DUMPSBOSS.

Java Foundations

Oracle 1z0-811

Version Demo

Total Demo Questions: 10

Total Premium Questions: 108

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

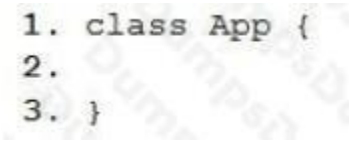
support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Java Basics	21
Topic 2, Java Data Types	8
Topic 3, Java Operators	10
Topic 4, Java Control Flow	12
Topic 5, Java Methods	10
Topic 6, Java Classes and Objects	24
Topic 7, Java Arrays	10
Topic 8, Java String	8
Topic 9, Java Exception Handling	5
Total	108

QUESTION NO: 1

Given the code fragment:



Which two code fragments are valid at line 2?

A. for (int count = 0; count < 5; count++) {
System.out.print(count);
}

B. package p1;

C. import java.util.*; public void display() { List nums = new ArrayList<> ();
}

D. {
private int num;
}

E. private String name = “John”;
public void display() { System.out.print(name);
}

ANSWER: B E

Explanation:

`package p1;` is a valid package declaration when placed in the compilation-unit declaration area shown in the fragment. A package declaration uses the `package` keyword followed by a qualified package name and a semicolon. It establishes the package to which the source file’s declared types belong. Java permits at most one such declaration in a compilation unit, and it must occur before import and type declarations. The Java Language Specification defines this source-file structure in JLS §7, Packages and Modules.

`private String name = “John”; public void display() { System.out.print(name); }` is valid class-member content at the indicated location. The field declaration creates an instance variable, initialized when an object is created, and the instance method can access that field directly because both members belong to the same class. Fields may use the `private` access modifier, and methods may use the `public` access modifier. Java class bodies can contain field declarations and method declarations, as specified in JLS §8, Classes. In normal Java source code, the quotation marks around `John` must be ordinary double-quote characters.

QUESTION NO: 2

Identify two valid data types for the operands of the addition (+) operator?

A. String

B. boolean

C. numeric

D. array

ANSWER: A C

Explanation:

`String` and numeric are valid operand types for Java's addition operator. When both operands are numeric primitive values, `+` performs arithmetic addition. Java supports numeric addition for the integral types `byte`, `short`, `int`, `long`, and `char`, as well as the floating-point types `float` and `double`. Before evaluating the expression, Java applies binary numeric promotion where necessary, so an expression such as `intValue + doubleValue` produces a numeric result of the promoted type.

The same operator also performs string concatenation when at least one operand is a `String`. For example, `"Total: " + 10` produces the string `"Total: 10"`; the non-`String` operand is converted to its string representation for the concatenation. Thus, `String` is specifically recognized by the language rules for the additive operator, while numeric types are recognized for arithmetic addition. Oracle's Java Language Specification defines these two behaviors under the additive operators and describes the primitive numeric types separately. See JLS §15.18, Additive Operators and JLS §4.2, Primitive Types and Values.

QUESTION NO: 3

Which statement is true about exception handling?

A. At least one catch block must accompany a try statement.

B. All statements in a try block are executed, even if an exception occurs in the middle of the try block.

C. At least one statement in a try block must throw an exception.

D. All catch blocks must be ordered from general to most specific.

E. A try statement without a resource specification must be followed by at least one catch block or a finally block.

ANSWER: E

Explanation:

"A try statement without a resource specification must be followed by at least one catch block or a finally block" is correct. In Java's basic `try` statement form, the protected code is placed in a `try` block and must be paired with exception-handling code in one or more `catch` clauses, cleanup code in a `finally` clause, or both. A `catch` clause provides a handler for an exception that can arise while the protected code runs. A `finally` clause provides code that is intended to run when control leaves the `try` construct, which is commonly used to perform cleanup. This requirement applies specifically to a `try` statement that has no resource specification. Java also has the distinct try-with-resources form, whose resource specification handles automatic closing of declared resources and has slightly different grammar. The Java Language Specification defines the syntax of a basic `try` statement as a `try` block followed by catches, a finally clause, or both. Oracle's Java tutorial also describes using `catch` to handle exceptions and `finally` for cleanup. See the Java Language Specification, §14.20 and the Oracle Java Exceptions tutorial.

QUESTION NO: 4

Identify two Java reserved words.

A. array

B. true

C. this

D. exception

E. string

ANSWER: B C

Explanation:

The correct selections are `true` and `this`. Java reserves `true` as a boolean literal. It has the predefined boolean value true and cannot be used as an identifier, such as the name of a variable, method, class, or package. Although it is categorized specifically as a literal rather than a keyword in the Java Language Specification, it is part of Java's reserved lexical vocabulary and is commonly included when questions use the broad term "reserved words."

`this` is a Java keyword. Within an instance method or constructor, it refers to the current object—the object on which the method or constructor is operating. For example, `this.name = name;` distinguishes an instance field from a parameter with the same name. Because `this` has this language-defined meaning, Java does not permit it to be declared or used as an identifier. The Java Language Specification defines Java's keywords and literal tokens in its lexical grammar; see JLS §3.9: Keywords and Restricted Identifiers and JLS §3.10.3: Boolean Literals.

QUESTION NO: 5

Given:

```
public class App {
    public static void main (String[] args) {
        System.out.println ("Hello Java!");
    }
}
```

Which statement is true about the main method?

A. It can be a non-static method.

B. Its parameter can be of type Integer [].

C. It cannot be defined in a non-public class.

D. It cannot be invoked by its name.

E. It must be declared static.

ANSWER: E

Explanation:

It must be declared static. In the traditional Java application entry-point model covered by Java Foundations, the Java launcher starts a class by locating and invoking a method with the signature `public static void main(String[] args)`, or its varargs equivalent, `public static void main(String... args)`. Declaring the method `static` is essential because the launcher invokes it through the class itself; it does not need to create an instance of the class first. The `String[]` parameter supplies command-line arguments as text values, and the `void` return type indicates that the entry-point method does not return a result to the launcher. Although program code may explicitly call `main` like any other accessible static method, the defining requirement for the conventional launcher-recognized main method is that it be static. Oracle's Java Language Specification describes the invocation of a class's `public static void main(String[])` method during program startup. See the Java Language Specification, section 12.1.4 and the Java launcher documentation.

QUESTION NO: 6

Which statement is true about a variable declared with the `final` modifier?

A. It can be assigned only once.

B. It must be declared static.

C. It cannot refer to an object.

D. It is accessible from all classes.

ANSWER: A

Explanation:

A `final` variable can be assigned only once. After the variable has been assigned, another assignment to that variable causes a compilation error. For a primitive variable, this prevents its stored value from changing. For a reference variable, it prevents the reference from being changed to refer to a different object; it does not necessarily make the referenced object immutable. See the Java Language Specification: final Variables.

QUESTION NO: 7

Given the code:

```
public class Calc {
    // line n1
        return a + b;
    }
    public static void main(String[] args) {
        Calc obj = new Calc();
        int c = obj.sum(10, 20);
        System.out.println("sum is " + c);
    }
}
```

Which code fragment, when inserted at line n1, enables the code to print sum is 30?

A. int sum(int a, b) {

B. int sum(int a, int b) {

C. int sum(int, int) {

D. int sum(int[] a, b) {

ANSWER: B

Explanation:

`int sum(int a, int b) {` is correct because it declares an instance method named `sum` that accepts two integer arguments, assigns each argument a valid parameter name, and returns an `int`. The method body shown after line n1 can therefore use `a` and `b` as its local parameter variables and return their arithmetic total. When the calling code supplies the values 10 and 20, Java binds those values to `a` and `b`, respectively. Evaluating `a + b` produces the integer value 30, which allows the surrounding print statement to display `sum is 30`.

Java method declarations require every formal parameter to specify both a type and an identifier. Here, `int` establishes that each input is a 32-bit signed integer, while `a` and `b` provide the names used by the method implementation. The opening brace also correctly begins the method body at the insertion point. Oracle's Java tutorial describes method declarations and their parameter lists at Java Methods, and the Java Language Specification defines formal parameters in JLS §8.4.1.

QUESTION NO: 8

Given the code fragment:

```
public static void main(String[] args) {
    String name = "Rita";
    int age = 14;
    /* line n1 */
}
```

Which code fragment, when inserted at line n1, enables it to print Rita is 14 years old?

A. System.out.println("%s is %d years old" +name+age);

B. System.out.println("%s is %n years old" name, age);

C. System.out.printf("%s is %d years old", name, age);

D. System.out.printf("%s is %n years old", name, age);

ANSWER: C

Explanation:

`System.out.printf("%s is %d years old", name, age);` is correct because `printf` writes formatted output using a format string followed by values that supply the format specifiers. The `%s` conversion formats the value held by `name` as a string, so it produces `Rita`. The `%d` conversion formats the integral value held by `age` as a decimal integer, so it produces `14`. The remaining literal characters in the format string, including the spaces and the words `is` and `years old`, are printed unchanged. Consequently, with `name` equal to `"Rita"` and `age` equal to `14`, the output is exactly `Rita is 14 years old`. Java's `PrintStream.printf` method uses the formatting rules defined by `Formatter`; its varargs argument list permits the name and age variables to be supplied in the same order as their corresponding conversions. See the PrintStream printf API and the Formatter API.

QUESTION NO: 9

Identify three features of the Java programming language.

A. distributed

B. direct memory management

C. multithreaded

D. strongly typed

E. dynamically typed

ANSWER: A C D

Explanation:

Java is **distributed** because its standard libraries provide networking capabilities that let applications communicate across networks, including support for URLs, sockets, and remote access to network resources. This makes Java suitable for client-server and networked applications. Java is also **multithreaded**: the platform includes built-in language and API support for executing multiple threads of work within a process, allowing programs to perform concurrent activities such as handling user interaction while background processing continues.

Java is **strongly typed** because every variable and expression has a defined type, and the compiler performs type checking before execution. The language requires declared types for variables (with local-variable type inference still resolving a static type at compile time), and only type-compatible assignments and operations are permitted. These language characteristics help detect type-related programming errors early and make program behavior more predictable. Oracle's Java documentation describes the platform's networking APIs and concurrency support, while the Java Language Specification defines the language's type system and compile-time checking rules. See Oracle Java networking APIs and Java Language Specification: Types, Values, and Variables.

QUESTION NO: 10

Which two statements are true about constructors?

A. A constructor has the same name as its class.

B. A constructor can be overloaded.

C. A constructor must declare the return type `void`.

D. A constructor is inherited by a subclass.

E. A constructor can be called directly on an existing object.

ANSWER: A B

Explanation:

A constructor has the same name as its class and does not declare a return type, not even `void`. It is invoked when an object is created with the `new` operator to initialize the newly created object.

A class can declare more than one constructor as long as the constructors have different parameter lists. This is constructor overloading. Constructors are not inherited by subclasses and cannot be invoked like ordinary methods on an existing object. See the Java Language Specification section on constructor declarations.