Q1. To return different values for multiple invocations of the same stubbed method, I am considering three instruction:

a)when(mockDatabaseImpl.getGrade(anyInt())).thenReturn("A","B", "C");

b)when(mockDatabaseImpl.getGrade(anyInt())).thenReturn("A").thenReturn("B").thenReturn("C");

c)doReturn("A").doReturn("B").doReturn("C").when(mockDatabaseImpl).getGrade(anyInt())

Which of the above one(s) will work

Options

- ○ Only a will work
- ◉ Only a and c will work
- ○ all a,b and c will work
- ○ only c will work

Time Remaining: 02:54:26 Submit Assess

Q2. We have multiple criteria while working with JUnit. Among the following criterias the most exhaustive one which requires that inside each decision, all combinations of conditions are tested is:

Options

- Redundant Condition/decision coverage
- Multiple condition coverage
- Modified Condition/decision coverage
- Lossless Condition/decision coverage

Q3. What Logstash will do if we run the bellow code? input {

```
 file {
   path => "/tmp/*_log"
 }
}

filter {
 if [path] =~ "access" {
   mutate { replace => { type => "apache_access" } }
   grok {
     match => { "message" => "%{COMBINEDAPACHELOG}" }
   }
   date {
     match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
   }
 } else if [path] =~ "error" {
   mutate { replace => { type => "apache_error" } }
 } else {
   mutate { replace => { type => "random_logs" } }
 }
}
```

```
    }
    date {
      match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
  } else if [path] =~ "error" {
    mutate { replace => { type => "apache_error" } }
  } else {
    mutate { replace => { type => "random_logs" } }
  }
 }

output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

Options

- Labels all events using the type field, but doesn't actually parse the error
- Labels all events using the type field, and parse the error
- Labels all events using the type field, and parse the random files
- Log all events using the type field, and parse the random files

Mark for Review | Prev | Next

Q4. Below are two git commands

A. Merge

B. Rebase

Below are descriptions for above git commands in randome order.

1. Combines all changes of a different branch into the current.

2. Produces a new generated commit

3. Re-comitting all commits of the current branch onto a different base commit.

4. Only moves existing commits

Find option where descriptions are arranged correctly.

Options

- A-1, B-2, A-3, B-4
- A-2, B-1, A-3, B-4
- A-1, A-2, B-3, B-4
- B-1, B-2, A-3, A-4

Q5. If you want to make radical changes to your team's project and don't want to impact the rest of the team, you should implement your changes in

Options

- ○ a tag.
- ○ the root.
- ○ the trunk.
- ◉ a branch

Q6. Mockito provides two different syntaxes for creating stubs like:

doReturn and thenReturn

Both these methods setup stubs and can be used to create/setup stubs and could be used interchangeably at times.But they do differ in their behavior.

Assume a method getItemDetails on mockedItemService which returns an object of type ItemSku.

I am writing following 4 code instructions for mocking.

1)when(mockedItemService.getItemDetails(123)).thenReturn(new ItemSku());

2)when(mockedItemService.getItemDetails(123)).thenReturn(expectedPrice);

3)doReturn(expectedPrice).when(mockedItemService.getItemDetails(123));

4)doReturn(new ItemSku()).when(mockedItemService.getItemDetails(123));

What is the possible behavior of each one?

Options

- 1)compiles successfully
 2)compiles successfully
 3)compiles successfully
 4)compiles successfully
- 1)compiles successfully
 2)throw compile time exception
 3)compiles successfully
 4)compiles successfully
- 1)compiles successfully
 2)throw compile time exception
 3)throw compile time exception
 4)compiles successfully
- 1)compiles successfully
 2)compiles successfully
 3)throw compile time exception
 4)compiles successfully
