 and 210.

Java Concurrency - 28

Given code of Test.java file:

```
import java.util.stream.IntStream;

public class Test {
    public static void main(String[] args) {
        IntStream stream = IntStream.rangeClosed(1, 5);
        System.out.println(stream.parallel().reduce((x, y) -> x +
         y).getAsInt());
    }
}
```

What will be the result of compiling and executing Test class?

A - It will print 15 on to the console

B - It can print any number between 1 and 15

C - None of the other options

D - It will print 0 on to the console

Java Concurrency - 29

Given code of Test.java file:

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.atomic.AtomicInteger;

public class Test {
    public static void main(String [] args) throws
         InterruptedException, ExecutionException{
        AtomicInteger ai = new AtomicInteger(10);
        /*INSERT*/
    }
}
```

Which of the following statements, if used to replace /*INSERT*/, will print '11:11' on to the console?

Select ALL that apply.

A - System.out.println(ai.incrementAndGet(1) + ":" + ai.get());

B - System.out.println(ai.addAndGet(1) + ":" + ai);

C - System.out.println(ai.incrementAndGet() + ":" + ai.get());

D - System.out.println(ai.getAndIncrement() + ":" + ai.get());

E - System.out.println(ai.getAndAdd(1) + ":" + ai.get());

Java Concurrency - 30

Given code of Test.java file:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.stream.IntStream;

public class Test {
    public static void main(String[] args) {
        List<Integer> list = Collections.synchronizedList(new
         ArrayList<>());
        IntStream stream = IntStream.rangeClosed(1, 7);
        stream.parallel().map(x -> {
            list.add(x); //Line 13
            return x;
        }).forEach(System.out::print); //Line 15
        System.out.println();
        list.forEach(System.out::print); //Line 17
    }
}
```

Which of the following statement is true about above code?

A - Line 15 and Line 17 will not print exact same output on to the console

B - Line 15 and Line 17 will print exact same output on to the console

C - Output cannot be predicted

Java Concurrency - 31

Given code of Test.java file:

```
import java.util.Arrays;

public class Test {
    public static void main(String[] args) {
        String s1 = Arrays.asList("A", "E", "I", "O",
         "U").stream().reduce("_", String::concat);
        String s2 = Arrays.asList("A", "E", "I", "O",
         "U").parallelStream().reduce("_", String::concat);
        System.out.println(s1.equals(s2));
    }
}
```

What will be the result of compiling and executing Test class?

A - It will always print false

B - Output cannot be predicted

C - It will always print true

Java Concurrency - 32

Given code of Test.java file:

```
import java.util.stream.Stream;

public class Test {
    public static void main(String[] args) {
        String str1 = Stream.iterate(1, i -> i + 1).limit(10)
                .reduce("", (i, s) -> i + s, (s1, s2) -> s1 + s2);
        String str2 = Stream.iterate(1, i -> i +
         1).limit(10).parallel()
                .reduce("", (i, s) -> i + s, (s1, s2) -> s1 + s2);
        System.out.println(str1.equals(str2));
    }
}
```

What will be the result of compiling and executing Test class?

A - It will always print true

B - Output cannot be predicted

C - It will always print false

Java Concurrency - 33

Given code of Test.java file:

```
import java.util.Arrays;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("A", "E", "I", "O", "U");
        System.out.println(list._________________.isParallel());
    }
}
```

Which of the options correctly fills the blank, such that output is true? Select ALL that apply.

A - parallel()

B - stream().parallel()

C - stream()

D - parallelStream( )

Java Concurrency - 34

Given code of Test.java file:

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

class MyCallable implements Callable<Integer> {
    private Integer i;

    public MyCallable(Integer i) {
        this.i = i;
    }

    public Integer call() throws Exception {
        return --i;
    }
}

class MyThread extends Thread {
    private int i;

    MyThread(int i) {
        this.i = i;
    }

    public void run() {
        i++;
    }
}

public class Test {
    public static void main(String[] args) throws
         ExecutionException, InterruptedException{
        ExecutorService es = Executors.newSingleThreadExecutor();
        MyCallable callable = new MyCallable(10);
        MyThread thread = new MyThread(10);
        System.out.println(es.submit(callable).get());
        System.out.println(es.submit(thread).get());
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A -

```
9
0
```

B -

```
9
11
```

C -

9
10

D -

9
null

Java Concurrency - 35

Given code of Adder.java file:

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class Adder {
    private static long LIMIT = 1000000000;
    private static final int THREADS = 100;

    static class AdderTask extends RecursiveTask<Long> {
        long from, to;

        public AdderTask(long from, long to) {
            this.from = from;
            this.to = to;
        }

        @Override
        protected Long compute() {
            if ((to - from) <= LIMIT/THREADS) {
                long localSum = 0;
                for(long i = from; i <= to; i++) {
                    localSum += i;
                }
                return localSum;
            }
            else {
                long mid = (from + to) / 2;
                AdderTask first = new AdderTask(from, mid);
                AdderTask second = new AdderTask(mid + 1, to);
                first.fork();
                /*INSERT*/
            }
        }
    }

    public static void main(String[] args) {
        ForkJoinPool pool = new ForkJoinPool(THREADS);
        long sum = pool.invoke(new AdderTask(1, LIMIT));
        System.out.printf("sum of the number from %d to %d is %d
        %n", 1, LIMIT, sum);
    }
}
```

Which of the following statement, if used to replace /*INSERT*/, will EFFICIENTLY add the numbers from 1 to 1000000000?

A - return second.compute();

B - return second.compute() + first.join();

C - return first.join();

D - return first.join() + second.compute();

Java Concurrency - 36

Given code of Test.java file:

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

public class Test {
    public static void main(String[] args) {
        ExecutorService es = Executors.newSingleThreadExecutor();
        es.execute(() -> System.out.println("HELLO"));
        es.shutdown();
    }
}
```

What will be the result of compiling and executing Test class?

A - An exception is thrown at runtime

B - Compilation error

C - null

D - HELLO

Java Concurrency - 37

Given code of Test.java file:

```
import java.util.stream.Stream;

public class Test {
    public static void main(String[] args) {
        Stream<String> stream = Stream.of("J", "A", "V", "A");
        String text =
         stream.parallel().reduce(String::concat).get();
        System.out.println(text);
    }
}
```

What will be the result of compiling and executing Test class?

A - None of the other options

B - Output cannot be predicted

C - It will always print JAVA on to the console

Java Concurrency - 38

Given code of Test.java file:

```
import java.util.ArrayList;
import java.util.List;

class Book {
    String isbn;
    double price;

    Book(String isbn, double price) {
        this.isbn = isbn;
        this.price = price;
    }

    public String toString() {
        return "Book[" + isbn + ":" + price + "]";
    }
}

public class Test {
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        books.add(new Book("9781976704031", 9.99));
        books.add(new Book("9781976704032", 15.99));

        Book b = books.stream().reduce(new Book("9781976704033",
         0.0), (b1, b2) -> {
            b1.price = b1.price + b2.price;
            return new Book(b1.isbn, b1.price);
        });

        books.add(b);
        books.parallelStream().reduce((x, y) -> x.price > y.price ?
         x : y).ifPresent(System.out::println);
    }
}
```

What will be the result of compiling and executing Test class?

A - Book[9781976704033:9.99]

B - Book[9781976704032:15.99]

C - Book[9781976704031:9.99]

D - Book[9781976704033:25.98]

E - Book[9781976704033:15.99]

Java Concurrency - 39

To efficiently use fork/join framework, after invoking fork() on first subtask, the order of invoking join() and compute() is as follows:

A - Invoke join() on 2nd subtask and then compute() on 1st subtask

B - Invoke join() on 1st subtask and then compute() on 2nd subtask

C - Invoke compute() on 1st subtask and then join() on 2nd subtask

D - Invoke compute() on 2nd subtask and then join() on 1st subtask

Java Concurrency - 40

Given code of Test.java file:

```
import java.util.Arrays;

public class Test {
    public static void main(String[] args) {
        String s1 = Arrays.asList("A", "E", "I", "O",
         "U").stream().reduce("", String::concat);
        String s2 = Arrays.asList("A", "E", "I", "O",
         "U").parallelStream().reduce("", String::concat);
        System.out.println(s1.equals(s2));
    }
}
```

What will be the result of compiling and executing Test class?

A - It will always print true

B - Output cannot be predicted

C - It will always print false

Java Concurrency - 41

Given code of Test.java file:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.stream.IntStream;

public class Test {
    public static void main(String[] args) {
        List<Integer> list = Collections.synchronizedList(new
         ArrayList<>());
        IntStream stream = IntStream.rangeClosed(1, 7);
        stream.parallel().map(x -> {
            list.add(x); //Line 13
            return x;
        }).forEachOrdered(System.out::print); //Line 15
        System.out.println();
        list.forEach(System.out::print); //Line 17
    }
}
```

Which of the following statement is true about above code?

A - Line 15 and Line 17 will not print exact same output on to the console

B - Output of both Line 15 and Line 17 can be predicted

C - Output of Line 17 can be predicted

D - Line 15 and Line 17 will print exact same output on to the console

E - Output of Line 15 can be predicted

Java Concurrency - 42

Which of the below classes help you to define recursive task?

Select ALL that apply.

A - RecursionTask

B - RecursionAction

C - RecursiveTask

D - Recursion

E - RecursiveAction