

# Oracle

## Exam Questions 1z0-829

Java SE 17 Developer



**NEW QUESTION 1**

Given:

```
import java.io.Serializable;
public class Software implements Serializable {
    private String title;
    public Software(String title) {
        this.title = title;
        System.out.print("Software ");
    }
    public String toString() { return title; }
}

public class Game extends Software {
    private int players;
    public Game(String title, int players) {
        super(title);
        this.players = players;
        System.out.print("Game ");
    }
    public String toString() { return super.toString()+" "+players; }
}

import java.io.*;
public class AppStore {
    public static void main(String[] args) {
        Software s = new Game("Chess", 2);
        try(ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("game.ser"))) {
            out.writeObject(s);
        } catch (Exception e) {
            System.out.println("write error");
        }
        try(ObjectInputStream in = new ObjectInputStream(new FileInputStream("game.ser"))) {
            s = (Software)in.readObject();
        } catch (Exception e) {
            System.out.println("read error");
        }
        System.out.println(s);
    }
}
```

What is the result?

- A. Software Game Chess 0
- B. Software Game Software Game Chess 2
- C. Software game write error
- D. Software Game Software Game chess 0
- E. Software Game Chess 2
- F. Software Game read error

**Answer:** B**Explanation:**

The answer is B because the code uses the writeObject and readObject methods of the ObjectOutputStream and ObjectInputStream classes to serialize and deserialize the Game object. These methods use the default serialization mechanism, which writes and reads the state of the object's fields, including the inherited ones. Therefore, the title field of the Software class is also serialized and deserialized along with the players field of the Game class. The toString method of the Game class calls the toString method of the Software class using super.toString(), which returns the value of the title field. Hence, when the deserialized object is printed, it shows Software Game Software Game Chess 2.

References:

- ? Oracle Certified Professional: Java SE 17 Developer
- ? Java SE 17 Developer
- ? OCP Oracle Certified Professional Java SE 17 Developer Study Guide
- ? Serialization and Deserialization in Java with Example

**NEW QUESTION 2**

Which statement is true?

- A. IllegalStateException is thrown if a thread in waiting state is moved back to runnable.
- B. thread in waiting state consumes CPU cycles.
- C. A thread in waiting state must handle InterruptedException.

D. After the timed wait expires, the waited thread moves to the terminated state.

**Answer:** C

**Explanation:**

A thread in waiting state is waiting for another thread to perform a particular action, such as calling `notify()` or `notifyAll()` on a shared object, or terminating a joined thread. A thread in waiting state can be interrupted by another thread, which will cause the waiting thread to throw an `InterruptedException` and return to the runnable state. Therefore, a thread in waiting state must handle `InterruptedException`, either by catching it or declaring it in the `throws` clause. References: `Thread.State` (Java SE 17 & JDK 17), `[Thread` (Java SE 17 & JDK 17)]

**NEW QUESTION 3**

Assuming that the data, txt file exists and has the following content:

Text1 Text2 Text3

Given the code fragment:

```
try {
    Path p = new File("data.txt").toPath();
    Stream lines = Files.lines(p);
    String data = lines.collect(Collectors.joining("-"));
    System.out.println(data);
    String data2 = Files.readAllLines(p).get(3);
    System.out.println(data2);
} catch (IOException ex) {
    System.out.println(ex);
}
```

What is the result?

- A. text1- text2- text3- text3
- B. text1-text2-text3 text1text2 text3
- C. text1-text2-text3A `java.lang.indexoutofBoundsException` is thrown.
- D. text1-text2-text3 text3

**Answer:** D

**Explanation:**

The answer is D because the code fragment reads the file `data.txt` and collects all the lines in the file into a single string, separated by hyphens. Then, it prints the resulting string. Next, it attempts to read the fourth line in the file (index 3) and print it. However, since the file only has three lines, an `IndexOutOfBoundsException` is thrown. References:

? Oracle Certified Professional: Java SE 17 Developer

? Java SE 17 Developer

? OCP Oracle Certified Professional Java SE 17 Developer Study Guide

? Read contents of a file using Files class in Java

**NEW QUESTION 4**

Which two code fragments compile?

A)

```
class L6 {  
    public static void main(String[] args) {  
        var x = new ArrayList<>();  
        x.add(10);  
        x.add("30");  
        System.out.println(x);  
    }  
}
```

B)

```
class L2 {  
    public void m(int x) {  
        var x = 10;  
    }  
}
```

C)

```
class A {}  
class B extends A {}  
class L4 {  
    public static void main(String[] args) {  
        var x = new A();  
        x = new B();  
    }  
}
```

D)

```
class L3 {  
    public static void main(String[] args) {  
        var a = 10;  
        a = "30";  
    }  
}
```

E)

```
class L5 {  
    public void m() {  
        var strVar = null;  
    }  
}
```

- A. Option A
- B. Option B
- C. Option C
- D. Option D
- E. Option E

**Answer:** BE**Explanation:**

The two code fragments that compile are B and E. These are the only ones that use the correct syntax for declaring and initializing a var variable. The var keyword is a reserved type name that allows the compiler to infer the type of the variable based on the initializer expression. However, the var variable must have an initializer, and the initializer must not be null or a lambda expression. Therefore, option A is invalid because it does not have an initializer, option C is invalid because it has a null initializer, and option D is invalid because it has a lambda expression as an initializer. Option B is valid because it has a String initializer, and option E is valid because it has an int initializer. <https://docs.oracle.com/en/java/javase/17/language/local-variable-type-inference.html>

**NEW QUESTION 5**

Given the product class:

```
import java.io.*;  
public class Product implements Serializable {  
    private static float averagePrice = 2.99f;  
    private String description;  
    private transient float price;  
    public Product(String description, float price) {  
        this.description = description;  
        this.price = price;  
    }  
    public void readObject(ObjectInputStream in)  
        throws IOException, ClassNotFoundException {  
        in.defaultReadObject();  
        price = averagePrice;  
    }  
    public String toString() {  
        return description+" "+price+" "+averagePrice;  
    }  
}
```

And the shop class:

```
import java.io.*;
public class Shop {
    public static void main(String[] args) {
        Product p = new Product("Cookie", 3.99f);
        try {
            try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("p.ser"))) {
                out.writeObject(p);
            }
            try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("p.ser"))) {
                p = (Product)in.readObject();
            }
        } catch (Exception e) { e.printStackTrace(); }
        System.out.println(p);
    }
}
```

What is the result?

- A. Cookie 2.99 2.99
- B. Cookie 3.99 2.99
- C. Cookie 0.0 0.0
- D. An exception is produced at runtime
- E. Compilation fails
- F. Cookie 0.0 2.99

**Answer:** E

**Explanation:**

The code fragment will fail to compile because the readObject method in the Product class is missing the @Override annotation. The readObject method is a special method that is used to customize the deserialization process of an object. It must be declared as private, have no return type, and take a single parameter of type ObjectInputStream. It must also be annotated with @Override to indicate that it overrides the default behavior of the ObjectInputStream class. Without the @Override annotation, the compiler will treat the readObject method as a normal method and not as a deserialization hook. Therefore, the code fragment will produce a compilation error. References: Object Serialization - Oracle, [ObjectInputStream (Java SE 17 & JDK 17) - Oracle]

**NEW QUESTION 6**

Which statement is true about migration?

- A. Every module is moved to the module path in a top-down migration.
- B. Every module is moved to the module path in a bottom-up migration.
- C. The required modules migrate before the modules that depend on them in a top-down migration.
- D. Unnamed modules are automatic modules in a top-down migration.

**Answer:** B

**Explanation:**

The answer is B because a bottom-up migration is a strategy for modularizing an existing application by moving its dependencies to the module path one by one, starting from the lowest-level libraries and ending with the application itself. This way, each module can declare its dependencies on other modules using the module-info.java file, and benefit from the features of the Java Platform Module System (JPMS), such as reliable configuration, strong encapsulation, and service loading.

Option A is incorrect because a top-down migration is a strategy for modularizing an existing application by moving it to the module path first, along with its dependencies as automatic modules. Automatic modules are non-modular JAR files that are treated as modules with some limitations, such as not having a module descriptor or a fixed name. A top-down migration allows the application to use the module path without requiring all of its dependencies to be modularized first.

Option C is incorrect because a top-down migration does not require any specific order of migrating modules, as long as the application is moved first and its dependencies are moved as automatic modules. A bottom-up migration, on the other hand, requires the required modules to migrate before the modules that depend on them.

Option D is incorrect because unnamed modules are not automatic modules in any migration strategy. Unnamed modules are modules that do not have a name or a module descriptor, such as classes loaded from the class path or dynamically generated classes. Unnamed modules have unrestricted access to all other modules, but they cannot be accessed by named modules, except through reflection with reduced security checks. References:

? Oracle Certified Professional: Java SE 17 Developer

? Java SE 17 Developer

? OCP Oracle Certified Professional Java SE 17 Developer Study Guide

? Migrating to Modules (How and When) - JavaDeploy

? Java 9 Modularity: Patterns and Practices for Developing Maintainable Applications

**NEW QUESTION 7**

Given the code fragment:

```
abstract sealed interface SInt permits Story, Art {
    default String getTitle() { return "Book Title" ; }
}
```

```
abstract sealed interface SInt permits Story, Art { default String getTitle() { return "Book Title" ; }
}
```

Which set of class definitions compiles?

- A. Interface story extends STnt {} Interface Art extends SInt {}
- B. Public interface story extends sInt {} Public interface Art extends SInt {}
- C. Sealed interface Story extends SInt {} Non-sealed class Art implements Sint {}
- D. Non-sealed interface story extends SInt {} Class Art implements Sint {}
- E. Non-sealed interface story extends SInt {} Non-sealed interaface Art extends Sint {}

**Answer:** C

**Explanation:**

The answer is C because the code fragment given is an abstract sealed interface SInt that permits Story and Art. The correct answer is option C, which is a sealed interface Story that extends SInt and a non-sealed class Art that implements SInt. This is because a sealed interface can only be extended by the classes or interfaces that it permits, and a non-sealed class can implement a sealed interface.

Option A is incorrect because interface is misspelled as interace, and Story and Art should be capitalized as they are the names of the permitted classes or interfaces.

Option B is incorrect because public is misspelled as public, and sInt should be SInt as it is the name of the sealed interface.

Option D is incorrect because a non-sealed interface cannot extend a sealed interface, as it would violate the restriction of permitted subtypes.

Option E is incorrect because both Story and Art cannot be non-sealed interfaces, as they would also violate the restriction of permitted subtypes.

References:

? Oracle Certified Professional: Java SE 17 Developer

? Java SE 17 Developer

? OCP Oracle Certified Professional Java SE 17 Developer Study Guide

? Sealed Classes and Interfaces in Java 15 | Baeldung

? Sealed Class in Java - Javatpoint

**NEW QUESTION 8**

Given:

```
1. class Item {
2.     String name;
3.     public static void display() {
4.         name = "Vase";
5.         System.out.println(name);
6.     }
7.     public void display(String design) {
8.         this.name += name;
9.         System.out.println(name);
10.    }
11. }
12. public class App {
13.     public static void main(String[] args) {
14.         Item i1 = new Item();
15.         i1.display("Flower");
16.     }
17. }
```

Which action enables the code to compile?

- A. Replace 15 with item.display ("Flower");
- B. Replace 2 with static string name;
- C. Replace 7 with public void display (string design) {
- D. Replace 3 with private static void display () {

**Answer:** C

**Explanation:**

The answer is C because the code fragment contains a syntax error in line 7, where the method display is declared without any parameter type. This causes a compilation error, as Java requires the parameter type to be specified for each method parameter. To fix this error, the parameter type should be added before the parameter name, such as string design. This will enable the code to compile and run without any errors. References:

? Oracle Certified Professional: Java SE 17 Developer

? Java SE 17 Developer

? OCP Oracle Certified Professional Java SE 17 Developer Study Guide

? Java Methods

**NEW QUESTION 9**

Given the code fragment:

```
String s = "10_00";  
Integer s2 = 10_00;  
// Line n1  
System.out.println(res);
```

Which two statements at Line n1 independently enable you to print 1250?

A. Integer res = 250 + integer.parseInt (s)

B. Integer res = 250 + s;

C. Integer res = 250 + integer (s2):

D. Integer res= 250 + s2;

E. Integer res = 250 + integer . valueOf (s);

F. Integer res = 250; Res = + s2;

**Answer:** AE

**Explanation:**

The code fragment is creating a string variable ??s?? with the value ??10\_00?? and an integer variable ??s2?? with the value 10. The string ??s?? is using an underscore as a separator for readability, which is allowed in Java SE 17.1. The question is asking for two statements that can add 250 to the numeric value of ??s?? and assign it to an integer variable ??res??. The correct answers are A and E because they use the methods parseInt and valueOf of the Integer class to convert the string ??s?? to an integer. Both methods interpret the string as a signed decimal integer and return the equivalent int or Integer value. The other options are incorrect because they either use invalid syntax, such as B and C, or they do not convert the string ??s?? to an integer, such as D and F. References: Binary Literals (The Java™ Tutorials > Learning the Java Language > Numbers and Strings), Integer (Java SE 17 & JDK 17), Integer (Java SE 17 & JDK 17)

**NEW QUESTION 10**

Given:

```
package com.transport.vehicle.cars;

public interface Car {
    int getSpeed();
}

and

package com.transport.vehicle.cars.impl;

import com.transport.vehicle.cars.Car;

public class CarImpl implements Car {
    private int speed;

    public CarImpl() {
        this(10);
    }

    public CarImpl (int speed) {
        this.speed = speed;
    }

    @Override
    public int getSpeed() {
        return speed;
    }
}
```

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

- A. Requires cm.transport.vehicle,cars:
- B. Provides com.transport.vehicle.cars.Car with com.transport.vehicle.car
- C. impt, CatImpl;
- D. Requires cm.transport.vehicle,cars:
- E. Provides com.transport.vehicle.cars.Car impl,CarImpl1 to com.transport.vehicle.car
- F. Cars
- G. exports com.transport.vehicle.cars.Car;
- H. Exports com.transport.vehicle.cars;
- I. Exports com.transport.vehicle;

**Answer:** BE

**Explanation:**

The answer is B and E because the module-info file should include a provides directive and an exports directive to represent the service provider interface. The provides directive declares that the module provides an implementation of a service interface, which is com.transport.vehicle.cars.Car in this case. The with clause specifies the fully qualified name of the service provider class, which is com.transport.vehicle.cars.impl.CarImpl in this case. The exports directive declares that the module exports a package, which is com.transport.vehicle.cars in this

case, to make it available to other modules. The package contains the service interface that other modules can use.

Option A is incorrect because requires is not the correct keyword to declare a service provider interface. Requires declares that the module depends on another module, which is not the case here.

Option C is incorrect because it has a typo in the module name. It should be com.transport.vehicle.cars, not cm.transport.vehicle.cars.

Option D is incorrect because it has a typo in the keyword provides. It should be provides, not Provides. It also has a typo in the service interface name. It should be com.transport.vehicle.cars.Car, not com.transport.vehicle.cars.Car impl. It also has an unnecessary to clause, which is used to limit the accessibility of an exported package to specific modules.

Option F is incorrect because it exports the wrong package. It should export com.transport.vehicle.cars, not com.transport.vehicle.cars.impl. The impl package contains the service provider class, which should not be exposed to other modules.

Option G is incorrect because it exports the wrong package. It should export com.transport.vehicle.cars, not com.transport.vehicle. The vehicle package does not contain the service interface or the service provider class. References:

? Oracle Certified Professional: Java SE 17 Developer

? Java SE 17 Developer

? OCP Oracle Certified Professional Java SE 17 Developer Study Guide

? Java Modules - Service Interface Module - GeeksforGeeks

? Java Service Provider Interface | Baeldung

#### NEW QUESTION 10

Given the content of the in. tart file: 23456789

and the code fragment:

```
char[] buffer = new char[8];
int count = 0;
try(FileReader in = new FileReader("in.txt");
    FileWriter out = new FileWriter("out.txt")) {
    while((count = in.read(buffer)) != -1) {
        out.write(buffer);
    }
}
```

What is the content of the out .txt file?

- A. 01234567801234
- B. 012345678
- C. 0123456789234567
- D. 0123456789
- E. 012345678901234
- F. 01234567

**Answer:** D

#### Explanation:

The answer is D because the code fragment reads the content of the in.txt file and writes it to the out.txt file. The content of the in.txt file is ??23456789??. The code fragment uses a char array buffer of size 8 to read the content of the in.txt file. The while loop reads the content of the in.txt file and writes it to the out.txt file until the end of the file is reached. Therefore, the content of the out.txt file will be ??0123456789??.

#### NEW QUESTION 15

Given the code fragments:

```
class Car implements Serializable {
    private static long serialVersionUID = 454L;
    String name;
    public Car(String name) { this.name = name; }
}

class LuxuryCar extends Car {           // line n1
    int flag_HHC;
    public LuxuryCar(String name, int flag_HHC) {
        super(name);
        this.flag_HHC = flag_HHC;
    }
    public String toString() {
        return name + " : " + flag_HHC;
    }
}

and:
public static void main(String[] args) {    // line n2
    Car b = new LuxuryCar("Wagon", 200);
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("car.ser"));
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream("car.ser"));) {
        oos.writeObject(b);
        System.out.println((Car)(ois.readObject()));           // line n3
    }
}
```

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:**

The code fragment is trying to read an object from a file using the ObjectInputStream class. This class throws an IOException and a ClassNotFoundException. To handle these exceptions, the main method signature should declare that it throws these exceptions. Otherwise, the code will not compile. If the main method throws these exceptions, the code will print Wagon : 200, which is the result of calling the toString method of the LuxuryCar object that was written to the file. References: ObjectInputStream (Java SE 17 & JDK 17) - Oracle, ObjectOutputStream (Java SE 17 & JDK 17) - Oracle

**NEW QUESTION 17**

Given:

```
public class Weather {
    public enum Forecast {
        SUNNY, CLOUDY, RAINY;
        @Override
        public String toString() { return "SNOWY";}
    }

    public static void main(String[] args) {
        System.out.print(Forecast.SUNNY.ordinal() + " ");
        System.out.print(Forecast.valueOf("cloudy".toUpperCase()));
    }
}
```

What is the result?

- A. 1 RAINY
- B. Compilation fails

- C. 1 Snowy
- D. 0 CLOUDY
- E. 0 Snowy

**Answer:** E

**Explanation:**

The code is defining an enum class called Forecast with three values: SUNNY, CLOUDY, and RAINY. The toString() method is overridden to always return ??SNOWY??. In the main method, the ordinal value of SUNNY is printed, which is 0, followed by the value of CLOUDY converted to uppercase, which is ??CLOUDY??. However, since the toString() method of Forecast returns ??SNOWY?? regardless of the actual value, the output will be ??0 SNOWY??. References: Enum (Java SE 17 & JDK 17), Enum.EnumDesc (Java SE 17 & JDK 17)

**NEW QUESTION 21**

Assume you have an automatic module from the module path display-ascii-0.2. jar. Which name is given to the automatic module based on the given JAR file?

- A. Display.ascii
- B. Display-ascii-0.2
- C. Display-ascii
- D. Display-ascii-0

**Answer:** C

**Explanation:**

An automatic module name is derived from the name of the JAR file when it does not contain a module-info.class file. If the JAR file has an ??Automatic-Module-Name?? attribute in its main manifest, then its value is the module name. Otherwise, the module name is derived from the JAR file??s name by removing any version numbers and converting it to lower case. Therefore, for a JAR named display-ascii-0.2.jar, the automatic module name would be display-ascii, following these rules.

**NEW QUESTION 25**

.....

## Thank You for Trying Our Product

### We offer two products:

1st - We have Practice Tests Software with Actual Exam Questions

2nd - Questions and Answers in PDF Format

### 1z0-829 Practice Exam Features:

- \* 1z0-829 Questions and Answers Updated Frequently
- \* 1z0-829 Practice Questions Verified by Expert Senior Certified Staff
- \* 1z0-829 Most Realistic Questions that Guarantee you a Pass on Your First Try
- \* 1z0-829 Practice Test Questions in Multiple Choice Formats and Updates for 1 Year

**100% Actual & Verified — Instant Download, Please Click**  
**[Order The 1z0-829 Practice Test Here](#)**