PassQueen
a.k.a dumpsqueen.com

Java SE 21 Developer Professional

Oracle 1z0-830

Version Demo

Total Demo Questions: 7

Total Premium Questions: 75

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Handling Date, Time, Text, Numeric and Boolean Values	11
Topic 2, Controlling Program Flow	5
Topic 3, Using Object-Oriented Concepts	10
Topic 4, Handling Exceptions	5
Topic 5, Working with Arrays and Collections	13
Topic 6, Working with Streams and Lambda expressions	15
Topic 7, Packaging and Deploying Java Code and Managing Java Projects	5
Topic 8, Managing Concurrent Code Execution	4
Topic 9, Using Java I/O API	6
Topic 10, Mix Questions	1
Total	75

QUESTION NO: 1

Given:

```
var cabarets = new TreeMap<>();

cabarets.put(1, "Moulin Rouge");

cabarets.put(2, "Crazy Horse");

cabarets.put(3, "Paradis Latin");

cabarets.put(4, "Le Lido");

cabarets.put(5, "Folies Bergère");

System.out.println(cabarets.subMap(2, true, 5, false));
```

What is printed?

A. {2=Crazy Horse, 3=Paradis Latin, 4=Le Lido, 5=Folies Bergère}

B. {2=Crazy Horse, 3=Paradis Latin, 4=Le Lido}

C. {}

D. An exception is thrown at runtime.

E. Compilation fails.

ANSWER: B

Explanation:

{2=Crazy Horse, 3=Paradis Latin, 4=Le Lido} is printed because `TreeMap` stores its entries in ascending key order. The keys inserted into this map are the integers 1 through 5, so their natural ordering determines the map's iteration and display order.

The four-argument `subMap` method returns a range view bounded by the supplied keys. Here, the lower endpoint is key `2` and its inclusion flag is `true`, so the entry `2=Crazy Horse` belongs to the returned view. The upper endpoint is key `5` and its inclusion flag is `false`, so the entry for key 5 is not part of that view. Every key strictly between these bounds is included; therefore, keys 3 and 4 contribute `3=Paradis Latin` and `4=Le Lido`.

When the resulting map view is passed to `println`, its standard map string representation lists those entries in the map's sorted order, separated by commas and enclosed in braces. Oracle documents this range operation in the NavigableMap subMap API; TreeMap implements that ordered navigable-map behavior.

QUESTION NO: 2

Given:

java

```
double amount = 42_000.00;

NumberFormat format = NumberFormat.getCompactNumberInstance(Locale.FRANCE, NumberFormat.Style.SHORT);

System.out.println(format.format(amount));
```

What is the output?

A. 42000E

B. 42 000,00 €

C. 42000

D. 42 k

ANSWER: D

Explanation:

`42 k` is the output. `NumberFormat.getCompactNumberInstance(Locale.FRANCE, NumberFormat.Style.SHORT)` creates a formatter that uses the locale-specific compact decimal patterns for France. Compact formatting abbreviates a value by applying the applicable magnitude pattern rather than rendering every digit. For a value of `42_000.00`, the relevant magnitude is thousands, so the formatter scales the value to `42` and appends the French short compact suffix `k`. The output uses a nonbreaking space between the number and suffix, represented in the option as ` `, yielding `42 k`. The decimal portion is not displayed because this compact representation has no fractional component for this value under the formatter's default precision behavior. Java's `NumberFormat` API specifies that compact number instances format numbers in a compact, locale-sensitive form, and its short style is intended for abbreviated output such as thousands and millions. The exact symbols and patterns are locale data, so selecting `Locale.FRANCE` is what produces the French compact presentation. See the Java SE 21 NumberFormat compact-number API documentation and the Java SE 21 Locale API documentation.

QUESTION NO: 3

Consider the following methods to load an implementation of MyService using ServiceLoader. Which of the methods are correct? (Choose all that apply)

A. MyService service = ServiceLoader.load(MyService.class).iterator().next();

B. MyService service = ServiceLoader.load(MyService.class).findFirst().get();

C. MyService service = ServiceLoader.getService(MyService.class);

D. MyService service = ServiceLoader.services(MyService.class).getFirstInstance();

ANSWER: A B

Explanation:

`MyService service = ServiceLoader.load(MyService.class).iterator().next();` is valid because `ServiceLoader.load(MyService.class)` creates a loader for providers of the `MyService` service. A `ServiceLoader` is iterable, so its `iterator()` supplies an iterator over discovered provider instances. Calling `next()` obtains the first available provider instance. Provider discovery and instantiation are lazy, meaning the provider may be located and created as iteration advances. This form therefore loads an implementation when at least one configured provider is available.

`MyService service = ServiceLoader.load(MyService.class).findFirst().get();` is also valid. `ServiceLoader.findFirst()` returns an `Optional` containing the first available provider instance, if one is found. Calling `get()` retrieves that instance, so the expression has type `MyService` and can be assigned directly to the variable. As with iteration, provider lookup is performed lazily. Both expressions assume that the service configuration makes at least one provider available; otherwise, their terminal retrieval operations fail because no provider can be obtained. The Java SE 21 specification documents both the iterable behavior and `findFirst()` on `ServiceLoader`: Java SE 21 ServiceLoader API. The behavior of the optional retrieval is documented in the Optional.get() API.

QUESTION NO: 4

Which StringBuilder variable fails to compile?

java

public class StringBuilderInstantiations {

public static void main(String[] args) {

var stringBuilder1 = new StringBuilder();

var stringBuilder2 = new StringBuilder(10);

var stringBuilder3 = new StringBuilder("Java");

var stringBuilder4 = new StringBuilder(new char[]{'J', 'a', 'v', 'a'});

}

}

A. None of them

B. stringBuilder4

C. stringBuilder1

D. stringBuilder3

E. stringBuilder2

ANSWER: B

Explanation:

`stringBuilder4` fails to compile because the expression `new char[]{'J', 'a', 'v', 'a'}` has type `char[]`, and `StringBuilder` provides no constructor that accepts a character array. The relevant constructors accept no argument, an `int` capacity, a `String`, or a `CharSequence`. Although a `String` implements `CharSequence`, an array type such as `char[]` does not implement that interface, so it cannot be supplied to the `CharSequence` constructor. Consequently, overload resolution finds no applicable `StringBuilder` constructor for that argument and compilation stops at this declaration. The Java API specification lists the supported `StringBuilder` constructors at StringBuilder API documentation. To construct a builder from the array's characters, first create a `String`, as in `new StringBuilder(new String(chars))`, or create an empty builder and use its `append(char[])` method. The append(char[]) API documentation describes the available character-array append operation.

QUESTION NO: 5

Given:

java

final Stream < String > strings =

Files.readAllLines(Paths.get("orders.csv"));

strings.skip(1)

.limit(2)

.forEach(System.out::println);

And that the orders.csv file contains:

mathematica

OrderID,Customer,Product,Quantity,Price

1,Kylian Mbappé,Keyboard,2,25.50

2,Teddy Riner,Mouse,1,15.99

3,Sebastien Loeb,Monitor,1,199.99

4,Antoine Griezmann,Headset,3,45.00

What is printed?

A. arduino1,Kylian Mbappé,Keyboard,2,25.502,Teddy Riner,Mouse,1,15.993,Sebastien Loeb,Monitor,1,199.994,Antoine Griezmann,Headset,3,45.00

B. arduino1,Kylian Mbappé,Keyboard,2,25.502,Teddy Riner,Mouse,1,15.99

C. arduino2,Teddy Riner,Mouse,1,15.993,Sebastien Loeb,Monitor,1,199.99

D. Compilation fails because Files.readAllLines returns a List, not a Stream.

E. Compilation fails.

ANSWER: D E

Explanation:

Compilation fails because the expression on the right side of the assignment has the wrong type. The `Files.readAllLines(Paths.get("orders.csv"))` method returns a `List<String>`, whereas the variable `strings` is declared as `Stream<String>`. A `List<String>` cannot be assigned directly to a `Stream<String>`, so no stream pipeline is created and nothing can be printed. The code also invokes a file-reading method whose declaration includes `IOException`; therefore, in a complete method body, that checked exception must be caught or declared with an appropriate `throws` clause. Either compile-time issue prevents execution. To use the existing list-based operation, the assignment would need to invoke `.stream()` on the returned list. Alternatively, `Files.lines(path)` directly returns a `Stream<String>`; that stream should be managed with try-with-resources because it holds an open file resource. The corrected compilation-failure statements are therefore accurate. Oracle documents the return type and exception contract for `readAllLines` in the Files API, and documents `Files.lines` in the same API.

QUESTION NO: 6

Which methods compile?

A. public List getListSuper() { return new ArrayList(); }

B. public List getListExtends() { return new ArrayList(); }

C. public List getListExtends() { return new ArrayList(); }

D. public List getListSuper() { return new ArrayList(); }

ANSWER: A B

Explanation:

`public List<? super IOException> getListSuper() { return new ArrayList<Exception>(); }` compiles because the lower-bounded wildcard describes a `List` whose element type is some unknown supertype of `IOException`. `Exception` is such a supertype, so an `ArrayList<Exception>` is a valid value for that return type. Likewise, `public List<? extends IOException> getListExtends() { return new ArrayList<FileNotFoundException>(); }` compiles because the upper-bounded wildcard permits a list parameterized by an unknown subtype of `IOException`. `FileNotFoundException` extends `IOException`, making `ArrayList<FileNotFoundException>` compatible with the declared result type. Wildcards express a family of permitted parameterizations rather than one fixed element type; therefore, assignment compatibility is determined by whether the concrete type argument satisfies the wildcard bound. This follows the Java Language Specification's rules for wildcard type arguments and subtyping, and the Oracle generics tutorial's descriptions of upper- and lower-bounded wildcards. See JLS §4.5.1, Type Arguments of Parameterized Types and Oracle: Upper Bounded Wildcards.

QUESTION NO: 7

Given:

java

String s = " ";

System.out.print("[" + s.strip());

s = " hello ";

System.out.print("," + s.strip());

s = "h i ";

System.out.print("," + s.strip() + "]");

What is printed?

A. [,hello,h i]

B. [,hello,h i]

C. [,hello,hi]

D. [, hello ,hi]

ANSWER: B

Explanation:

The output is `[,hello,h i]`. The `String.strip()` method returns a string with leading and trailing Unicode whitespace removed; it does not remove whitespace occurring between non-whitespace characters. For the initial value containing only one space, `strip()` produces the empty string, so the first print call emits only the opening bracket. When `s` is assigned `" hello "`, stripping removes the space at each end and leaves `hello`; the second call therefore appends `,hello`. Finally, the value `"h i "` has a trailing space and an internal space. Stripping removes the trailing space but preserves the internal space between `h` and `i`. The last print call appends `,h i]`, including the closing bracket. Because `System.out.print`, rather than `println`, is used for every call, all fragments are written consecutively on one line with no intervening line terminators. Oracle specifies that `strip()` uses `Character.isWhitespace` to identify removable leading and trailing characters and returns the resulting string. See the Java SE 21 String.strip() API documentation and the Character.isWhitespace API documentation.