**Given this VSCode settings.json snippet, which configuration enables Copilot inline suggestions only in Markdown files?**

**text**

**{**

**"editor.inlineSuggest.enabled": false,**

**"github.copilot.inlineSuggest.enabled": true**

**}**

**Pick ONE option**

1. Add a markdown-specific section enabling Copilot inline suggestions without altering global settings

"[markdown]": {

"github.copilot.inlineSuggest.enabled": true

}

1. **Disable Copilot inline suggestions globally and enable them only for Markdown files**

**"github.copilot.inlineSuggest.enabled": false,**

**"[markdown]": {**

**"github.copilot.inlineSuggest.enabled": true**

**}**

1. Use the built-in editor inline suggestion setting for Markdown while leaving Copilot settings unchanged

"[markdown]": {

"editor.inlineSuggest.enabled": true

}

This question asks how to configure VSCode so that GitHub Copilot inline suggestions appear **only in Markdown files**. It shows three possible settings.json configurations and asks you to pick the correct one.

**Explanation of Given Snippet**

The snippet shown in the question:

{

"editor.inlineSuggest.enabled": false,

"github.copilot.inlineSuggest.enabled": true

}

* Disables VSCode native inline suggestions globally.
* Enables Copilot's inline suggestions globally.

But the question wants inline Copilot suggestions **only for Markdown files**, not globally.

**Reviewing the Options**

**Option 1**

"[markdown]": {

"github.copilot.inlineSuggest.enabled": true

}

* This tries to enable Copilot inline suggestions only for Markdown, using a language-specific section.
* This is close, but Copilot settings sometimes require the "github.copilot.enable" key instead of "github.copilot.inlineSuggest.enabled".
* Moreover, if the global setting is not disabled, Copilot suggestions may still show in other files.

**Option 2**

**json**

**"github.copilot.inlineSuggest.enabled": false,**

**"[markdown]": {**

**"github.copilot.inlineSuggest.enabled": true**

**}**

* **Disables Copilot inline suggestions globally.**
* **Enables Copilot inline suggestions only for Markdown by overriding in a language-specific section.**
* **This matches the question requirement: "enable them only for Markdown files".**

**Option 3**

json

"[markdown]": {

"editor.inlineSuggest.enabled": true

}

* Enables VSCode's native inline suggestions (not Copilot) only for Markdown files.
* Does not affect Copilot settings, so Copilot might not be enabled for Markdown specifically.

**Correct and Incorrect Answers**

**Correct Option**

* **Option 2** is correct because it disables Copilot inline suggestions everywhere, then enables them specifically for Markdown files using the [markdown] language override.

**Incorrect Options**

* **Option 1** may not fully disable Copilot suggestions in other languages unless the global setting is set to false. Also, the correct setting to use is often "github.copilot.enable" rather than "github.copilot.inlineSuggest.enabled" in recent Copilot versions.
* **Option 3** only changes VSCode's own inline suggestion behavior, not Copilot's; it will not enable Copilot for Markdown.

**Summary Table**

| **Option** | **Effect** | **Correct?** |
| --- | --- | --- |
| Option 1 | Enables Copilot only for Markdown (may not be enough) | ✗ |
| Option 2 | Disables Copilot globally, enables for Markdown | **✓ (Correct)** |
| Option 3 | Enables VSCode native inline suggestions for Markdown | ✗ |

**Option 2** best matches the requirement: "enable Copilot inline suggestions only in Markdown files".

**Markdown files** are plain text files that use special, lightweight syntax to format text—such as headings, lists, bold, italics, links, code, and images—without requiring HTML or a rich text editor.

**What Markdown Files Are**

* Markdown files typically have the extension .md or .markdown.
* They use characters like # for headings, \* or \_ for bold/italic, and brackets for links.
* The syntax is designed to be readable as plain text, but can be easily converted to formatted HTML or other formats by Markdown processors.

**Common Uses**

* Writing README files and documentation in coding projects (especially on GitHub).
* Creating notes, web pages, technical documentation, blogs, and even eBooks.
* Markdown is popular for collaboration, as it works across many platforms and is easy to edit.

**Example**

A sample Markdown file content:

text

# My Project

This is a \*\*Markdown\*\* file.

- Bullet 1

- Bullet 2

Visit [GitHub](https://github.com).

When processed, this renders as a formatted document with a heading, bold text, a list, and a hyperlink.

In summary, Markdown files are a simple, portable way to create formatted documents using plain text syntax.

**Q2:**

**A developer is creating a user registration REST API and wants Copilot to suggest edge cases to test the endpoint. Which comment would most likely prompt Copilot to generate relevant edge cases?**

**Pick ONE option**

* **Add a comment explicitly asking for edge cases in user registration  
  // Suggest edge cases to test user registration API**
* Add a comment requesting Copilot to generate the entire user registration class  
  // Generate user registration class
* Add a comment instructing Copilot to generate a REST API endpoint for user registration  
  // Generate REST API endpoint for user registration
* Add a comment asking Copilot to write async/await code for user registration  
  // Write async/await code for user registration

The correct answer is:  
**// Suggest edge cases to test user registration API**

**Explanation**

* This comment directly and explicitly asks Copilot to provide **edge cases** related to testing a user registration REST API.
* Copilot interprets comments contextually. The more specific and focused the comment is regarding your intent (in this case, generating edge test cases), the more likely Copilot will generate output that matches your needs.

**Why the Other Options Are Incorrect**

* **// Generate user registration class**
  + Prompts Copilot to generate the code for a user registration class, not edge cases or tests for it.
* **// Generate REST API endpoint for user registration**
  + Tells Copilot to write the API code, but does not request any edge cases or test scenarios.
* **// Write async/await code for user registration**
  + Instructs Copilot to implement asynchronous code, unrelated to listing or suggesting test cases for the endpoint.

**Summary Table**

| **Option** | **Outcome** | **Correct?** |
| --- | --- | --- |
| // Suggest edge cases to test user registration API | Generates edge cases for registration API | **✓** |
| // Generate user registration class | Generates user registration class code | ✗ |
| // Generate REST API endpoint for user registration | Generates REST endpoint code, not test cases | ✗ |
| // Write async/await code for user registration | Generates async code, not edge case tests | ✗ |

The first option is correct because it matches the developer's need: getting edge test cases for the API endpoint.

**Question 3:**

**Scan logs are redirected to a file scan.log using ProcessBuilder.redirectOutput. Each new scan overwrites the existing log. Determine which redirect method call should replace redirectOutput(Redirect.to(file)) so that new log entries are appended rather than overwriting the file.**

**Pick ONE option**

* **Use the appendTo(File) redirect method  
  redirectOutput(ProcessBuilder.Redirect.appendTo(file));**
* Use the to(File) redirect with absolute file path  
  redirectOutput(ProcessBuilder.Redirect.to(file.getAbsoluteFile()));
* Use the appendTo(Path) redirect method  
  redirectOutput(ProcessBuilder.Redirect.appendTo(file.toPath()));
* Use the to(FileDescriptor) redirect from an append-mode FileOutputStream  
  redirectOutput(ProcessBuilder.Redirect.to(new FileOutputStream(file, true).getFD()));

The correct answer is:  
**Use the appendTo(File) redirect method**  
redirectOutput(ProcessBuilder.Redirect.appendTo(file));

**Explanation**

* The question wants the existing log file to be **appended** with new entries instead of being overwritten.
* In ProcessBuilder.Redirect, the method appendTo(File) is explicitly designed to redirect output to the file with appending behavior.

**Why the Other Options Are Incorrect**

* **Use the to(File) redirect with absolute file path**  
  redirectOutput(ProcessBuilder.Redirect.to(file.getAbsoluteFile()));  
  This method redirects output to the file but **overwrites** it each time. It does not append.
* **Use the appendTo(Path) redirect method**  
  redirectOutput(ProcessBuilder.Redirect.appendTo(file.toPath()));  
  This is actually a valid method to append output, too, but it needs a Path argument (converted using file.toPath()). If the method signature expects a File, this might not be directly applicable. However, it is conceptually similar to the correct answer.
* **Use the to(FileDescriptor) redirect from an append-mode FileOutputStream**  
  redirectOutput(ProcessBuilder.Redirect.to(new FileOutputStream(file, true).getFD()));  
  While this low-level approach could work, it is more cumbersome and less idiomatic than using appendTo. The question asks which method to replace redirectOutput(Redirect.to(file)), so this is likely not the expected answer.

| **Option** | **Effect** | **Correct?** |
| --- | --- | --- |
| appendTo(File) | Appends output to file | **✓** |
| to(File.getAbsoluteFile()) | Overwrites file | ✗ |
| appendTo(Path) | Appends output (for Path) | Valid but depends on signature context; less common |
| to(FileDescriptor) with append mode FileOutputStream | Complex, less idiomatic | ✗ |

**Summary Table**

In conclusion, the simplest and most direct way to append logs is to use ProcessBuilder.Redirect.appendTo(file).

**Question 4:**

A Java method contains multiple nested if-else statements. SonarQube is configured to flag methods with high cyclomatic complexity. Which aspect of the following code will be highlighted?

text

public void process(int x, int y) {

if (x > 0) {

if (y > 0) {

// do something

} else {

if (x < 10) {

// do something else

} else {

// another branch

}

}

} else {

// fallback

}

}

Pick ONE option

* The method’s cyclomatic complexity due to multiple nested conditional branches  
  (Shows code with nested if-else statements)
* The use of primitive data types for method parameters  
  (Shows code with int parameters)
* The absence of exception handling within the method  
  (Shows code with no try-catch block)

**The correct answer is:  
The method’s cyclomatic complexity due to multiple nested conditional branches**

Explanation:

* Cyclomatic complexity measures the number of linearly independent paths through a program's source code, often increased by conditional branches like nested if-else statements.
* The given code has multiple nested if-else blocks, which significantly increase cyclomatic complexity.
* SonarQube flags methods with high cyclomatic complexity as potentially hard to maintain or test, so it would highlight this aspect.

Why the Other Options Are Incorrect

* The use of primitive data types for method parameters
  + Using primitives like int is normal and not a complexity or maintainability issue flagged by SonarQube for cyclomatic complexity.
* The absence of exception handling within the method
  + While lack of try-catch blocks might be a concern in some contexts, this code’s main complex trait SonarQube detects is cyclomatic complexity, not exception handling absence.

**Summary Table:**

| **Option** | **Reason for correctness** | **Correct?** |
| --- | --- | --- |
| **Cyclomatic complexity from nested conditionals** | **Increases decision points, flagged by SonarQube** | **✓** |
| **Use of primitive data types for parameters** | **Not relevant to cyclomatic complexity** | **✗** |
| **Absence of exception handling** | **Not related to cyclomatic complexity** | **✗** |

The nested conditional structure in the method is exactly what SonarQube's cyclomatic complexity rule checks for.

**Question 5:**

**How do you scale a Kubernetes deployment?**

**kubectl scale deployment myapp --replicas=3**

Pick ONE option

* **kubectl scale deployment**
* kubectl resize deployment
* kubectl expand deployment
* kubectl grow deployment

The correct answer is:  
**kubectl scale deployment**

**Explanation**

* The kubectl scale command is specifically designed to change the number of replicas in a Kubernetes deployment or other scalable resources.
* Example usage: kubectl scale deployment myapp --replicas=3 sets the deployment "myapp" to run 3 replicas, effectively scaling it up or down.

**Why the Other Options Are Incorrect**

* **kubectl resize deployment**
  + There is no resize command in the official kubectl CLI. Kubernetes uses scale for adjusting replicas.
* **kubectl expand deployment**
  + No such command in kubectl; "expand" is not a valid scaling command.
* **kubectl grow deployment**
  + "grow" is not a recognized kubectl command either; scaling is done via scale.

**Summary Table:**

| **Option** | **Reason** | **Correct?** |
| --- | --- | --- |
| kubectl scale deployment | Official command to scale replicas | ✓ |
| kubectl resize deployment | Not a valid kubectl command | ✗ |
| kubectl expand deployment | Not a valid kubectl command | ✗ |
| kubectl grow deployment | Not a valid kubectl command | ✗ |

In Kubernetes, scaling deployments is universally done using the kubectl scale command.

**Question 6:**

A Java method contains multiple nested if-else statements. SonarQube is configured to flag methods with high cyclomatic complexity. Which aspect of the following code will be highlighted?

public void process(int x, int y) {

if (x > 0) {

if (y > 0) {

// do something

} else {

if (x < 10) {

// do something else

} else {

// another branch

}

}

} else {

// fallback

}

}

**Pick ONE option**

* The method’s cyclomatic complexity due to multiple nested conditional branches  
  (Shows code with nested if-else statements)
* The use of primitive data types for method parameters  
  (Shows code with int parameters)
* The absence of exception handling within the method  
  (Shows code with no try-catch block)

This contains the full question and all answer choices as shown in your image.

**Question 6:**

A PaaS offering advertises that infrastructure maintenance tasks such as OS patching, network configuration, and runtime updates are abstracted away from developers. The team focuses solely on application logic. Which operational responsibility is fully offloaded to the PaaS provider?

Pick ONE option

* **Patch management for the hosted operating system and runtime environment.**
* Deployment and versioning of application code.
* Configuration of application-level environment variables.
* Definition of database schema and migrations.

**The correct answer is:  
Patch management for the hosted operating system and runtime environment.**

Explanation

* In a Platform as a Service (PaaS) model, the provider manages and offloads infrastructure-level responsibilities including patching the OS, applying runtime updates, and managing network configurations. This frees developers to focus on writing and deploying application logic without worrying about the underlying platform maintenance.
* The question states that infrastructure maintenance tasks like OS patching are abstracted away, which directly points to patch management being offloaded to the PaaS provider.

Why Other Options Are Incorrect

* Deployment and versioning of application code
  + This responsibility typically lies with the development team or DevOps; PaaS provides tools but does not fully offload deployment/version management.
* Configuration of application-level environment variables
  + Developers or DevOps teams usually manage environment variables to configure applications at runtime, not the PaaS provider.
* Definition of database schema and migrations
  + Schema design and migrations are application-level concerns managed by developers or database engineers, not the PaaS provider.

| **Option** | **Responsibility Location** | **Correct?** |
| --- | --- | --- |
| **Patch management for OS and runtime environment** | **Fully managed by PaaS provider** | **✓** |
| Deployment and versioning of application code | Managed by development or DevOps teams | ✗ |
| Configuration of application-level environment vars | Managed by developers or DevOps | ✗ |
| Definition of database schema and migrations | Managed by developers or DBAs | ✗ |

**Summary Table**

**Thus, patch management of OS and runtime is the operational responsibility fully offloaded to the PaaS provider according to the question context.**

**Question:**

**A Java class contains two overloaded methods with the same name but different parameters. The code coverage tool shows one method fully covered and the other not covered. What does this indicate about the test cases?**

**public class Calculator {**

**public int add(int a, int b) {**

**return a + b;**

**}**

**public double add(double a, double b) {**

**return a + b;**

**}**

**}**

***// Code coverage tool reports:***

***// add(int, int) - 100% covered***

***// add(double, double) - 0% covered***

**Pick ONE option**

* **Test cases only invoke the method with integer parameters.  
  Calculator calc = new Calculator();  
  int result = calc.add(2, 3);**
* Test cases only invoke the method with double parameters.  
  Calculator calc = new Calculator();  
  double result = calc.add(2.0, 3.0);
* Test cases involve both overloaded methods with appropriate parameters.  
  Calculator calc = new Calculator();  
  int result = calc.add(2, 3);  
  double result2 = calc.add(2.0, 3.0);
* Test cases do not invoke any of the overloaded methods.

**The correct answer is:  
Test cases only invoke the method with integer parameters.**

**java**

**Calculator calc = new Calculator();**

**int result = calc.add(2, 3);**

Explanation

* The code coverage tool shows that the add(int, int) method is 100% covered, but the add(double, double) method is 0% covered.
* This means the tests are only calling and exercising the integer version of the add method, but never the double version.

Why the Other Options Are Incorrect

* Test cases only invoke the method with double parameters.
  + If this were true, the coverage for add(double, double) would be 100%, not 0%.
* Test cases invoke both overloaded methods with appropriate parameters.
  + If both were invoked, both would have some coverage, so the double version would not be 0%.
* Test cases do not invoke any of the overloaded methods.
  + This would mean both versions have 0% coverage, which contradicts the 100% coverage for the integer add method.

**Summary Table:**

**So, the coverage report clearly indicates only the integer overload is tested. The correct answer is: Test cases only invoke the method with integer parameters.**

**Calculator calc = new Calculator();**

**int result = calc.add(2, 3);**

Explanation

* The coverage tool shows 100% coverage for add(int, int) and 0% for add(double, double), indicating only the integer method is tested.
* If both overloaded methods were tested, both would have some coverage, not zero for either.
* If only the double method was called, its coverage would be 100%, not zero.

Why Others are Incorrect

* Test cases only invoke the method with double parameters — coverage for double would be 100%, not 0%.
* Test cases invoke both overloaded methods — both methods would show coverage, so double would not be 0%.
* Test cases do not invoke any methods — integer method would not be 100% covered.

This coverage pattern means only the integer-parameter method is exercised by tests.

**Question:**

public class Feature {

public void run() {

*// Profile-based inclusion*

if (System.getProperty("feature.enabled").equals("true")) {

System.out.println("Feature enabled");

} else {

System.out.println("Feature disabled");

}

}

}

Pick ONE option

* Coverage tools like JaCoCo ignore conditional compilation and report 100% coverage for all code.s

public class Feature {

public void run() {

*// JaCoCo reports 100% regardless of actual execution*

if (System.getProperty("feature.enabled").equals("true")) {

System.out.println("Feature enabled");

}

}

}

* **Coverage metrics are calculated based on the union of all possible build profiles.**

public class Feature {

public void run() {

*// Coverage reflects all code from all profiles*

if (System.getProperty("feature.enabled").equals("true")) {

System.out.println("Feature enabled");

} else {

System.out.println("Feature disabled");

}

}

}

**The correct answer is:  
Coverage metrics are calculated based on the union of all possible build profiles.**

**Explanation**

* **Coverage tools like JaCoCo analyze which parts of the code are executed during tests. When the code contains conditional logic based on build profiles (e.g., feature flags or system properties), coverage metrics are calculated from all possible executions across these profiles.**
* **This means JaCoCo reports coverage reflecting code across all possible build profiles collectively, so uncovered code in one profile might be covered in another.**
* **The provided example shows if-else logic dependent on a system property, so coverage includes both branches assuming tests cover different profiles.**

**Why the Other Option Is Incorrect**

* **JaCoCo ignores conditional compilation and reports 100% regardless of actual execution**
  + **This is false; JaCoCo measures actual execution during tests. It does not blindly report 100% coverage. It reports based on code paths exercised, including conditions and branches.**

| **Option** | **Correctness** | **Reason** |
| --- | --- | --- |
| **Coverage metrics calculated across union of all build profiles** | **✓** | **Reflects combined coverage over all profiles** |
| **JaCoCo ignores conditionals and reports 100% coverage** | **✗** | **JaCoCo reports actual code execution coverage** |

**Summary Table:**

**Thus, code coverage metrics reflect the union of all code run across different build configurations or profiles.**

**Question:**

**Which of the following annotations can be used to inject mock attributes into the test object automatically?**

**Pick ONE option**

* **@Inject**
* **@MockInjects**
* **@InjectMocks**
* **@Mocks**

**The correct answer is:  
@InjectMocks**

**Explanation**

* **@InjectMocks is used in Mockito to automatically inject mock or spy objects into the tested object. It injects mocks annotated with @Mock or created manually into the fields of the class under test, simplifying test setup.**

**Why the Other Options Are Incorrect**

* **@Inject**
  + **This is a standard Java CDI (Contexts and Dependency Injection) annotation and is not specifically used by Mockito for injecting mocks in tests.**
* **@MockInjects**
  + **This annotation does not exist in Mockito or standard Java testing frameworks; likely a non-existent or incorrect name.**
* **@Mocks**
  + **The correct annotation for creating mock objects is @Mock, not @Mocks. Also, @Mock only creates mocks but does not inject them automatically into the test object.**

**Summary Table**

| **Annotation** | **Purpose** | **Correct?** |
| --- | --- | --- |
| **@Inject** | **CDI dependency injection (not Mockito specific)** | **✗** |
| **@MockInjects** | **Non-existent/incorrect annotation** | **✗** |
| **@InjectMocks** | **Automatically injects mock objects into test object** | **✓** |
| **@Mocks** | **Incorrect; Mockito uses @Mock to create mock objects** | **✗** |

**Hence, @InjectMocks is the correct annotation to automatically inject mock attributes into the test object.**

**Question :**

**Consider the following code block**

**text**

**import org.junit.jupiter.api.BeforeEach;**

**import org.junit.jupiter.api.Test;**

**import java.util.ArrayList;**

**import java.util.List;**

**import static org.junit.jupiter.api.Assertions.assertEquals;**

**import static org.mockito.Mockito.\*;**

public class JUnitWhenMultipleUsage {

private List<String> nameList;

@BeforeEach

public void init() {

nameList = mock(ArrayList.class);

}

@Test

public void testWithMockedArrayList() {

// ...

}

}

Pick ONE option

* X = 41  
  Y = 41  
  Z = 41  
  T = atLeastOnce()
* X = 41  
  Y = 41  
  Z = 41  
  T = times(3)
* **X = 10  
  Y = 20  
  Z = 30  
  T = times(3)**
* X = 41  
  Y = 41  
  Z = 41  
  T = atLeast(3)

**This I have explained in class with real time exampless**

**Question:**

**You have the following scenario:**

* **main branch has commits A-B-C**
* **feature branch has commits D-E  
  You want to integrate the "feature" branch into main, but you prefer to keep a linear commit history. What is the appropriate command?**

Pick ONE option

* git checkout main  
  git merge feature
* git checkout feature  
  git rebase main
* **git checkout main  
  git rebase feature**
* git merge --no-ff feature

**The correct answer is:  
git checkout main  
git rebase feature**

**Explanation**

* **To keep a linear commit history when integrating a feature branch (D-E) into the main branch (A-B-C), rebasing the main branch on top of the feature branch commits is the right approach.**
* **git checkout main switches to the main branch.**
* **git rebase feature moves the main branch commits (A-B-C) on top of the feature branch commits (D-E) so the commit history appears linear without merge commits.**

**Why Other Options Are Incorrect**

* **git checkout main; git merge feature**
  + **This creates a merge commit, resulting in a non-linear history with branches converging, not preserving a linear commit history.**
* **git checkout feature; git rebase main**
  + **This rebases the feature branch onto main; but the question asks to integrate into main and keep the history linear from main's perspective. Rebasing main on feature is the right method, not the other way around.**
* **git merge --no-ff feature**
  + **This forces a merge commit even if a fast-forward is possible, leading to a non-linear history, which contradicts the requirement to keep a linear commit history.**

**Summary Table:**

| **Command** | **Effect** | **Correct?** |
| --- | --- | --- |
| **git checkout main; git merge feature** | **Creates merge commit, non-linear history** | **✗** |
| **git checkout feature; git rebase main** | **Rebases feature onto main (not integration into main)** | **✗** |
| **git checkout main; git rebase feature** | **Rebases main onto feature, linear history** | **✓** |
| **git merge --no-ff feature** | **Forces merge commit, non-linear history** | **✗** |

**Thus, the best way to keep a linear commit history while integrating is to rebase the main branch on top of the feature branch.**

**Question:**

**The Transactions table contains transaction\_id, account\_id, and amount.  
You want to find accounts with a total transaction amount exceeding 10,000.  
Evaluate this query:**

Which variation is valid?

Pick ONE option

* SELECT account\_id, SUM(amount)  
  FROM Transactions  
  GROUP BY account\_id  
  WHERE total\_amount > 10000;
* **SELECT account\_id, SUM(amount) AS total\_amount  
  FROM Transactions  
  GROUP BY account\_id  
  HAVING SUM(amount) > 10000;**
* SELECT account\_id, SUM(amount)  
  FROM Transactions  
  GROUP BY account\_id  
  WHERE SUM(amount) > 10000;
* SELECT account\_id, SUM(amount)  
  FROM Transactions  
  GROUP BY account\_id  
  HAVING total\_amount > 10000;

**The correct answer is:  
SELECT account\_id, SUM(amount) AS total\_amount  
FROM Transactions  
GROUP BY account\_id  
HAVING SUM(amount) > 10000;**

**Explanation**

* **The SQL clause HAVING is used to filter groups created by the GROUP BY clause, based on aggregate functions like SUM().**
* **You cannot use WHERE to filter on aggregate functions because WHERE filters rows before grouping happens.**
* **The query correctly uses HAVING SUM(amount) > 10000 to filter accounts whose total transaction amount exceeds 10,000.**

**Why Other Options Are Incorrect**

* **Option 1:  
  Uses WHERE total\_amount > 10000 — invalid since total\_amount is an alias created in the SELECT clause and aggregates cannot be filtered in WHERE.**
* **Option 3:  
  Uses WHERE SUM(amount) > 10000 — invalid; aggregate functions can only be used in HAVING, not WHERE.**
* **Option 4:  
  Uses HAVING total\_amount > 10000 — invalid; total\_amount alias cannot be used directly in HAVING, the aggregate function must be repeated.**

**Summary Table**

| **Option** | **Correctness** | **Reason** |
| --- | --- | --- |
| **GROUP BY + WHERE total\_amount > 10000** | **✗** | **Cannot use WHERE to filter aggregates** |
| **GROUP BY + HAVING SUM(amount) > 10000** | **✓** | **Correct syntax for filtering aggregation** |
| **GROUP BY + WHERE SUM(amount) > 10000** | **✗** | **Aggregate function not allowed in WHERE** |
| **GROUP BY + HAVING total\_amount > 10000** | **✗** | **Alias not allowed in HAVING, must repeat func** |

**Thus, the second option is the valid SQL query to find accounts with total transaction amounts exceeding 10,000.**

**Question:**

**Which expression is not correct?**

Pick ONE option

* The primary key uniquely identifies each record in the table.
* The primary key must contain unique values.
* **The primary key can contain NULL values, but this is highly undesirable for performance reasons.**
* A table can have only one primary key.

**The primary key can contain NULL values, but this is highly undesirable for performance reasons.**

**Explanation**

* **In relational databases, a primary key cannot contain NULL values; it must uniquely identify each record, and NULLs are not allowed in primary keys by definition.**
* **This statement is incorrect because allowing NULL values in a primary key violates database integrity rules.**

**Why Other Options Are Correct**

* **The primary key uniquely identifies each record in the table.**
  + **This is true; the main purpose of a primary key is to uniquely identify rows.**
* **The primary key must contain unique values.**
  + **Correct; primary key values must be unique across the table.**
* **A table can have only one primary key.**
  + **Correct; a table can have only one primary key constraint, although that primary key can be composed of multiple columns (composite key).**

**Summary Table**

| **Statement** | **Correctness** | **Reason** |
| --- | --- | --- |
| **Primary key uniquely identifies each record** | **Correct** | **Defines uniqueness for rows** |
| **Primary key must contain unique values** | **Correct** | **Primary keys enforce uniqueness** |
| **Primary key can contain NULL values (undesirable)** | **Incorrect** | **NULLs not allowed in primary keys** |
| **A table can have only one primary key** | **Correct** | **Only one primary key constraint per table** |

**Therefore, the statement about primary keys containing NULL values is incorrect.**

**Question:**

**Select the extraneous expression (not the same as the others).  
Select all that apply.**

**Pick ONE option**

* SELECT \* FROM categories ORDER BY id
* SELECT \* FROM categories ORDER BY id ASC
* **SELECT \* FROM categories ORDER BY id DESC**
* None of the above, all expressions are the same.

**The correct answer is:  
SELECT \* FROM categories ORDER BY id DESC**

**Explanation**

* **The question asks for the "extraneous" expression—not the same as the others.**
* **These are the options:**
  + **SELECT \* FROM categories ORDER BY id**
  + **SELECT \* FROM categories ORDER BY id ASC**
  + **SELECT \* FROM categories ORDER BY id DESC**
  + **None of the above, all expressions are the same.**
* **In SQL, ORDER BY id and ORDER BY id ASC are equivalent—both sort the results in ascending order by id.**
* **ORDER BY id DESC sorts in descending order by id. This is different from the other two.**

**Why Others Are Incorrect**

* **SELECT \* FROM categories ORDER BY id: Ascending order, same as ORDER BY id ASC.**
* **SELECT \* FROM categories ORDER BY id ASC: Ascending order, same as above.**
* **None of the above, all expressions are the same: Incorrect, because ORDER BY id DESC produces a different ordering from the other two.**

**Summary Table**

| **Expression** | **Order produced** | **Same as others?** | **Extraneous?** |
| --- | --- | --- | --- |
| **SELECT \* FROM categories ORDER BY id** | **Ascending** | **Yes** | **No** |
| **SELECT \* FROM categories ORDER BY id ASC** | **Ascending** | **Yes** | **No** |
| **SELECT \* FROM categories ORDER BY id DESC** | **Descending** | **No** | **Yes** |
| **None of the above...** | **(not an expression)** | **-** | **No** |

**The extraneous expression is:  
SELECT \* FROM categories ORDER BY id DESC  
because it sorts the results in descending order, making it different from the others.**

**When multiple instances of a Spring Boot microservice are running, which mechanism is commonly used to distribute incoming requests evenly among them to ensure high availability and scalability?**

Pick ONE option

* Using a load balancer to route requests to available service instances
* Storing requests in a shared database for manual retrieval by service instances
* Directly connecting clients to each microservice instance without routing
* Using a message queue to sequentially deliver requests to service instances

**The correct answer is:  
Using a load balancer to route requests to available service instances**

**Explanation**

* **When multiple instances of a Spring Boot microservice are running, a load balancer is typically employed to distribute incoming requests evenly across the available instances.**
* **This ensures high availability by redirecting requests to healthy instances and enhances scalability by balancing the load.**

**Why Other Options Are Incorrect**

* **Storing requests in a shared database for manual retrieval by service instances**
  + **This approach is inefficient and does not ensure real-time load distribution or automatic failover.**
* **Directly connecting clients to each microservice instance without routing**
  + **This lacks scalability and fault tolerance since clients must be aware of all instances manually.**
* **Using a message queue to sequentially deliver requests to service instances**
  + **While useful for asynchronous processing, this does not evenly distribute synchronous incoming requests and can create bottlenecks.**

**Summary Table**

| **Option** | **Correctness** | **Reason** |
| --- | --- | --- |
| **Using load balancer to route requests** | **Correct** | **Ensures even distribution, scalability** |
| **Storing requests in shared DB for manual retrieval** | **Incorrect** | **Inefficient, non-real-time** |
| **Directly connecting clients without routing** | **Incorrect** | **Non-scalable, no fault tolerance** |
| **Using message queue for sequential requests** | **Incorrect** | **Not suitable for load balancing** |

**Hence, the load balancer is the common and effective mechanism for distributing requests across microservice instances.The correct answer is:  
Using a load balancer to route requests to available service instances**

**Explanation**

* **When multiple instances of a Spring Boot microservice are running, a load balancer is used to evenly distribute incoming requests to ensure the system remains highly available and scalable.**
* **This mechanism dynamically routes requests to healthy service instances and balances the workload, preventing any single instance from being overwhelmed.**

**Why Other Options Are Incorrect**

* **Storing requests in a shared database for manual retrieval by service instances**
  + **Inefficient for load balancing, introduces latency and complexity, and does not automatically distribute requests.**
* **Directly connecting clients to each microservice instance without routing**
  + **Not scalable or fault tolerant, as clients must manage connection details and failover manually.**
* **Using a message queue to sequentially deliver requests to service instances**
  + **Suitable for asynchronous processing rather than balanced synchronous request routing.**

**Summary Table**

| **Option** | **Correctness** | **Reason** |
| --- | --- | --- |
| **Using a load balancer to route requests** | **Correct** | **Ensures high availability and scalability** |
| **Storing requests in shared DB for manual retrieval** | **Incorrect** | **Inefficient and manual** |
| **Directly connecting clients without routing** | **Incorrect** | **Not scalable or fault tolerant** |
| **Using message queue for sequential request delivery** | **Incorrect** | **Used for asynchronous processing** |

**Using a load balancer is the standard method for request distribution in microservices architecture.**

**Question:**

**Which communication pattern is most suitable for synchronous data retrieval between two Spring Boot microservices when immediate response is required?**

Pick ONE option

* Utilizing event-driven architecture with event publishing and subscribing
* **Using RESTful HTTP calls for direct synchronous communication**
* Implementing asynchronous messaging with a message broker
* Applying batch processing with scheduled data synchronization

**The correct answer is:  
Using RESTful HTTP calls for direct synchronous communication**

**Explanation**

* **Synchronous data retrieval with immediate response requires direct, blocking communication between services, which is typically achieved via RESTful HTTP calls.**
* **RESTful APIs facilitate request-response interactions where the client waits for the server to process the request and send back a response immediately.**

**Why Other Options Are Incorrect**

* **Utilizing event-driven architecture with event publishing and subscribing**
  + **This is asynchronous and decoupled, not suitable for immediate synchronous responses.**
* **Implementing asynchronous messaging with a message broker**
  + **Also asynchronous; message brokers do not provide immediate request-response communication.**
* **Applying batch processing with scheduled data synchronization**
  + **Batch processing operates on delayed schedules and is not for real-time synchronous communication.**

**Summary Table**

| **Option** | **Correctness** | **Reason** |
| --- | --- | --- |
| **RESTful HTTP calls for synchronous communication** | **Correct** | **Supports immediate, blocking request-response** |
| **Event-driven architecture with publishing/subscribing** | **Incorrect** | **Asynchronous pattern** |
| **Asynchronous messaging with message broker** | **Incorrect** | **Non-blocking, delayed response** |
| **Batch processing with scheduled synchronization** | **Incorrect** | **Not real-time, delayed operation** |

**Thus, RESTful HTTP calls are most suitable for synchronous immediate responses between Spring Boot microservices.**

**Question:**

**Why is API versioning important in a Spring Boot microservices architecture when exposing RESTful endpoints to clients?**

Pick ONE option

* It ensures that all microservices use the same database schema.
* It enables automatic scaling of microservices without manual intervention.
* **It allows backward compatibility for clients when APIs evolve over time.**
* It eliminates the need for authentication between microservices.

**The correct answer is:  
It allows backward compatibility for clients when APIs evolve over time.**

**Explanation**

* **API versioning is crucial in microservices to support backward compatibility, so existing client applications continue to function as the API evolves or new versions are released.**
* **This allows developers to expose multiple versions of an API, giving clients time to migrate to updated endpoints.**

**Why Other Options Are Incorrect**

* **It ensures that all microservices use the same database schema.**
  + **API versioning does not enforce or require shared database schemas; each microservice can have its own schema.**
* **It enables automatic scaling of microservices without manual intervention.**
  + **Scaling is managed via orchestration tools or infrastructure (e.g., Kubernetes, load balancers), not by API versioning.**
* **It eliminates the need for authentication between microservices.**
  + **Authentication is a separate concern; versioning does not affect authentication requirements.**

**Summary Table**

| **Option** | **Correctness** | **Reason** |
| --- | --- | --- |
| **Enables backward compatibility for evolving APIs** | **Correct** | **Addressed by versioning** |
| **Ensures same database schema for all microservices** | **Incorrect** | **Unrelated to API versioning** |
| **Enables auto scaling of microservices** | **Incorrect** | **Not achieved via versioning** |
| **Eliminates authentication between microservices** | **Incorrect** | **Authentication unrelated to versioning** |

**API versioning helps support old clients as APIs change, ensuring smoother upgrades and service continuity.**

**Given a Spring Cloud Gateway application that must terminate HTTPS connections and forward requests to backend microservices over HTTP. Identify the necessary configuration properties to enable SSL termination and specify the keystore details. Assume the keystore file is named gateway.p12 with password 'changeit'.**

**Pick ONE option**

* Configure server.ssl properties with keystore file and password in application.yml to enable HTTPS termination at the gateway.

text

server:

port: 443

ssl:

enabled: true

key-store: classpath:gateway.p12

key-store-password: changeit

key-store-type: PKCS12

* Set spring.cloud.gateway.ssl properties with keystore details to enable SSL termination at the gateway.

text

spring:

cloud:

gateway:

ssl:

key-store: classpath:gateway.p12

key-store-password: changeit

key-store-type: PKCS12

* Configure spring.security.ssl properties with keystore file and password to enable HTTPS termination in the gateway.

text

spring:

security:

ssl:

key-store: classpath:gateway.p12

key-store-password: changeit

key-store-type: PKCS12

**Configure server.ssl properties with keystore file and password in application.yml to enable HTTPS termination at the gateway.**

**text**

**server:**

**port: 443**

**ssl:**

**enabled: true**

**key-store: classpath:gateway.p12**

**key-store-password: changeit**

**key-store-type: PKCS12**

**Explanation**

* **To enable HTTPS termination in a Spring Cloud Gateway application, the recommended approach is to configure SSL parameters under server.ssl in application.yml, specifying the port (443 for HTTPS), keystore location, password, and type.**
* **This configuration instructs the embedded web server (such as Tomcat or Netty, which runs the gateway) to terminate HTTPS connections using the provided keystore.**

**Why Other Options Are Incorrect**

* **Set spring.cloud.gateway.ssl properties with keystore details**
  + **This is not a standard property recognized by Spring Cloud Gateway for HTTPS termination. SSL termination is handled by the web server svia server.ssl, not these properties.**
* **Configure spring.security.ssl properties for gateway**
  + **spring.security.ssl is not a valid property for configuring SSL termination. Spring Security does not configure the embedded web server's HTTPS settings this way.**

**Summary Table**

| **Option** | **Correctness** | **Reason** |
| --- | --- | --- |
| **Configure server.ssl in application.yml** | **Correct** | **Standard approach for HTTPS termination in Spring Boot applications** |
| **Set spring.cloud.gateway.ssl** | **Incorrect** | **Not recognized by Spring Cloud Gateway for terminating HTTPS** |
| **Configure spring.security.ssl** | **Incorrect** | **Not used for web server HTTPS termination** |

**Configuring server.ssl properties is the standard and correct way to enable HTTPS termination at the gateway.**

**Question:**

**Which of the following statements describe the properties of a microservice?**

**Pick ONE or MORE options**

* Focuses on a single responsibility principle
* Too large and complex to fully understand and make changes fast and correctly
* Responsible for its own task and communicates with other services through simple APIs to solve a larger complex business problem
* Should implement a fully testable end-to-end business use-case

**The correct answers are:**

* **Focuses on a single responsibility principle**
* **Responsible for its own task and communicates with other services through simple APIs to solve a larger complex business problem**

**Explanation**

* **Microservices are designed to follow the single responsibility principle, meaning each service focuses on a specific, well-defined function or business capability.**
* **Each microservice is responsible for its own task and typically communicates with other services using simple APIs (e.g., RESTful APIs) to collectively solve broader, more complex business problems.**
* **These principles help microservices to be loosely coupled, independently deployable, and maintainable.**

**Why Other Options Are Incorrect**

* **Too large and complex to fully understand and make changes fast and correctly**
  + **This describes monolithic applications, not microservices. Microservices aim to be small and manageable for faster changes.**
* **Should implement a fully testable end-to-end business use-case**
  + **While end-to-end testability might be a goal at the system integration level, a microservice usually handles a part of the business logic and not the entire end-to-end use case on its own.**

**Summary Table:**

| **Statement** | **Correct?** | **Reason** |
| --- | --- | --- |
| **Focuses on a single responsibility principle** | **Yes** | **Fundamental microservice design principle** |
| **Responsible for own task and communicates via simple APIs** | **Yes** | **Key microservice property** |
| **Too large and complex for fast changes and understanding** | **No** | **Describes monoliths, opposite of microservices** |
| **Should implement fully testable end-to-end use case** | **No** | **Microservices are part of end-to-end, not whole** |

**Hence, microservices focus on single responsibilities and simple API communication for modular, scalable architectures.**

**A company deals has implemented the GPT-3 AI Text Generation using microservices infrastructure for handling thousands of users. The operations team receives a notification that three of their services are down. There is evidence of image pullback failure. Upon investigating, they deduced that one of their services was internet facing and was exploited. The exploited service started messaging other services with corrupted data. The team applied security patches to fix the problem.**

**What is the fix?**

**Pick ONE option**

* **The team added mTLS and added rules for inter-service communications. After that, they pushed the internet-facing server behind a load balancer and applied JWT tokens for authentication.**
* **The team cut off the internet-facing service from the infrastructure and applied a JWT token for each call to other services.**
* **A and B**

**The correct answer is:  
A and B**

**Explanation**

* **The most robust security fix for the scenario described involves both:**
  + **Adding mutual TLS (mTLS) and authentication/authorization (JWT tokens) for inter-service communication to ensure only trusted services communicate with each other and that every call is authenticated.**
  + **Cutting off internet-facing services and applying security controls at the gateway/load balancer, further reducing the attack surface and requiring JWT tokens for every service call.**
* **Combining these approaches results in stronger defense-in-depth for microservices, preventing exploitations through network isolation, mutual trust, and authenticated requests.**

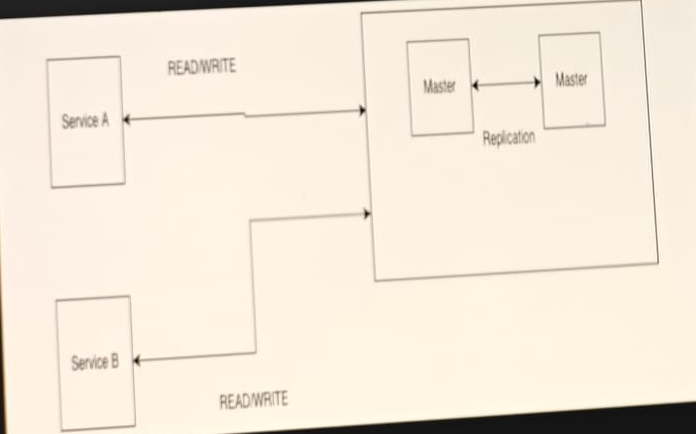
**Breakdown of Options**

* **Option A:**
  + **Implements mTLS for service-to-service security and moves public-facing components behind a load balancer with JWT authentication—industry best practice for zero trust security in microservices.**
* **Option B:**
  + **Removes vulnerable services from direct internet exposure and requires JWT authentication for every inter-service call, blocking malicious access and data corruption.**
* **A and B:**
  + **Combining both strategies provides comprehensive protection, addressing the root cause and strengthening system resilience.**

**Why Only A or B Alone Is Less Correct**

* **Relying solely on one solution may leave gaps: only authentication (JWT) doesn't prevent all network-level attacks, and only mTLS doesn't enforce API-level authorization on each call.**
* **A combination is required for robust microservice security.**

**Therefore, "A and B" is correct, as both are recommended security fixes in a compromised microservice architecture scenario.**

****

**Consider the diagram below:**

**Diagram with Service A, Service B, and replication between Master databases; both services have READ/WRITE access.**

**Pick ONE option**

* **increased latency due to synchronization**
* **volatile consistency due to lag in write**
* **The more read slaves are added to the system, the more you have to replicate, which leads to greater replication lag.**
* **potential loss of data if one of the masters fails**
* **All of the above.**

**The correct answer is:  
All of the above.**

**Explanation**

* **Master-master database replication with multiple services performing READ/WRITE can lead to several issues:**
  + **Increased latency due to synchronization: Each master must communicate changes to the other, causing delays.**
  + **Volatile consistency due to lag in write: Writes may not instantly synchronize, leading to temporary inconsistent states between masters (replication lag).**
  + **Replication lag increases with more read slaves: More replicas mean more data to synchronize, so lag and performance overhead grow as the system scales.**
  + **Potential loss of data if one of the masters fails: If a master fails before replicating changes, those changes can be lost, risking data integrity.**

**Summary Table**

| **Risk/Issue** | **Applicability** | **Reason** |
| --- | --- | --- |
| **Increased latency due to synchronization** | **Yes** | **Replication/sync overhead** |
| **Volatile consistency due to lag in write** | **Yes** | **Changes may not be immediately reflected across all masters** |
| **Greater replication lag with more read slaves** | **Yes** | **Replication increases lag and complexity as system grows** |
| **Potential loss of data if a master fails** | **Yes** | **Non-replicated data may be lost during failures** |

**All options describe valid problems with master-master replication in distributed systems architectures.**

**Question:s**

**In a RESTful application using Spring, which of the following is not used to create an object validator?**

**Pick ONE option**

* **@Min(value = 1)  
  @Max(999999)  
  private int id;**
* **@Size(limit = 100)  
  private String name;**
* **@NotNull  
  private Boolean isActive;**
* **@ValidCategory(categoryType="sample")  
  private String category;**

**The correct answer is:  
@ValidCategory(categoryType="sample")**

**Explanation**

* **@Min, @Max, @Size, and @NotNull are standard validation annotations provided by Java Bean Validation (JSR-380, implemented as Hibernate Validator in Spring), which are used to create object validators in Spring RESTful applications.**
* **@ValidCategory(categoryType="sample") is not a standard validator annotation in Spring or Java Bean Validation; it appears to be a custom or non-existent annotation, so it is not generally used for object validation.**

| **Annotation** | **Spring/Bean Validator?** | **Used for object validation?** |
| --- | --- | --- |
| **@Min/@Max** | **Yes** | **Yes** |
| **@Size** | **Yes** | **Yes** |
| **@NotNull** | **Yes** | **Yes** |
| **@ValidCategory(categoryType="...")** | **No (custom/invalid)** | **No** |

**Summary Table**

**Thus, the expression not used to create an object validator in Spring is @ValidCategory(categoryType="sample").**

**Question:**

**In the classpath of a Spring Boot application, there are 2 files with properties application.properties and application-prod.properties. It is required to always load properties from application.properties and override with values in the application-prod.properties file only when the application is deployed on the production server.**

**What should the value of the environment be to achieve this on the production server?**

**Pick ONE option**

* spring.properties=application-prod.properties
* spring.profiles.active=application-prod
* **spring.profiles.active=prod**
* environment=prod

**s**

**The correct answer is:  
spring.profiles.active=prod**

**Explanation**

* **In Spring Boot, if there are files named application.properties and application-prod.properties, the way to make the application load overrides only for production is by setting the active profile to prod.**
* **This is done by using the environment variable or JVM argument:**

**text**

**spring.profiles.active=prod**

* **With this setting, Spring Boot will first load values from application.properties, and then override with those in application-prod.properties when deployed on the production server.**

**Why Other Options Are Incorrect**

* **spring.properties=application-prod.properties**
  + **This is not a valid Spring Boot configuration property.**
* **spring.profiles.active=application-prod**
  + **The filename should be application-prod.properties (profile name is prod), so the profile must be set to prod, not application-prod.**
* **environment=prod**
  + **This is not a recognized property for activating profiles in Spring Boot.**

**Summary Table**

| **Property** | **Correct?** | **Reason** |
| --- | --- | --- |
| **spring.profiles.active=prod** | **Yes** | **Activates application-prod.properties for prod env** |
| **spring.profiles.active=application-prod** | **No** | **Wrong profile name; doesn't match convention** |
| **spring.properties=application-prod.properties** | **No** | **Invalid Spring Boot property** |
| **environment=prod** | **No** | **Not used for profile activation in Spring Boot** |

**Setting spring.profiles.active=prod is the standard way to achieve this environment-specific property override.**

**A Spring application has an interface called Server, and two implementations: ServerA and ServerB. There is a class, ServerManager, that uses the Server bean as a dependency.**

**java**

**public interface Server {**

**}**

**@Service**

**public class ServerA implements Server {**

**}**

**@Service**

**public class ServerB extends ServerA {**

**}**

**@Service**

**public class ServerManager {**

**@Autowired**

**Server server;**

**}**

**Which of the following statements is true about this code?**

**Pick ONE option**

* The code throws InterfaceNotInstantiatableException.
* The code runs fine. A random Server implementation is injected into the server field.
* **The code throws NoUniqueBeanDefinitionException.**
* The code does not compile.

**s**

**The correct answer is:  
The code throws NoUniqueBeanDefinitionException.**

**Explanation**

* **In Spring, when there are multiple beans that match a single @Autowired dependency (here, both ServerA and ServerB implement Server), Spring's auto-wiring mechanism cannot decide which bean to inject.**
* **As a result, unless you qualify the injection using @Primary or @Qualifier, the application context will fail to start and throw a NoUniqueBeanDefinitionException due to multiple matching beans for the type Server.**

**Why Other Options Are Incorrect**

* **The code throws InterfaceNotInstantiatableException.**
  + **Spring does not try to instantiate interfaces directly; it looks for implementations. This exception is not thrown in this scenario.**
* **The code runs fine. A random Server implementation is injected into the server field.**
  + **This only happens if there is exactly one bean matching the type or if @Primary/@Qualifier is used, which is not the case here. Without explicit selection, Spring fails due to ambiguity.**
* **The code does not compile.**
  + **All definitions are syntactically correct; the error occurs at runtime during dependency injection, not at compile time.**

**Summary Table**

| **sOption** | **Correct?** | **Reason** |
| --- | --- | --- |
| **Throws InterfaceNotInstantiatableException** | **No** | **Spring never tries to instantiate interfaces** |
| **Runs fine, a random Server impl injected** | **No** | **Ambiguity causes runtime error** |
| **Throws NoUniqueBeanDefinitionException** | **Yes** | **Multiple beans match, so Spring fails to resolve** |
| **Code does not compile** | **No** | **Compiles, runtime bean resolution throws error** |

**Spring throws the exception NoUniqueBeanDefinitionException in cases of ambiguous dependency injection.**

**Question: During the startup of a Spring Boot application, it needs to read bean configuration metadata and change it before the container instantiates any beans.**

**How can this be achieved in an efficient and scalable way?**

**Pick ONE option**

* Implement BeanPostProcessor.
* **Implement BeanFactoryPostProcessor.**
* It is not possible to change beans metadata on runtime. All beans metadata is defined at compile time.
* Implement Aspect.

**The correct answer is:  
Implement BeanFactoryPostProcessor.**

**Explanation**

* **During Spring Boot application startup, if there is a need to read and modify bean configuration metadata before any bean instances are created, the appropriate approach is to implement a BeanFactoryPostProcessor.**
* **BeanFactoryPostProcessor allows for modification of the application context's internal bean factory before any beans are instantiated, enabling changes to bean definitions such as property values or scopes.**
* **BeanPostProcessor works on already instantiated beans; it is too late in the lifecycle if bean metadata must be changed before instantiation.**
* **Changing bean metadata at runtime is possible with BeanFactoryPostProcessor, so the statement about it being impossible is incorrect.**
* **Implementing an Aspect is for cross-cutting concerns (AOP), but not meant for modifying bean metadata before instantiation.**

**Why Others Are Incorrect**

| **Option** | **Explanation** |
| --- | --- |
| **BeanPostProcessor** | **Acts after bean instantiation, cannot modify metadata early** |
| **BeanFactoryPostProcessor** | **Correct: modifies bean definitions before instantiation** |
| **Not possible to change metadata at runtime** | **Incorrect; BeanFactoryPostProcessor allows this** |
| **Implement Aspect** | **Not intended for bean metadata modification** |

**Therefore, to modify bean metadata efficiently before beans are created at startup, implement**

**Question:s**

**Consider the following code.**

**NotifierMetricLogger.java**

**java**

**package spring.listener;**

**import org.aspectj.lang.annotation.Around;**

**import org.aspectj.lang.annotation.Aspect;**

**import org.aspectj.lang.annotation.Before;**

**import org.aspectj.lang.annotation.AdviceName;**

**import org.springframework.stereotype.Component;**

**import org.aspectj.lang.JoinPoint;**

**import org.aspectj.lang.ProceedingJoinPoint;**

**import import java.util.logging.Logger;**

***// X***

**@Component**

**public class NotifierMetricLogger {**

**private static final Logger log =**

**Logger.getLogger(NotifierAspect.class.getName());**

**public Object beforeNotifyLogging(ProceedingJoinPoint joinPoint) throws Throwable {**

**long startDate = System.currentTimeMillis();**

**Object process = joinPoint.proceed();**

**long executionMs = System.currentTimeMillis() - startDate;**

**log.info("Notify process time =" + executionMs);**

**return process;**

**}**

**}**

**Pick ONE option**

* X = @Aspect, Y = @Before("execution\* spring.service.impl.*notifyIf.*")
* X = @AdviceName("NotifierMetricLogger"), Y = @Around("execution\* spring.service.impl.*notifyIf.*")
* X = @AdviceName("NotifierMetricLogger"), Y = @Before("execution\* spring.service.impl.*notifyIf.*")
* **X = @Aspect, Y = @Around("execution\* spring.service.impl.Notifier.notifyIf.\*")**

**The correct answer is:  
*X = @Aspect, Y = @Around("execution spring.service.impl.Notifier.notifyIf.*")\*\***

**Explanation**

* **The class NotifierMetricLogger is an aspect class, so it should be annotated with @Aspect to indicate it contains advice methods for AOP.**
* **The method beforeNotifyLogging is designed to measure execution time around a method call, which is suitable for an around advice annotated with @Around.**
* **The pointcut expression "execution\* spring.service.impl.Notifier.notifyIf.\*" matches the execution of methods in the specified package/class that the aspect should intercept.**
* **Other options either use incorrect annotations (@AdviceName is not a standard Spring AOP annotation) or inappropriate advice types (@Before instead of @Around for timing execution).**

**Why Other Options Are Incorrect**

* **@Before advice cannot control method execution or measure duration around the method call.**
* **@AdviceName is not a valid Spring AOP annotation.**
* **The pointcut expression must correctly specify the class and method pattern for interception, which the chosen answer does.**

**Thus, marking the class with @Aspect and using @Around with the correct pointcut expression is the accurate implementation for this logging aspect.**

**Question:**

**Consider the following code.**

**Circular.java**

**package spring.circular;**

**public interface Circular {**

**void doCircularThings();**

**}**

**CircularBeanA.java**

**java**

**package spring.circular;**

**import org.springframework.stereotype.Component;**

**import javax.annotation.PostConstruct;**

***// X***

**@Component**

**public class CircularBeanA implements Circular {**

**private Circular circularBeanB;**

**public CircularBeanA(Circular circularBeanB) {**

**this.circularBeanB = circularBeanB;**

**}**

**@PostConstruct**

**public void init() {**

**System.out.println("CircularBeanA: Initialized");**

**doCircularThings();**

**}**

**public void doCircularThings() {**

**System.out.println("CircularBeanA: did circular things");**

**}**

**}**

**Pick ONE or MORE options**

* The application runs successfully and it'll output  
  CircularBeanA: Initialized  
  CircularBeanA: did circular things
* The application won't run.  
  The code doesn't compile.
* The application runs but it will throw NoUniqueBeanDefinitionException and it will stall.
* If @Qualifier("circularBeanA") annotation is put on Z and @Qualifier("circularBeanB") put on Y, the application will not throw BeanCurrentlyInCreationException and runs successfully.
* **The application runs and it will write  
  CircularBeanA: Initialized into console  
  then it will throw org.springframework.beans.factory.BeanCurrentlyInCreationException.**
* **If @PostConstruct annotation is removed in CircularBeanB, it will run successfully and will print  
  CircularBeanA: Initialized into console**

**The correct answer is:  
The application runs and it will write  
CircularBeanA: Initialized into console  
then it will throw org.springframework.beans.factory.BeanCurrentlyInCreationException.**

**Explanation**

* **The constructor of CircularBeanA requires a bean of type Circular, but if there is a circular dependency (e.g., if CircularBeanB also depends, directly or indirectly, on CircularBeanA), Spring will encounter problems resolving this during bean creation.**
* **The @PostConstruct annotation makes the initialization sequence explicit, and after printing "CircularBeanA: Initialized", a circular reference causes Spring to raise a BeanCurrentlyInCreationException.**
* **This is a classic symptom of unresolvable circular dependencies in Spring's constructor injection. The application may start printing the initialization message, but then crash when it cannot complete bean creation.**

**Why Other Options Are Incorrect**

* **The application runs successfully and it'll output CircularBeanA: Initialized and CircularBeanA: did circular things**
  + **Incorrect, as the circular dependency will prevent successful bean creation and application startup.**
* **The application won't run. The code doesn't compile.**
  + **Incorrect, the code is syntactically valid Java/Spring; the error is at runtime.**
* **The application runs but it will throw NoUniqueBeanDefinitionException**
  + **No ambiguity—if only two beans exist, runtime failure is not due to multiple candidates, but circular creation.**
* **If @Qualifier("circularBeanA") / @Qualifier("circularBeanB") is put...**
  + **This may resolve ambiguity between beans, but qualifier does not resolve circular dependencies.**
* **If @PostConstruct annotation is removed in CircularBeanB, it will run successfully...**
  + **The presence or absence of @PostConstruct does not resolve circular references in constructor injection.**

| **Option** | **Correct** | **Reason** |
| --- | --- | --- |
| **Runs and prints, then throws BeanCurrentlyInCreationException** | **Yes** | **Circular dependency triggers this runtime Spring error** |
| **Runs and prints both messages, no error** | **No** | **Circular dependency breaks app startup** |
| **Code doesn't compile** | **No** | **Compile-time error not relevant** |
| **Throws NoUniqueBeanDefinitionException** | **No** | **Not caused by circular reference** |
| **@Qualifier solves it** | **No** | **Does not fix the underlying circular dependency** |
| **Removing @PostConstruct from B fixes it** | **May be** | **Read belows** |

**Summary Table**

**Thus, given the code and dependency setup, the application fails at runtime with a BeanCurrentlyInCreationException, not at compile time or due to ambiguous beans.**

**The last option states:  
"If @PostConstruct annotation is removed in CircularBeanB, it will run successfully and will print CircularBeanA: initialized into console."**

**Rechecking this with respect to Spring's circular dependency handling and the role of @PostConstruct:**

* **In Spring, circular dependencies with constructor injection are generally not resolvable by default, resulting in exceptions like BeanCurrentlyInCreationException, regardless of lifecycle annotations like @PostConstruct.**
* **However, @PostConstruct triggers initialization logic after the bean is fully created and dependencies injected. If removing @PostConstruct means no early method invocation, it might delay or avoid triggering some calls that fail due to incomplete initialization.**
* **But this does not fully resolve a circular dependency caused by constructor injection itself; Spring will still fail to create the beans if they depend circularly via constructors.**
* **In this specific case, if the circular dependency is caused only by the method call in @PostConstruct, then removing it could prevent the failure and allow successful bean initialization.**
* **Therefore, depending on the exact circular references and where the failure is triggered, this option could be true in certain scenarios.**

**Conclusion**

* **The last option could be correct if the circular dependency failure is caused or revealed by the method annotated with @PostConstruct trying to access not-yet-fully-initialized beans.**
* **So, the two options checked (The application runs and then throws BeanCurrentlyInCreationException... and If @PostConstruct is removed ... it will run successfully...) can both be valid depending on the context and exact code behavior.**

| **Option** | **Validity** |
| --- | --- |
| **Application runs, prints init message, then throws BeanCurrentlyInCreationException** | **Correct** |
| **Removing @PostConstruct in CircularBeanB allows successful run and prints init message** | **Also correct (can be valid depending on when failure triggers)** |

**Summary**

**Both options offer plausible behaviors based on the timing and cause of the circular reference failure in Spring.**

**Question:**

**Consider the following code.**

**java**

**<CODE HERE>**

**public class Processor {**

**public void process() {**

***// not relevant code***

**}**

**}**

**@RestController**

**public class ProcessController {**

**@Autowired**

**private Processor processor;**

**@GetMapping("/process")**

**public void process() {**

**processor.process();**

**}**

**}**

**What should be inserted in place of <CODE HERE> to have a new instance of processor created every time the /process endpoint is called?**

**Pick ONE option**

* @Scope("prototype") @Component
* @Prototype
* **@Component @Scope(scopeName = "prototype", proxyMode = ScopedProxyMode.TARGET\_CLASS)**
* @Component(autowireCandidate=true)
* @Service(alwaysNew=true)

**The correct answer is:  
@Component @Scope(scopeName = "prototype", proxyMode = ScopedProxyMode.TARGET\_CLASS)**

**Explanation**

* **In Spring, the default bean scope is singleton, meaning only one instance of the bean is created and shared.**
* **By using @Scope("prototype") along with @Component, Spring creates a new bean instance each time it is requested from the container.**
* **However, when injecting a prototype bean into a singleton (like a controller), the prototype behavior is preserved only if a proxy is used. Therefore, proxyMode = ScopedProxyMode.TARGET\_CLASS is necessary for ProcessController to receive a fresh Processor instance on each endpoint call.**
* **The other options either miss proper annotation combination or do not enable true prototype/proxy behavior.**

**Why Other Options Are Incorrect**

* **@Scope("prototype") @Component**
  + **Would work if the bean was always requested manually, but autowiring into a singleton (the controller) would not guarantee a new instance per call without proxy mode.**
* **@Prototype**
  + **Not a valid Spring annotation; the correct usage is @Scope("prototype").**
* **@Component(autowireCandidate=true)**
  + **Has no impact on bean scope or instantiation frequency.**
* **@Service(alwaysNew=true)**
  + **Not a standard Spring annotation or supported property.**

**Summary Table**

| **Option** | **Correct?** | **Reason** |
| --- | --- | --- |
| **@Component @Scope("prototype", proxyMode = ScopedProxyMode.TARGET\_CLASS)** | **Yes** | **Ensures new bean instance with proper proxying** |
| **@Scope("prototype") @Component** | **No** | **Doesn't trigger new instance when autowired in a singleton** |
| **@Prototype** | **No** | **Not a valid annotation** |
| **@Component(autowireCandidate=true)** | **No** | **Doesn't control instantiation** |
| **@Service(alwaysNew=true)** | **No** | **Not a valid setting** |

**sThus, to create a new instance of Processor each time the endpoint is called, use @Component @Scope(scopeName = "prototype", proxyMode = ScopedProxyMode.TARGET\_CLASS).**

**Question:**

**Analyze the behavior of the following code that collects a stream into an unmodifiable map when duplicate keys are present. Identify what exception, if any, will be thrown during execution.s**

**import java.util.stream.\*;**

**import java.util.\*;**

**public class Test {**

**public static void main(String[] args) {**

**List list = Arrays.asList("a", "b", "a");**

**Map map = list.stream()**

**.collect(Collectors.toUnmodifiableMap(**

**s -> s,**

**s -> s.toUpperCase()**

**));**

**System.out.println(map);**

**}**

**}**

**Pick ONE option**

* **s**
* The code merges duplicate keys by keeping the last value without throwing any exception**.**

**The correct answer is:  
The code throws IllegalStateException due to duplicate keys when collecting to an unmodifiable map.**

**Explanation**

* **In Java streams, Collectors.toUnmodifiableMap() creates an unmodifiable map and does not allow duplicate keys.**
* **When the input stream contains duplicate keys (as in the example where "a" appears twice), the default behavior is to throw an IllegalStateException indicating duplicate keys are present during the collection phase.**
* **The alternative of merging duplicate keys by keeping the last value requires using an explicit merge function, which is not provided in this code snippet.**
* **Therefore, the code as shown results in an exception due to duplicate keys, preventing map creation and printing.**

**Summary**

| **Behavior** | **Correctness** | **Reason** |
| --- | --- | --- |
| **Throws IllegalStateException due to duplicate keys** | **Correct** | **Default collector behavior prohibits duplicate keys in unmodifiable map** |
| **Merges keys by keeping last value without exceptions** | **Incorrect** | **Requires explicit merge function which is absent in this code** |

**Hence, the code throws an exception at runtime because of duplicate keys when collecting to an unmodifiable map**

An unmodifiable map in Java is a read-only view of a map whose content cannot be altered through that view.

Attempts to modify it (such as put, remove, or clear) will result in an UnsupportedOperationException.

Question:

Which link will hit the following method, where the application context is /home?

java

@RequestMapping(value = {"/ev/basic/bar", "/ev/basic/foo"},

method = RequestMethod.GET)

**public** String getPath() {

**return** "FooBar";

}

**Pick ONE option**

* **"GET request**[**http://localhost:8080/home/ev/basic/bar**](http://localhost:8080/home/ev/basic/bar)**"**
* "GET request <http://localhost:8080/home/ev/basic>"
* "POST Request <http://localhost:8080/home/ev/basic/bar>"
* Two routes cannot be mapped to one resource.

The correct answer is:  
**GET request**[**http://localhost:8080/home/ev/basic/bar**](http://localhost:8080/home/ev/basic/bar)

**Explanations**

* The method is mapped with @RequestMapping to two paths: "/ev/basic/bar" and "/ev/basic/foo" with HTTP GET method.
* The application context is /home, so the full path for either mapping becomes /home/ev/basic/bar or /home/ev/basic/foo.
* The request must be a **GET request** to these paths for the method to be invoked.
* The option for /home/ev/basic without the /bar or /foo suffix does not match the mapping, so it won't hit this method.
* The POST request option is incorrect since the method is mapped to GET only.
* Mapping multiple paths to a single method is valid in Spring, so the statement about routes not being mappable to the same resource is false.

Thus, the correct link that triggers the method is:  
**GET request**[**http://localhost:8080/home/ev/basic/bar**](http://localhost:8080/home/ev/basic/bar).

Question:

An HTTP POST endpoint accepts a JSON representation of a User object. It uses the following User and UserController classes.

java

**public** **class** User {

@NotEmpty

**private** String name;

@NotEmpty

**private** String surname;

}

@RestController

**public** **class** UserController {

@PostMapping("/users")

**public** **void** create(@RequestBody User user) {

System.out.println("name: " + user.getName() + " surname: " + user.getSurname());

}

}

What is the result of a call with the payload below?

json

{

"name": "Duke",

"surname": **null**

}

**Pick ONE option**

* BAD REQUEST 400 and a message that user.surname is validated by @NotEmpty annotation.
* OK 200 and prints 'name: Duke surname: null'
* SERVER ERROR 500 because of a NullPointerException during validation of null string for @NotEmpty
* OK 200 and prints 'name: Duke surname:'

The correct answer is:  
**OK 200 and prints 'name: Duke surname: null'**

**Explanation**

* The @NotEmpty annotation is a validation constraint, but **it requires explicit validation trigger**, such as using @Valid on the @RequestBody parameter in the controller method.
* In the provided code, there is **no @Valid annotation** on the User parameter, so validation won't be performed automatically.
* As a result, the POST request with "surname": null will **not cause validation errors**, and the method will run normally printing the values, including surname: null.
* No 400 Bad Request or 500 errors will be thrown because validation is not triggered and no exception occurs.

**Why Other Options Are Incorrect**

* **BAD REQUEST 400 and message from @NotEmpty violation**
  + Validation is not triggered since @Valid is missing.
* **SERVER ERROR 500 due to NullPointerException during validation**
  + Validation does not run, so no exception occurs.
* **OK 200 and prints 'name: Duke surname:'**
  + The actual printed value will be null as passed, not an empty string.

**Summary**

| **Outcome** | **Correctness** | **Reason** |
| --- | --- | --- |
| OK 200 and prints with surname null | Correct | No validation triggered, so prints passed values including null |
| BAD REQUEST 400 on validation error | Incorrect | Validation not executed without @Valid on User parameter |
| SERVER ERROR 500 due to validation NullPointerException | Incorrect | No validation runs; no exception |
| OK 200 and prints empty string for surname | Incorrect | It prints null as-is from JSON payload |

Thus, without explicit validation activation, @NotEmpty is ineffective, leading to the observed behavior.

A Spring Boot application has the following hierarchy of classes.

java

**public** **class** Animal {

@PostConstruct

**private** **void** init() {

System.out.println("Animal init");

}

}

@Component

**public** **class** Cat **extends** Animal {

@PostConstruct

**public** **void** init() {

System.out.println("Cat init");

}

}

@Lazy

@Component

**public** **class** Dog **extends** Animal {

@PostConstruct

**public** **void** init() {

System.out.println("Dog init");

}

}

What is the output?

**Pick ONE option**

* IllegalBeanDefinitionException: @PostConstruct should be applied to a public method
* Cat init Dog init
* **Animal init Cat init**
* Animal init Cat init Animal init Dog init
* Animal init Cat init Dog init or Animal init Dog init Cat init

The correct answer is:  
**Animal init Cat init**

**Explanation**

* Spring Boot initializes beans based on their scope and lifecycle. Cat is a normal @Component, while Dog is marked with @Lazy.
* **Lazy beans are not initialized during application startup** unless explicitly referenced or injected somewhere.
* For inheritance with @PostConstruct, Spring calls the parent class @PostConstruct method first, then the child class method.
* Since Dog is lazy and not referenced in the code shown, it won't be initialized at startup, so only Cat gets initialized.
* The initialization order for Cat will be: Animal init (parent) followed by Cat init (child).

**Why Other Options Are Incorrect**

* **IllegalBeanDefinitionException**: @PostConstruct can be applied to private methods; Spring uses reflection to access them.
* **Cat init Dog init**: This ignores parent class initialization and assumes Dog gets initialized (but it's lazy).
* **Animal init Cat init Animal init Dog init**: Dog won't initialize at startup due to @Lazy.
* **Animal init Cat init Dog init or Animal init Dog init Cat init**: Dog won't initialize at startup.

**Summary**

Since Dog is @Lazy and not referenced, only Cat initializes at startup with its parent Animal, producing the output: **Animal init Cat init**.

Question:

There is a requirement to upgrade an existing project with Maven support using an editor.

The project specifications are as follows:

* The Java version needs to be 1.8.
* External dependency: abc-def version 1.0.0, where abc-def is the name of the external dependencies.
* Dependency package information: abc:def

Which of the following are the valid maven entries for the build file according to the given data?

**Pick ONE option**

<properties>

<maven.compiler.source>8</maven.compiler.source>

<maven.compiler.target>8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>abc-def</groupId>

<artifactId>abc-def</artifactId>

<version>1.0.0</version>

</dependency>

</dependencies>

<properties>

<maven.compiler.source>8</maven.compiler.source>

<maven.compiler.target>8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>abc:def</groupId>

<artifactId>abc:def</artifactId>

<version>1.0.0</version>

</dependency>

</dependencies>

<dependencies>

<dependency>

<groupId>abc-def</groupId>

<artifactId>abc-def</artifactId>

<version>1.0.0</version>

</dependency>

</dependencies>

<properties>

<maven.compiler.source>8</maven.compiler.source>

<maven.compiler.target>8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>abc-def</groupId>

<artifactId>abc-def</artifactId>

</dependency>

</dependencies>

The correct answer is:  
**Option 1** (first option in the list)

**Explanation**

* The correct Maven entries should include the Java compiler configuration (version 8 for Java 1.8) and a proper dependency declaration.
* Maven groupId and artifactId must be valid identifiers, using hyphens (-), not colons (:). The package information abc:def is not a valid Maven artifact format, but abc-def is valid and matches industry convention.
* The answer with both properties and dependencies, and groupId/artifactId as abc-def, correctly configures the build file for the specified project update.

**Thus, the valid Maven entries are provided in Option 1.**

The fourth option is incorrect because:

* It **omits the <version> element** in the <dependency> declaration, which is critical for Maven to know which version of the external dependency to download and use.
* Without specifying the version, Maven will fail to resolve the dependency properly, resulting in build errors or unexpected behavior.
* The project specification explicitly mentions the external dependency version as **1.0.0**, so this version must be included in the <dependency> section.
* Other parts of the fourth option (Maven compiler properties and groupId/artifactId) are correct, but missing the version makes the dependency declaration incomplete and invalid for the given requirement.

**What is the output of the following code?**

java

**public** **class** Main{

**public** **static** **void** main(String[] args) {

**try** {

**int**[] a = **new** **int**[5];

String s="1.0";

**try** {

System.out.println(Integer.parseInt(s+a[4]));

} **catch** (Exception e) {

System.out.println("Exception 1 occurs. " + e.getClass());

**throw** **new** Exception();

}

} **catch** (Exception e) {

System.out.println("Exception 2 occurs");

}

}

}

**Pick ONE option**

* Exception 2 occurs
* **Exception 1 occurs. class java.lang.NumberFormatException Exception 2 occurs**
* Exception 1 occurs. class ArrayIndexOutOfBoundsException Exception 2 occurs
* Compile-Time Error

The correct answer is:  
**Exception 1 occurs. class java.lang.NumberFormatException Exception 2 occurs**

**Explanation**

* The code concatenates the string s ("1.0") with the value of a (which is 0, since a is an int array of size 5 initialized with zeros).
* So the expression s + a results in "1.00".
* Then, Integer.parseInt("1.00") is called, which attempts to parse "1.00" as an integer. This causes a **NumberFormatException** because "1.00" is not a valid integer format.
* This exception is caught in the inner catch block (exception 1), which prints the exception class and then throws a new checked exception.
* The new exception is caught by the outer catch block, which prints "Exception 2 occurs".

**Output Steps:**

1. "Exception 1 occurs. class java.lang.NumberFormatException" is printed.
2. "Exception 2 occurs" is printed.

**Why Other Options Are Incorrect**

* The code does not cause **ArrayIndexOutOfBoundsException** because accessing a is valid (index 4 exists in array of size 5).
* There is no **compile-time error** since the syntax is correct.
* "Exception 2 occurs" alone doesn't cover the first exception print.

Thus, the output matches the option indicating both exceptions with a NumberFormatException first.

Analyze the behavior when collecting a stream into an unmodifiable map where either keys or values are null. Identify if null keys or values are permitted or cause exceptions.

**import** java.util.Map;

**import** java.util.stream.Collectors;

**import** java.util.stream.Stream;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Map map = Stream.of("a", "b")

.collect(Collectors.toUnmodifiableMap(

k -> k.equals("a") ? **null** : k,

v -> v.equals("b") ? **null** : v

));

System.out.println(map);

}

}

**Pick ONE option**

* **Collecting with null keys throws NullPointerException immediately during collection.**
* Collecting with null values throws NullPointerException immediately during collection.
* Collectors.toUnmodifiableMap allows null keys and values without exceptions.
* Collectors.toUnmodifiableMap silently ignores entries with null keys or values.

Explanation already done before

A dependency has been added to pom.xml in a Spring application as follows.

xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

What is the starter used for?

**Pick ONE option**

* application analysis
* creation of an application
* automation of an application
* production of application instances

The correct answer is:  
**application analysis**

**Explanation**

* The spring-boot-starter-actuator starter is used to add **application monitoring and management features** to a Spring Boot application.
* It provides ready-to-use endpoints that expose application metrics, health, environment properties, and other monitoring data which helps in **analyzing application behavior at runtime**.
* It is not used for creation, automation, or production of application instances but specifically for **observability and analysis** of the running application.

Hence, the spring-boot-starter-actuator is primarily for **application analysis**.The correct answer is:  
**spring.profiles.active=prod**

**Explanation**

* In Spring Boot, profile-specific properties files like application-prod.properties are activated by setting the active profile to prod.
* This is done by setting the environment variable or property:  
  spring.profiles.active=prod
* This makes Spring Boot load application.properties and override properties with those in application-prod.properties when the profile is active (such as on the production server).
* Other options like spring.properties=application-prod.properties or environment=prod are not valid ways to activate profiles in Spring Boot.

Therefore, to override properties only on production server, set the environment variable or system property like this: