Started on	Monday, 28 April 2025, 3:16 PM
State	Finished
Completed on	Monday, 28 April 2025, 3:26 PM
Time taken	9 mins 59 secs
Marks	8.00/10.00
Grade	**80.00** out of 100.00

Question 1

Complete

Mark 1.00 out of 1.00

In the following code, how many threads are created?

```
public class Test {
    public static void main(String[] args) {
        ExecutorService service = Executors.newFixedThreadPool(5);
        for (int i = 0; i < 10; i++) {
            service.submit(() -> System.out.print(Thread.currentThread().getName() + " "));
        }
        service.shutdown();
    }
}
```

a. 10

b. 15

c. 5

d. Depends on JVM

Question 2

Complete

Mark 0.00 out of 1.00

What is wrong with this Runnable usage?

```
Runnable r = () -> {
    Thread.sleep(1000);
    System.out.println("Done");
};
new Thread(r).start();
```

a. Compile-time error due to missing return

b. No problem

c. Multiple threads will be created

d. Thread.sleep must be inside try-catch block

Question 3

Complete

Mark 1.00 out of 1.00

What happens in this code?

```
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<String> future = executor.submit(() -> {
    throw new RuntimeException("Error occurred!");
});
future.get();
```

a. No Exception is thrown

b. ExecutionException is thrown when get() is called

c. Thread terminates silently

d. Future returns null

Question 4

Complete

Mark 1.00 out of 1.00

```
ExecutorService service = Executors.newFixedThreadPool(2);
Future<Integer> future1 = service.submit(() -> 1);
Future<Integer> future2 = service.submit(() -> 2);
System.out.print(future1.isDone() + " " + future2.isDone());
service.shutdown();
```

At the point of printing, what is most likely?

a. Any combination depending on timing

b. true true

c. false false

d. true false

Question 5

Complete

Mark 0.00 out of 1.00

In this code, what type does the Future hold?

```
Future<?> future = executor.submit(() -> System.out.println("Task"));
```

a. Future<Void>

b. Future<String>

c. Future<Integer>

d. Future<Object>

Question 6

Complete

Mark 1.00 out of 1.00

What will be the output?

```
class MyCallable implements Callable<String> {
    public String call() {
        return "Callable";
    }
}
public class Test {
    public static void main(String[] args) throws Exception {
        FutureTask<String> task = new FutureTask<>(new MyCallable());
        new Thread(task).start();
        System.out.print(task.get());
    }
}
```

a. null

b. Runtime exception

c. Callable

d. Compile-time error

Question 7

Complete

Mark 1.00 out of 1.00

What happens for the following code?

```
Callable<String> c = () -> {
    Thread.sleep(2000);
    return "Result";
};
ExecutorService executor = Executors.newFixedThreadPool(1);
Future<String> future = executor.submit(c);
System.out.print(future.get(1, TimeUnit.SECONDS));
```

a. "Result" will be printed

b. TimeoutException

c. ExecutionException

d. IllegalStateException

Question 8

Complete

Mark 1.00 out of 1.00

What will be the output of the following code?

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.print("Runnable");
    }
}
public class Test {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

a. Runnable

b. No output

c. Runtime exception

d. Compile-time error

Question 9

Complete

Mark 1.00 out of 1.00

What will happen when the following code is executed?

```
class MyCallable implements Callable<Integer> {
    public Integer call() {
        return 100;
    }
}
public class Test {
    public static void main(String[] args) {
        ExecutorService service = Executors.newSingleThreadExecutor();
        Future<Integer> future = service.submit(new MyCallable());
        service.shutdown();
    }
}
```

- a. call() will still be executed even without future.get().
- b. Compile-time error.
- c. call() will throw an exception.
- d. call() will never be executed because get() is missing.

Question 10

Complete

Mark 1.00 out of 1.00

Identify the problem in the following code:

```
class MyTask implements Runnable {
    public String run() {
        return "Hello";
    }
}
```

- a. Infinite loop
- b. Valid code
- c. Runtime exception
- d. Compile-time error because Runnable.run() must return void