



## Online Java Programming Test :: Java Programming Test - Random

Home » Online Test » Online Java Programming Test » Java Programming Test - Random

Marks : 4/20

Total number of questions : 20

Number of answered questions : 15

Number of unanswered questions : 5

**Test Review : View answers and explanation for this test.**

1. What will be the output of the program?

```
import java.util.*;
public class NewTreeSet2 extends NewTreeSet
{
    public static void main(String [] args)
    {
        NewTreeSet2 t = new NewTreeSet2();
        t.count();
    }
}
protected class NewTreeSet
{
    void count()
    {
        for (int x = 0; x < 7; x++,x++ )
        {
            System.out.print(" " + x);
        }
    }
}
```

- ☐ Ⓐ 0 2 4
- ☐ Ⓑ 0 2 4 6
- ☐ Ⓒ Compilation fails at line 2
- ☒ Ⓓ Compilation fails at line 10

[Contact Us](#) [Copyright](#) [Privacy Policy](#)

© IndiaBIX™ Technologies

**Your Answer:** Option Ⓓ

**Correct Answer:** Option **D**

**Explanation:**

Nonnested classes cannot be marked **protected** (or **final** for that matter), so the compiler will fail at **protected class TreeSet**.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Declarations and Access Control](#)

[#]

2. What will be the output of the program?

```
public class Test
{
    public static void main(String args[])
    {
        class Foo
        {
            public int i = 3;
        }
        Object o = (Object)new Foo();
        Foo foo = (Foo)o;
        System.out.println("i = " + foo.i);
    }
}
```

- ☐ **A** i = 3 ✓
- ☐ **B** Compilation fails.
- ☐ **C** i = 5
- ☒ **D** A **ClassCastException** will occur. ✗

**Your Answer:** Option **D**

**Correct Answer:** Option **A**

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Declarations and Access Control](#)

[#]

3. What will be the output of the program?

```
class Test
{
    public static void main(String [] args)
    {
        int x=20;
```

```
String sup = (x < 15) ? "small" : (x < 22)? "tiny" : "huge";
System.out.println(sup);
}
}
```

- ☐ Ⓐ small
- ☐ Ⓑ tiny ✓
- ☐ Ⓒ huge
- ☒ Ⓓ Compilation fails ✗

**Your Answer:** Option Ⓓ

**Correct Answer:** Option Ⓑ

**Explanation:**

This is an example of a nested ternary operator. The second evaluation `(x < 22)` is `true`, so the `"tiny"` value is assigned to `sup`.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Operators and Assignments](#)

[#]

4. What will be the output of the program?

```
public class If1
{
    static boolean b;
    public static void main(String [] args)
    {
        short hand = 42;
        if ( hand < 50 && !b ) /* Line 7 */
            hand++;
        if ( hand > 50 );      /* Line 9 */
        else if ( hand > 40 )
        {
            hand += 7;
            hand++;
        }
        else
            --hand;
        System.out.println(hand);
    }
}
```

- ☐ Ⓐ 41
- ☐ Ⓑ 42
- ☐ Ⓒ 50

☒ 51 ☒

**Your Answer:** Option ☐

**Correct Answer:** Option ☐

**Explanation:**

In Java, boolean instance variables are initialized to **false**, so the if test on line 7 is true and hand is incremented. Line 9 is legal syntax, a do nothing statement. The else-if is true so hand has 7 added to it and is then incremented.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Flow Control](#)

[#]

5. What will be the output of the program?

```
for(int i = 0; i < 3; i++)
{
    switch(i)
    {
        case 0: break;
        case 1: System.out.print("one ");
        case 2: System.out.print("two ");
        case 3: System.out.print("three ");
    }
}
System.out.println("done");
```

- ☒ ☐ done ☒
- ☐ ☐ one two done
- ☐ ☐ one two three done
- ☐ ☐ one two three two three done ☒

**Your Answer:** Option ☐

**Correct Answer:** Option ☐

**Explanation:**

The variable i will have the values 0, 1 and 2.

When i is 0, nothing will be printed because of the break in **case 0**.

When i is 1, "one two three" will be output because **case 1**, **case 2** and **case 3** will be executed (they don't have break statements).

When i is 2, "two three" will be output because **case 2** and **case 3** will be executed (again

no break statements).

Finally, when the for loop finishes "done" will be output.



[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Flow Control](#)

[#]

6. What will be the output of the program?

```
public class Switch2
{
    final static short x = 2;
    public static int y = 0;
    public static void main(String [] args)
    {
        for (int z=0; z < 3; z++)
        {
            switch (z)
            {
                case y: System.out.print("0 "); /* Line 11 */
                case x-1: System.out.print("1 "); /* Line 12 */
                case x: System.out.print("2 "); /* Line 13 */
            }
        }
    }
}
```

- ☒ **A** 0 1 2 
- ☐ **B** 0 1 2 1 2 2
- ☐ **C** Compilation fails at line 11. 
- ☐ **D** Compilation fails at line 12.

**Your Answer:** Option **A**

**Correct Answer:** Option **C**

**Explanation:**

Case expressions must be constant expressions. Since **x** is marked **final**, lines 12 and 13 are legal; however **y** is not a **final** so the compiler will fail at line 11.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Flow Control](#)

[#]

7. What will be the output of the program?

```
public class RTExcept
{
    public static void throwit ()
    {
        System.out.print("throwit ");
        throw new RuntimeException();
    }
    public static void main(String [] args)
    {
        try
        {
            System.out.print("hello ");
            throwit();
        }
        catch (Exception re )
        {
            System.out.print("caught ");
        }
        finally
        {
            System.out.print("finally ");
        }
        System.out.println("after ");
    }
}
```

- ☐ Ⓐ hello throwit caught
- ☒ Ⓑ Compilation fails ❌
- ☐ Ⓒ hello throwit `RuntimeException` caught after
- ☐ Ⓓ hello throwit caught finally after ✔

**Your Answer:** Option Ⓑ

**Correct Answer:** Option Ⓓ

**Explanation:**

The `main()` method properly catches and handles the `RuntimeException` in the catch block, finally runs (as it always does), and then the code returns to normal.

A, B and C are incorrect based on the program logic described above. Remember that properly handled exceptions do not cause the program to stop executing.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Exceptions](#)

[#]

8. What will be the output of the program?

```
public class MyProgram
{
```

```
public static void main(String args[])
{
    try
    {
        System.out.print("Hello world ");
    }
    finally
    {
        System.out.println("Finally executing ");
    }
}
```

- ☐ Ⓐ Nothing. The program will not compile because no exceptions are specified.
- ☐ Ⓑ Nothing. The program will not compile because no catch clauses are specified.
- ☐ Ⓒ Hello world.
- ☒ Ⓓ Hello world Finally executing ✓

**Your Answer:** Option Ⓓ

**Correct Answer:** Option Ⓓ

**Explanation:**

Finally clauses are always executed. The program will first execute the **try** block, printing Hello world, and will then execute the finally block, printing Finally executing.

Option A, B, and C are incorrect based on the program logic described above. Remember that either a catch or a finally statement must follow a try. Since the finally is present, the catch is not required.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Exceptions](#)

[#]

9. Which interface provides the capability to store objects using a key-value pair?

- ☒ Ⓐ Java.util.Map ✓
- ☐ Ⓑ Java.util.Set
- ☐ Ⓒ Java.util.List
- ☐ Ⓓ Java.util.Collection

**Your Answer:** Option Ⓐ

**Correct Answer:** Option Ⓐ

**Explanation:**

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Objects and Collections](#)

[#]

10. What is the numerical range of char?

- ☐ Ⓐ 0 to 32767
- ☐ Ⓑ 0 to 65535 ✓
- ☐ Ⓒ -256 to 255
- ☒ Ⓓ -32768 to 32767 ✗

**Your Answer:** Option Ⓓ

**Correct Answer:** Option Ⓑ

**Explanation:**

The char type is integral but unsigned. The range of a variable of type char is from 0 to  $2^{16}-1$  or 0 to 65535. Java characters are Unicode, which is a 16-bit encoding capable of representing a wide range of international characters. If the most significant nine bits of a char are 0, then the encoding is the same as seven-bit ASCII.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Objects and Collections](#)

[#]

11. Which collection class allows you to grow or shrink its size and provides indexed access to its elements, but whose methods are not synchronized?

- ☐ Ⓐ java.util.HashSet
- ☐ Ⓑ java.util.LinkedHashSet
- ☐ Ⓒ java.util.List
- ☐ Ⓓ java.util.ArrayList ✓

**Your Answer:** Option (Not Answered)

**Correct Answer:** Option Ⓓ

**Explanation:**



All of the collection classes allow you to grow or shrink the size of your collection. **ArrayList** provides an index to its elements. The newer collection classes tend not to have synchronized methods. Vector is an older implementation of **ArrayList** functionality and has synchronized methods; it is slower than **ArrayList**.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Objects and Collections](#)

[#]

12. Which is true about an anonymous inner class?

- ☒ Ⓐ It can extend exactly one class and implement exactly one interface. ❌
- ☐ Ⓑ It can extend exactly one class and can implement multiple interfaces.
- ☐ Ⓒ It can extend exactly one class or implement exactly one interface. ✔️
- ☐ Ⓓ It can implement multiple interfaces regardless of whether it also extends a class.

**Your Answer:** Option Ⓐ

**Correct Answer:** Option Ⓒ

**Explanation:**

Option C is correct because the syntax of an anonymous inner class allows for only one named type after the new, and that type must be either a single interface (in which case the anonymous class implements that one interface) or a single class (in which case the anonymous class extends that one class).

Option A, B, D, and E are all incorrect because they don't follow the syntax rules described in the response for answer Option C.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Inner Classes](#)

[#]

13.

```
public class MyOuter
{
    public static class MyInner
    {
        public static void foo() { }
    }
}
```

which statement, if placed in a class other than **MyOuter** or **MyInner**, instantiates an instance of the nested class?

- ☐ Ⓐ `MyOuter.MyInner m = new MyOuter.MyInner();` ✓
- ☒ Ⓑ `MyOuter.MyInner mi = new MyInner();` ✗
- ☐ Ⓒ `MyOuter m = new MyOuter();`  
`MyOuter.MyInner mi = m.new MyOuter.MyInner();`
- ☐ Ⓓ `MyInner mi = new MyOuter.MyInner();`

**Your Answer:** Option Ⓑ

**Correct Answer:** Option Ⓐ

**Explanation:**

**MyInner** is a static nested class, so it must be instantiated using the fully-scoped name of *MyOuter.MyInner*.

Option B is incorrect because it doesn't use the enclosing name in the new.

Option C is incorrect because it uses incorrect syntax. When you instantiate a nested class by invoking new on an instance of the enclosing class, you do not use the enclosing name. The difference between Option A and C is that Option C is calling new on an instance of the enclosing class rather than just new by itself.

Option D is incorrect because it doesn't use the enclosing class name in the variable declaration.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Inner Classes](#)

[#]

14. What will be the output of the program?

```
public class Test
{
    public static void main (String [] args)
    {
        final Foo f = new Foo();
        Thread t = new Thread(new Runnable()
        {
            public void run()
            {
                f.doStuff();
            }
        });
        Thread g = new Thread()
        {
            public void run()
            {
                f.doStuff();
            }
        };
    }
}
```

```
        t.start();
        g.start();
    }
}
class Foo
{
    int x = 5;
    public void doStuff()
    {
        if (x < 10)
        {
            // nothing to do
            try
            {
                wait();
            } catch (InterruptedException ex) { }
        }
        else
        {
            System.out.println("x is " + x++);
            if (x >= 10)
            {
                notify();
            }
        }
    }
}
```

- ☐ Ⓐ The code will not compile because of an error on `notify();` of class `Foo`.
- ☐ Ⓑ The code will not compile because of some other error in class `Test`.
- ☐ Ⓒ An exception occurs at runtime. ✓
- ☐ Ⓓ It prints "x is 5 x is 6".

**Your Answer:** Option (Not Answered)

**Correct Answer:** Option Ⓒ

#### Explanation:

C is correct because the thread does not own the lock of the object it invokes `wait()` on. If the method were synchronized, the code would run without exception.

A, B are incorrect because the code compiles without errors.

D is incorrect because the exception is thrown before there is any output.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Threads](#)

[#]

15. Which statement is true?

- ☐ Ⓐ All objects that are eligible for garbage collection will be garbage collected by the garbage collector.
- ☐ Ⓑ Objects with at least one reference will never be garbage collected.
- ☐ Ⓒ Objects from a class with the `finalize()` method overridden will never be garbage collected.
- ☐ Ⓓ Objects instantiated within anonymous inner classes are placed in the garbage collectible heap. ✓

**Your Answer:** Option (Not Answered)

**Correct Answer:** Option Ⓓ

**Explanation:**

All objects are placed in the garbage collectible heap.

Option A is incorrect because the garbage collector makes no guarantees.

Option B is incorrect because islands of isolated objects can exist.

Option C is incorrect because `finalize()` has no such mystical powers.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Garbage Collections](#)

[#]

16. Which statement is true?

- ☐ Ⓐ Memory is reclaimed by calling `Runtime.gc()`.
- ☐ Ⓑ Objects are not collected if they are accessible from live threads. ✓
- ☐ Ⓒ An `OutOfMemory` error is only thrown if a single block of memory cannot be found that is large enough for a particular requirement.
- ☐ Ⓓ Objects that have `finalize()` methods always have their `finalize()` methods called before the program ends.

**Your Answer:** Option (Not Answered)

**Correct Answer:** Option Ⓑ

**Explanation:**

Option B is correct. If an object can be accessed from a live thread, it can't be garbage collected.

Option A is wrong. `Runtime.gc()` asks the garbage collector to run, but the garbage collector never makes any guarantees about when it will run or what unreachable objects it will free from memory.

Option C is wrong. The garbage collector runs immediately the system is out of memory before an **OutOfMemoryException** is thrown by the JVM.

Option D is wrong. If this were the case then the garbage collector would actively hang onto objects until a program finishes - this goes against the purpose of the garbage collector.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Garbage Collections](#)

[#]

17. Which statement is true?

- ☐ Ⓐ Programs will not run out of memory.
- ☐ Ⓑ Objects that will never again be used are eligible for garbage collection.
- ☐ Ⓒ Objects that are referred to by other objects will never be garbage collected.
- ☐ Ⓓ Objects that can be reached from a live thread will never be garbage collected. ✓

**Your Answer:** Option (Not Answered)

**Correct Answer:** Option Ⓓ

**Explanation:**

Option D is correct.

Option C is wrong. See the note above on Islands of Isolation (An object is eligible for garbage collection when no live thread can access it - even though there might be references to it).

Option B is wrong. "Never again be used" does not mean that there are no more references to the object.

Option A is wrong. Even though Java applications can run out of memory there another answer supplied that is more right.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Garbage Collections](#)

[#]

18. What will be the output of the program?

```
public class Test
{
    public static void main(String[] args)
    {
        int x = 0;
        assert (x > 0) ? "assertion failed" : "assertion passed" ;
        System.out.println("finished");
    }
}
```

```
}
```

- ☒ Ⓐ finished ✖
- ☐ Ⓑ Compilation fails. ✔
- ☐ Ⓒ An **AssertionError** is thrown and finished is output.
- ☐ Ⓓ An **AssertionError** is thrown with the message "assertion failed."

**Your Answer:** Option Ⓐ

**Correct Answer:** Option Ⓑ

**Explanation:**

Compilation Fails. You can't use the Assert statement in a similar way to the ternary operator. Don't confuse.

Discuss about this problem : [Discuss in Forum](#)

Learn more problems on : [Assertions](#)

[#]

19. What will be the output of the program?

```
interface Foo141
{
    int k = 0; /* Line 3 */
}
public class Test141 implements Foo141
{
    public static void main(String args[])
    {
        int i;
        Test141 test141 = new Test141();
        i = test141.k; /* Line 11 */
        i = Test141.k;
        i = Foo141.k;
    }
}
```

- ☐ Ⓐ Compilation fails.
- ☐ Ⓑ Compiles and runs ok. ✔
- ☐ Ⓒ Compiles but throws an Exception at runtime.
- ☒ Ⓓ Compiles but throws a **RuntimeException** at runtime. ✖

**Your Answer:** Option Ⓓ

**Correct Answer:** Option Ⓑ

**Explanation:**

The variable **k** on line 3 is an interface constant, it is implicitly **public**, **static**, and **final**. **Static** variables can be referenced in two ways:

Via a reference to any instance of the class (line 11)

Via the class name (line 12).



[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Java.lang Class](#)

[\[#\]](#)

20. What will be the output of the program?

```
String s = "ABC";  
s.toLowerCase();  
s += "def";  
System.out.println(s);
```

- ☒ **A** ABC 
- ☐ **B** abc
- ☐ **C** ABCdef 
- ☐ **D** Compile Error

**Your Answer:** Option **A**

**Correct Answer:** Option **C**

**Explanation:**

**String** objects are immutable. The object **s** above is set to " **ABC** ". Now ask yourself if this object is changed and if so where - remember strings are immutable.

Line 2 returns a string object but does not change the originag string object **s**, so after line 2 **s** is still " **ABC** ".

So what's happening on line 3? Java will treat line 3 like the following:

```
s = new StringBuffer().append(s).append("def").toString();
```

This effectively creates a new **String** object and stores its reference in the variable **s**, the old **String** object containing " **ABC** " is no longer referenced by a live thread and becomes available for garbage collection.

[Discuss about this problem : Discuss in Forum](#)

[Learn more problems on : Java.lang Class](#)

[\[#\]](#)

\*\*\* END OF THE TEST \*\*\*

Post your test result / feedback here:

Quality of the Test :

Difficulty of the Test :

 Feedback

 Full Name

 Email (optional)



### Current Affairs

Check out the latest current affairs questions and answers.

### Quick links

#### Quantitative Aptitude

- › Arithmetic
- › Data Interpretation

#### Verbal (English)

- › Verbal Ability
- › Verbal Test

#### Reasoning



- › Logical
- › Verbal
- › Nonverbal

### Programming

- › C Programming
- › C++
- › C#
- › Java

### Interview

- › GD
- › HR
- › Technical Interview

### Placement Papers

- › Placement Papers
- › Submit Paper