

DUMPS ARENA

Java SE 17 Developer

Oracle 1z0-829

Version Demo

Total Demo Questions: 6

Total Premium Questions: 68

Buy Premium PDF

<https://dumpsarena.co>

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Handling Date, Time, Text, Numeric and Boolean Values	9
Topic 2, Controlling Program Flow	3
Topic 3, Using Object-Oriented Concepts	11
Topic 4, Handling Exceptions	2
Topic 5, Working with Arrays and Collections	4
Topic 6, Working with Streams and Lambda expressions	12
Topic 7, Using Java I/O API	6
Topic 8, Using the Java SE 17 Platform Module System	6
Topic 9, Concurrency	6
Topic 10, Database Applications with JDBC	1
Topic 11, Localization	1
Topic 12, Mix Questions	7
Total	68

QUESTION NO: 1

Given the code fragment:

```
Optional<String> code = Optional.of("A");

String result = code.or(() -> {
    throw new IllegalStateException();
}).orElse("B");

System.out.println(result);
```

What is the result?

- A. A
- B. B
- C. An `IllegalStateException` is thrown.
- D. Compilation fails.

ANSWER: A

Explanation:

The result is A. The `Optional` referenced by `code` is non-empty, so `Optional.or()` returns that existing optional. Its supplier is not evaluated.

Consequently, the lambda that throws `IllegalStateException` is never invoked. The resulting optional contains A, and `orElse("B")` returns that contained value.

QUESTION NO: 2

Given:

Which two should the module-info file include for it to represent the service provider interface?

- A. Requires `com.transport.vehicle,cars`;
- B. Provides `com.transport.vehicle.cars.Car` with `com.transport.vehicle.cars. impt, CatImpl`;
- C. Requires `com.transport.vehicle,cars`;
- D. Provides `com.transport.vehicle.cars.Car impl,Carlmp1` to `com.transport.vehicle.cars. Cars`
- E. exports `com.transport.vehicle.cars.Car`;
- F. Exports `com.transport.vehicle.cars`;
- G. Exports `com.transport.vehicle`;

ANSWER: B F

Explanation:

A module that supplies a service implementation declares the relationship using a `provides` directive with a service type and its implementation class. In this case, the intended declaration is `provides com.transport.vehicle.cars.Car with com.transport.vehicle.cars.impl.CarImpl`; . This makes the implementation discoverable through `ServiceLoader` without a configuration file. The service type must be accessible to modules that load the service, so the package that contains the `Car` service interface must also be exported. The intended export is `exports com.transport.vehicle.cars`; ; a module directive exports a package, rather than an individual class. Together, these declarations publish the service contract and register the provider implementation while keeping implementation-package

details out of the module's public API. The Java Language Specification defines the `provides ... with ...` module directive and its provider-class requirements in [JLS 7.7.4](#). The Java Platform Module System specification describes service binding and how providers are located by service type in [ServiceLoader API documentation](#).

QUESTION NO: 3

Given the code fragments:

Which action prints Wagon : 200?

- A. At line n1, implement the `java.io.Serializable` interface.
- B. At line n3, replace `readObject()` with `readLine()`.
- C. At Line n3, replace `Car` with `LuxurayCar`.
- D. At Line n1, implement the `java.io.AutoCloseable` interface
- E. At line n2, in the main method signature, add `throws IOException, ClassCastException`.
- F. At line n2, in the main method signature, add `throws IOException, ClassNotFoundException`.

ANSWER: F

Explanation:

Adding `throws IOException, ClassNotFoundException` to the main method signature permits the deserialization code to compile and run. Constructing an `ObjectInputStream`, closing it, and reading an object through `readObject()` involve checked exceptions. In particular, `readObject()` declares `ClassNotFoundException` because the JVM may be unable to locate the class named in the serialized stream, and stream operations can throw `IOException`. Declaring both exceptions allows them to propagate from `main` rather than requiring local `try/catch` handling. Once the serialized `LuxuryCar` instance is successfully read and assigned through the compatible `Car` reference, invoking `toString()` uses the runtime object's overridden implementation. Consequently, the output is `Wagon : 200`. Oracle's `ObjectInputStream.readObject()` API documents both checked exceptions in its method contract, while the Java language specification describes how a method may declare checked exceptions with a `throws` clause. See [ObjectInputStream.readObject\(\) documentation](#) and [JLS chapter 11: Exceptions](#).

QUESTION NO: 4

Given the code fragment:

Which code fragment returns different values?

- A. `int sum = listOfNumbers.parallelStream().reduce(5, Integer::sum);`
- B. `int sum = listOfNumbers.stream().reduce(5, (a, b) -> a + b);`
- C. `int sum = listOfNumbers.stream().reduce(Integer::sum).orElse(5) + 5;`
- D. `int sum = listOfNumbers.parallelStream().reduce((m, n) -> m + n).orElse(5) + 5;`
- E. `int sum = listOfNumbers.stream().reduce(0, Integer::sum) + 5;`

ANSWER: A

Explanation:

`int sum = listOfNumbers.parallelStream().reduce(5, Integer::sum);` is the fragment that can return a different result. In a reduction with an identity value, the identity must be neutral for the accumulator operation: combining it with any stream element must leave that element unchanged. For integer addition, the neutral identity is 0, not 5. A parallel

stream is divided into independent substreams, and each substream's partial reduction may begin with the supplied identity. Consequently, 5 can be introduced more than once—once for each partition—and those extra values are included when partial results are combined. The result can therefore differ from a sequential reduction that introduces 5 only once, and it may depend on how the stream is partitioned. This is precisely why reduction operations intended for parallel execution require an identity compatible with the accumulator. Oracle's `Stream.reduce(identity, accumulator)` documentation specifies that the identity must be an identity for the accumulator function, while the Streams API package documentation describes the requirements for associative, non-interfering reduction functions. See [Stream.reduce API documentation](#) and [java.util.stream package documentation](#).

QUESTION NO: 5

Given these module declarations:

```
module util {
    exports com.acme.util;
}

module library {
    requires transitive util;
    exports com.acme.library.api;
}

module app {
    requires library;
}
```

Which two statements are true? Select two.

- A. The app module reads the util module.
- B. The app module must declare `requires util;` before it can use public types in `com.acme.util`.
- C. The app module can access public types in `com.acme.util`.
- D. The app module can access public types in every package contained in the util module.
- E. The library module does not read the util module.

ANSWER: A C

Explanation:

The `requires transitive util` directive makes readability of `util` implied for modules that read `library`. Since `app` requires `library`, `app` also reads `util`. Because `util` exports `com.acme.util` without qualification, public types in that package are accessible to `app`.

The `app` module does not need its own `requires util` directive. If `library` used ordinary `requires util` instead, readability would not be propagated to modules reading `library`. Exporting a package controls access to public types; it does not make non-exported packages accessible.

QUESTION NO: 6

Given the module declaration:

```
module orders {
    exports com.acme.order.api;
    exports com.acme.order.audit to reports;
}
```

Assume that both the `reports` module and the `audit` module require `orders`. Which two statements are true? Select two.

- A. The `reports` module can access public types in both exported packages.
- B. The `audit` module can access public types in `com.acme.order.api` but not in `com.acme.order.audit`.
- C. The `audit` module can access both packages because it requires `orders`.
- D. The qualified export opens `com.acme.order.audit` for deep reflection by every module.
- E. The package `com.acme.order.api` is accessible only to the `reports` module.

ANSWER: A B

Explanation:

The unqualified export of `com.acme.order.api` makes its public and protected types accessible to every module that reads `orders`. Therefore, both `reports` and `audit` can access that package.

The package `com.acme.order.audit` is exported only to the module named `reports`. The `audit` module reads `orders`, but it is not a target of the qualified export and cannot access public types in that package. An `exports` directive does not make a package open for deep reflection; that requires an `opens` directive.