

**A.** 4321

**B.** 0000

**C.** An exception is thrown at runtime.

**D.** Compilation fails because of an error in line 18.

**Answer: D**

**Explanation:** The basic rule is a call (or invocation) to a constructor should be the first statement that JVM needs to execute,

So when you have a super class with only parameterized constructor and no default constructor, and base class has no explicit call to the parameterized constructor of the super class, JVM provides the super(); call which throws error as there is no default constructor for the super class, so either we provide a default constructor in the super class or we explicitly call the parameterized constructor of the super class in the base class constructor. when we give the explicit call, then JVM doesn't bother to put the line super(); as constructor invocation should be the first statement of the method, which cannot happen (because of our explicit call).

**QUESTION NO: 2**

Given:

abstract class C1 {

public C1() { System.out.print(1); }

}

class C2 extends C1 {

public C2() { System.out.print(2); }

}

class C3 extends C2 {

public C3() { System.out.println(3); }

}

public class Ctest {

public static void main(String[] a) { new C3(); }

}

What is the result?

**A.** 3

**B.** 23

**C.** 32

**D.** 123

**E.** 321

**F.** Compilation fails.

**G.** An exception is thrown at runtime.

**Answer: D**

**Explanation:** F & G are wrong as the code has no error.

A,B,C,E are wrong because when we call the derived class constructor(creating the derived class object) it will internally call base class constructor using super().

So in this example when we create the object C3 the constructor call will be in the following sequential

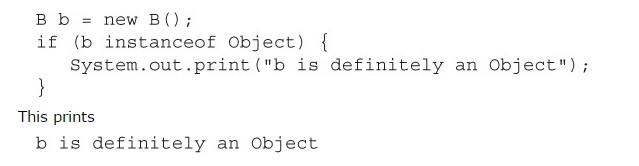
C3->C2->C1(C3 will call C2, C2 will call C1)

So D is the correct answer.

OCJP 7:

Operator:

1. **The equals() Method in Class Object** The equals() method in class Object works the same way that the == operator works. If two references point to the same object, the equals() method will return true. If two references point to different objects, even if they have the same values, the method will return false.
2. **The equals() Method in Class String** The equals() method in class String has been overridden. When the equals() method is used to compare two strings, it will return true if the strings have the same value, and it will return false if the strings have different values. For String’s equals() method, values ARE case sensitive.
3. You can test an object reference against its own class type or any of its superclasses. This means that any object reference will evaluate to true if you use the instanceof operator against type Object, as follows:



1. It is legal to test whether the null reference is an instance of a class. This will always result in false, of course.
2. Any array is always an instance of Object class. As arrays are objects in java.

image TWO-MINUTE DRILL

Here are some of the key points from each section in this chapter.

**Relational Operators (OCA Objectives 3.1 and 3.3)**

image Relational operators always result in a boolean value (true or false).

image There are six relational operators: >, >=, <, <=, ==, and !=. The last two (== and !=) are sometimes referred to as *equality operators*.

 When comparing characters, Java uses the Unicode value of the character as the numerical value.



image Equality operators

image There are two equality operators: == and !=.

image Four types of things can be tested: numbers, characters, booleans, and reference variables.

image When comparing reference variables, == returns true only if both references refer to the same object.

**instanceof Operator (OCA Objective 3.1)**

image instanceof is for reference variables only; it checks whether the object is of a particular type.

image The instanceof operator can be used only to test objects (or null) against class types that are in the same class hierarchy.

image For interfaces, an object passes the instanceof test if any of its superclasses implement the interface on the right side of the instanceof operator.

**Arithmetic Operators (OCA Objectives 3.1 and 3.2)**

image The four primary math operators are add (+), subtract (–), multiply (\*), and divide (/).

* The remainder (a.k.a. modulus) operator (%) returns the remainder of a division.

image Expressions are evaluated from left to right, unless you add parentheses, or unless some operators in the expression have higher precedence than others.

image The \*, /, and % operators have higher precedence than + and –.

**String Concatenation Operator (OCA Objective 3.1)**

image If either operand is a String, the + operator concatenates the operands.

image If both operands are numeric, the + operator adds the operands.

**Increment/Decrement Operators (OCA Objectives 3.1 and 3.2)**

image Prefix operators (for example, ++ x and --x) run before the value is used in the expression.

image Postfix operators (for example, x++ and x--) run after the value is used in the expression.

image In any expression, both operands are fully evaluated *before* the operator is applied.

image Variables marked final cannot be incremented or decremented.

**Ternary (Conditional) Operator (OCA Objective 3.1)**

image Returns one of two values based on whether its boolean expression is true or false.

image Returns the value after the ? if the expression is true.

image Returns the value after the : if the expression is false.

**Logical Operators (OCA Objective 3.1)**

image The exam covers six “logical” operators: &, |, ^, !, &&, and ||.

image Logical operators work with two expressions (except for !) that must resolve to boolean values.

image The && and & operators return true only if both operands are true.

image The || and | operators return true if either or both operands are true.

image The && and || operators are known as short-circuit operators.

image The && operator does not evaluate the right operand if the left operand is false.

image The || does not evaluate the right operand if the left operand is true.

image The & and | operators always evaluate both operands.

image The ^ operator (called the “logical XOR”) returns true if exactly one operand is true.

image The ! operator (called the “inversion” operator) returns the opposite value of the boolean operand it precedes.

Question: 1

public class Operator1 {

public static void main (String [] args)

{

if( args.length== 1 | args[1].equals("test"))

System.out.println("test case");

else

{

System.out.println("Production");

}

}

}

image

The output is

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1

at oracle.Operator1.main(Operator1.java:6)

Because the args has one element(live2) args[1] expression will give rutime arrayindexoutofboundsexception .

If we change if( args.length== 1 | args[1].equals("test")) as below the out will be

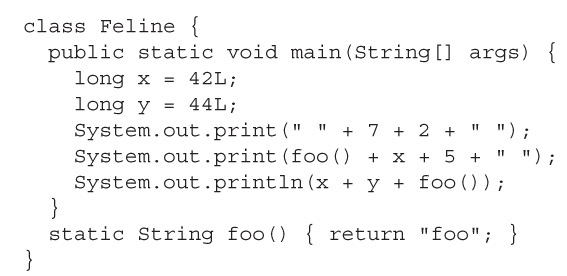
1. if( args.length== 1 | | args[1].equals("test"))

Output: test case

1. if( args.length== 1 | args[0].equals("test"))

Output: test case

**Question** 2:



**Output**: 72 foo425 86foo

**Explanation**: Concatenation runs from left to right, and if either operand is a String, the operands are concatenated. If both operands are numbers, they are added together.

**NOTE:** When dividing ints, remainders are always rounded down.

**Exception:**

**Definition:** When a program violates the semantic constraints of java programming language, the java virtual machine signals this error to the program as an exception.

* Semantics is meaning.
* Syntax is Symbolic representation.

Types of Exception:

**Compile Time Error: Checked Exception**

A Complier will check your syntax for you if syntax is wrong it will raise compile time error. These exceptions must be caught by a catch block or the thread will terminate, and so will the application will terminate if it is the only thread.

**Runtime Exception: Unchecked Exception**

Compiler will derive the semantics of syntax from language rules(mapping the syntax to machine instructions) if semantic is wrong it will raise run time exception. It won’t find all semantics error at compile time so it is called as unchecked exception.