# Java

### 1. What is the difference between a constructor and a method?

A constructor is a member function of a class that is used to create objects of that class. It has the same name as the class itself, has no return type, and is invoked using the new operator.

A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the dot operator.

### 2. What is the purpose of garbage collection in Java, and when is it used?

The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused.

A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used.

### 3. Describe synchronization in respect to multithreading.

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources.

Without synchonization, it is possible for one thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to significant errors.

### 4. What is an abstract class?

Abstract class must be extended/subclassed (to be useful). It serves as a template. A class that is abstract may not be instantiated (ie. you may not call its constructor), abstract class may contain static data.

Any class with an abstract method is automatically abstract itself, and must be declared as such. A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated.

### 5. What is the difference between an Interface and an Abstract class?

An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract.

An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (`private, protected`, etc.), but has some abstract methods.

**6. Explain different way of using thread?**

The thread could be implemented by using runnable interface or by inheriting from the `Thread` class. The former is more advantageous, 'cause when you are going for multiple inheritance, the only interface can help.

**7. What is an Iterator?**

Some of the collection classes provide traversal of their contents via a `java.util.Iterator` interface. This interface allows you to walk through a collection of objects, operating on each object in turn.

Remember when using Iterators that they contain a snapshot of the collection at the time the Iterator was obtained; generally it is not advisable to modify the collection itself while traversing an Iterator.

**8. State the significance of public, private, protected, default modifiers both singly and in combination and state the effect of package relationships on declared items qualified by these modifiers.**

`public:` Public class is visible in other packages, field is visible everywhere (class must be public too)

`private :` Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that owns the feature.

`protected :` Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature. This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.

What you get by default ie, without any access modifier (ie, public private or protected). It means that it is visible to all within a particular package.

**9. What is static in java?**

Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class.Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object.
A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a nonstatic method. In other words, you can't change a static method into an instance method in a subclass.

**10. What is final class?**

A `final` class can't be extended ie., final class may not be subclassed. A final method can't be overridden when its class is inherited. You can't change value of a final variable (is a constant).

**11. What if the main() method is declared as private?**

The program compiles properly but at runtime it will give "`main()` method not public." message.

**12. What if the static modifier is removed from the signature of the main() method?**

Program compiles. But at runtime throws an error "NoSuchMethodError".

**13. What if I write static public void instead of public static void?**

Program compiles and runs properly.

**14. What if I do not provide the String array as the argument to the method?**

Program compiles but throws a runtime error "NoSuchMethodError".

**15. What is the first argument of the String array in main() method?**

The String array is empty. It does not have any element. This is unlike C/C++ where the first element by default is the program name.

**16. If I do not provide any arguments on the command line, then the String array of main() method will be empty or null?**

It is empty. But not null.

**17. How can one prove that the array is not null but empty using one line of code?**

Print `args.length`. It will print 0. That means it is empty. But if it would have been null then it would have thrown a`NullPointerException` on attempting to print `args.length`.

**18. What environment variables do I need to set on my machine in order to be able to run Java programs?**

`CLASSPATH` and `PATH` are the two variables.

**19. Can an application have multiple classes having main() method?**

Yes it is possible. While starting the application we mention the class name to be run. The JVM will look for the Main method only in the class whose name you have mentioned.
Hence there is not conflict amongst the multiple classes having `main()` method.

**20. Can I have multiple main() methods in the same class?**

No the program fails to compile. The compiler says that the `main()` method is already defined in the class.

**21. Do I need to import java.lang package any time? Why ?**

No. It is by default loaded internally by the JVM.

**22. Can I import same package/class twice? Will the JVM load the package twice at runtime?**

One can import the same package or same class multiple times. Neither compiler nor JVM complains about it. And the JVM will internally load the class only once no matter how many times you import the same class.

**23. What are Checked and UnChecked Exception?**

A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Making an exception checked forces client programmers to deal with the possibility that the exception will be thrown.

Example: `IOException` thrown by `java.io.FileInputStream`'s `read()` method·

Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked. With an unchecked exception, however, the compiler doesn't force client programmers either to catch the exception or declare it in a throws clause. In fact, client programmers may not even know that the exception could be thrown.

Example: `StringIndexOutOfBoundsException` thrown by String's `charAt()` method· Checked exceptions must be caught at compile time. Runtime exceptions do not need to be. Errors often cannot be.

**24. What is Overriding?**

When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class overrides the method in the superclass.

When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass. Methods may be overridden to be more public, not more private.

**25. Are the imports checked for validity at compile time? Example: will the code containing an import such as java.lang.ABCD compile?**

Yes the imports are checked for the semantic validity at compile time. The code containing above line of import will not compile. It will throw an error saying, can not resolve symbol

symbol : `class ABCD`

location: `package io`

`import java.io.ABCD;`

# Java Based Interview Questions

## 1. What is the difference between a constructor and a method?

A constructor is a member function of a class that is used to create objects of that class. It has the same name as the class itself, has no return type, and is invoked using the new operator.

A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the dot operator.

## 2. What is the purpose of garbage collection in Java, and when is it used?

The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused.

A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used.

## 3. Describe synchronization in respect to multithreading.

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources.

Without synchonization, it is possible for one thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to significant errors.

## 4. What is an abstract class?

Abstract class must be extended/subclassed (to be useful). It serves as a template. A class that is abstract may not be instantiated (ie. you may not call its constructor), abstract class may contain static data.

Any class with an abstract method is automatically abstract itself, and must be declared as such. A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated.

## 5. What is the difference between an Interface and an Abstract class?

An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract.

An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

## 6. Explain different way of using thread?

The thread could be implemented by using runnable interface or by inheriting from the Thread class. The former is more advantageous, 'cause when you are going for multiple inheritance, the only interface can help.

**7. What is an Iterator?**

Some of the collection classes provide traversal of their contents via a java.util.Iterator interface. This interface allows you to walk through a collection of objects, operating on each object in turn.

Remember when using Iterators that they contain a snapshot of the collection at the time the Iterator was obtained; generally it is not advisable to modify the collection itself while traversing an Iterator.

**8. State the significance of public, private, protected, default modifiers both singly and in combination and state the effect of package relationships on declared items qualified by these modifiers.**

public: Public class is visible in other packages, field is visible everywhere (class must be public too)

private : Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that owns the feature.

protected : Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature. This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.

What you get by default ie, without any access modifier (ie, public private or protected). It means that it is visible to all within a particular package.

**9. What is static in java?**

Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class.Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object.

A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a nonstatic method. In other words, you can't change a static method into an instance method in a subclass.

**10. What is final class?**

A final class can't be extended ie., final class may not be subclassed. A final method can't be overridden when its class is inherited. You can't change value of a final variable (is a constant).

**11. What if the mainmethod is declared as private?**

The program compiles properly but at runtime it will give "main() method not public." message.

**12. What if the static modifier is removed from the signature of the main() method?**

Program compiles. But at runtime throws an error "NoSuchMethodError".

**13. What if I write static public void instead of public static void?**

Program compiles and runs properly.

**14. What if I do not provide the String array as the argument to the method?**

Program compiles but throws a runtime error "NoSuchMethodError".

**15. What is the first argument of the String array in main() method?**

The String array is empty. It does not have any element. This is unlike C/C++ where the first element by default is the program name.

**16. If I do not provide any arguments on the command line, then the String array of main() method will be empty or null?**

It is empty. But not null.

**17. How can one prove that the array is not null but empty using one line of code?**

Print args.length. It will print 0. That means it is empty. But if it would have been null then it would have thrown a NullPointerException on attempting to print args.length.

**18. What environment variables do I need to set on my machine in order to be able to run Java programs?**

CLASSPATH and PATH are the two variables.

1. What will be the output of the program?

```
classA
{
   finalpublicintGetResult(int a, int b) { return0; }
}
classBextendsA
{
   publicintGetResult(int a, int b) {return1; }
}
publicclassTest
{
   publicstaticvoid main(String args[])
   {
      B b = new B();
      System.out.println("x = " + b.GetResult(0, 1));
   }
}
```

   A.    x = 0

   B.    x = 1

   C.    Compilation fails.

   D.    An exception is thrown at runtime.

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

**Explanation:**

The code doesn't compile because the method GetResult() in class A is final and so cannot be overridden.

2. What will be the output of the program?

```
classSC2
{
   publicstaticvoid main(String [] args)
   {
      SC2 s = new SC2();
      s.start();
   }

   void start()
   {
      int a = 3;
      int b = 4;
      System.out.print(" " + 7 + 2 + " ");
      System.out.print(a + b);
      System.out.print(" " + a + b + " ");
      System.out.print(foo() + a + b + " ");
      System.out.println(a + b + foo());
```

```
  }

  String foo()
  {
    return"foo";
  }
}
```

    A.    9 7 7 foo 7 7foo

    B.    72 34 34 foo34 34foo

    C.    9 7 7 foo34 34foo

    D.    72 7 34 foo34 7foo

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

**Explanation:**

Because all of these expressions use the + operator, there is no precedence to worry about and all of the expressions will be evaluated from left to right. If either operand being evaluated is a String, the + operator will concatenate the two operands; if both operands are numeric, the + operator will add the two operands.

3.  What will be the output of the program?

```
classBoolArray
{
  boolean [] b = newboolean[3];
  int count = 0;

  void set(boolean [] x, inti)
  {
    x[i] = true;
    ++count;
  }

  publicstaticvoid main(String [] args)
  {
    BoolArrayba = newBoolArray();
    ba.set(ba.b, 0);
    ba.set(ba.b, 2);
    ba.test();
  }

  void test()
  {
    if ( b[0] && b[1] | b[2] )
       count++;
    if ( b[1] && b[(++count - 2)] )
       count += 7;
    System.out.println("count = " + count);
```

```
    }
}
```

|    |                |    |                |
|----|----------------|----|----------------|
| A. | count = 0      | B. | count = 2      |
| C. | count = 3      | D. | count = 4      |

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

**Explanation:**

The reference variables b and x both refer to the same boolean array. count is incremented for each call to the set() method, and once again when the first if test istrue. Because of the && short circuit operator, count is not incremented during the second if test.

4. Which two statements are equivalent?

   1. 3/2
   2. 3<2
   3. 3*4
   4. 3<<2

|    |          |    |          |
|----|----------|----|----------|
| A. | 1 and 2  | B. | 2 and 3  |
| C. | 3 and 4  | D. | 1 and 4  |

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

**Explanation:**

(1) is wrong. 3/2 = 1 (integer arithmetic).

(2) is wrong. 3 < 2 = false.

(3) is correct. 3 * 4 = 12.

(4) is correct. 3 <<2= 12. In binary 3 is 11, now shift the bits two places to the left and we get 1100 which is 12 in binary (3*2*2).

5. 
```
publicvoid foo( boolean a, boolean b)
{
   if( a )
   {
      System.out.println("A"); /* Line 5 */
   }
   elseif(a && b) /* Line 7 */
   {
      System.out.println( "A && B");
   }
   else/* Line 11 */
   {
```

```
       if ( !b )
       {
          System.out.println( "notB") ;
       }
       else
       {
          System.out.println( "ELSE" ) ;
       }
    }
}
```

A. If a is true and b is true then the output is "A && B"

B. If a is true and b is false then the output is "notB"

C. If a is false and b is true then the output is "ELSE"

D. If a is false and b is false then the output is "ELSE"

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

**Explanation:**

Option C is correct. The output is "ELSE". Only when a is false do the output lines after 11 get some chance of executing.

Option A is wrong. The output is "A". When a is true, irrespective of the value of b, only the line 5 output will be executed. The condition at line 7 will never be evaluated (when a is true it will always be trapped by the line 12 condition) therefore the output will never be "A && B".

Option B is wrong. The output is "A". When a is true, irrespective of the value of b, only the line 5 output will be executed.

Option D is wrong. The output is "notB".

6. What will be the output of the program?

```
Float f = new Float("12");
switch (f)
{
   case12: System.out.println("Twelve");
   case0: System.out.println("Zero");
   default: System.out.println("Default");
}
```

A. Zero

B. Twelve

C. Default

D. Compilation fails

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

**Explanation:**

The switch statement can only be supported by integers or variables more "narrow" than an integer i.e. byte, char, short. Here a Float wrapper object is used and so the compilation fails.

7. What will be the output of the program?

```
publicclassTest
{
   publicstaticvoidaMethod() throws Exception
   {
      try/* Line 5 */
      {
         thrownew Exception(); /* Line 7 */
      }
      finally/* Line 9 */
      {
         System.out.print("finally "); /* Line 11 */
      }
   }
   publicstaticvoid main(String args[])
   {
      try
      {
         aMethod();
      }
      catch (Exception e) /* Line 20 */
      {
         System.out.print("exception ");
      }
      System.out.print("finished"); /* Line 24 */
   }
}
```

    A.    finally

    B.    exception finished

    C.    finally exception finished

    D.    Compilation fails

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

**Explanation:**

This is what happens:

(1) The execution of the try block (line 5) completes abruptly because of the throwstatement (line 7).

(2) The exception cannot be assigned to the parameter of any catch clause of the trystatement therefore the finally block is executed (line 9) and "finally" is output (line 11).

(3) The finally block completes normally, and then the try statement completes abruptly because of the throw statement (line 7).

(4) The exception is propagated up the call stack and is caught by the catch in the main method (line 20). This prints "exception".

(5) Lastly program execution continues, because the exception has been caught, and "finished" is output (line 24).

8.  Which statement is true for the class java.util.ArrayList?

   A.   The elements in the collection are ordered.

   B.   The collection is guaranteed to be immutable.

   C.   The elements in the collection are guaranteed to be unique.

   D.   The elements in the collection are accessed using a unique key.

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

   **Explanation:**

   Yes, always the elements in the collection are ordered.

9.  Which is true about a method-local inner class?

   A.   It must be marked final.

   B.   It can be marked abstract.

   C.   It can be marked public.

   D.   It can be marked static.

   **Answer:** Option **B**

   **Explanation:**

   Option B is correct because a method-local inner class can be abstract, although it means a subclass of the inner class must be created if the abstract class is to be used (so an abstract method-local inner class is probably not useful).

   Option A is incorrect because a method-local inner class does not have to be declaredfinal (although it is legal to do so).

   C and D are incorrect because a method-local inner class cannot be made public(remember-you cannot mark any local variables as public), or static.

10. classXimplementsRunnable
    {
       publicstaticvoid main(String args[])

```
    {
       /* Missing code? */
    }
    publicvoid run() {}
}
```

Which of the following line of code is suitable to start a thread ?

   A.    Thread t = new Thread(X);

   B.    Thread t = new Thread(X); t.start();

   C.    X run = new X(); Thread t = new Thread(run); t.start();

   D.    Thread t = new Thread(); x.run();

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

**Explanation:**

Option C is suitable to start a thread.

11. What will be the output of the program?

```
classMyThreadextendsThread
{
    publicstaticvoid main(String [] args)
    {
        MyThread t = newMyThread();
        t.start();
        System.out.print("one. ");
        t.start();
        System.out.print("two. ");
    }
    publicvoid run()
    {
        System.out.print("Thread ");
    }
}
```

   A.    Compilation fails

   B.    An exception occurs at runtime.

   C.    It prints "Thread one. Thread two."

   D.    The output cannot be determined.

**Answer:** Option **B**

**Explanation:**

When the start() method is attempted a second time on a single Thread object, the method will throw an IllegalThreadStateException (you will not need to know this exception name for the exam). Even if the thread has finished running, it is still illegal to call start() again.

12. What will be the output of the program?

```
classMyThreadextendsThread
{
   MyThread() {}
   MyThread(Runnable r) {super(r); }
   publicvoid run()
   {
      System.out.print("Inside Thread ");
   }
}
classMyRunnableimplementsRunnable
{
   publicvoid run()
   {
      System.out.print(" Inside Runnable");
   }
}
classTest
{
   publicstaticvoid main(String[] args)
   {
      newMyThread().start();
      newMyThread(newMyRunnable()).start();
   }
}
```

    A.    Prints "Inside Thread Inside Thread"

    B.    Prints "Inside Thread Inside Runnable"

    C.    Does not compile

    D.    Throws exception at runtime

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

**Explanation:**

If a Runnable object is passed to the Thread constructor, then the run method of theThread class will invoke the run method of the Runnable object.

In this case, however, the run method in the Thread class is overridden by the run method in MyThread class. Therefore the run() method in MyRunnable is never invoked.

Both times, the run() method in MyThread is invoked instead.

13. What will be the output of the program?

```
classsimplementsRunnable
{
    int x, y;
    publicvoid run()
    {
        for(inti = 0; i<1000; i++)
            synchronized(this)
            {
                x = 12;
                y = 12;
            }
        System.out.print(x + " " + y + " ");
    }
    publicstaticvoid main(String args[])
    {
        s run = new s();
        Thread t1 = new Thread(run);
        Thread t2 = new Thread(run);
        t1.start();
        t2.start();
    }
}
```

| | | | |
|---|---|---|---|
| A. | DeadLock | B. | It print 12 12 12 12 |
| C. | Compilation Error | D. | Cannot determine output. |

**Answer:** Option **B**

**Explanation:**

The program will execute without any problems and print 12 12 12 12.

14. 
```
publicclassTest
{
    publicvoid foo()
    {
        assertfalse; /* Line 5 */
        assertfalse; /* Line 6 */
    }
    publicvoid bar()
    {
        while(true)
        {
            assertfalse; /* Line 12 */
        }
        assertfalse;  /* Line 14 */
    }
}
```
What causes compilation to fail?

    A.   Line 5

B. Line 6

C. Line 12

D. Line 14

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

**Explanation:**

Option D is correct. Compilation fails because of an unreachable statement at line 14. It is a compile-time error if a statement cannot be executed because it is unreachable. The question is now, why is line 20 unreachable? If it is because of the assert then surely line 6 would also be unreachable. The answer must be something other than assert.

Examine the following:

A while statement can complete normally if and only if at least one of the following is true:

- The while statement is reachable and the condition expression is not a constant expression with value true.

-There is a reachable break statement that exits the while statement.

The while statement at line 11 is infinite and there is no break statement therefore line 14 is unreachable. You can test this with the following code:

```
publicclassTest80
{
    publicvoid foo()
    {
        assertfalse;
        assertfalse;
    }
    publicvoid bar()
    {
        while(true)
        {
            assertfalse;
            break;
        }
        assertfalse;
    }
}
```

15. What will be the output of the program?

```
publicclassTest
{
    publicstaticint y;
    publicstaticvoid foo(int x)
    {
```

```
      System.out.print("foo ");
      y = x;
   }
   publicstaticint bar(int z)
   {
      System.out.print("bar ");
      return y = z;
   }
   publicstaticvoid main(String [] args )
   {
      int t = 0;
      assert t >0 : bar(7);
      assert t >1 : foo(8); /* Line 18 */
      System.out.println("done ");
   }
}
```

A.  bar

B.  bar done

C.  foo done

D.  Compilation fails

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

**Explanation:**

The foo() method returns void. It is a perfectly acceptable method, but because it returns void it cannot be used in an assert statement, so line 18 will not compile.

16. Which of the following statements is true?

A.  In an assert statement, the expression after the colon ( : ) can be any Java expression.

B.  If a switch block has no default, adding an assert default is considered appropriate.

C.  In an assert statement, if the expression after the colon ( : ) does not have a value, the assert's error message will be empty.

D.  It is appropriate to handle assertion failures using a catch clause.

**Answer:** Option **B**

**Explanation:**

Adding an assertion statement to a switch statement that previously had no default case is considered an excellent use of the assert mechanism.

Option A is incorrect because only Java expressions that return a value can be used. For instance, a

method that returns void is illegal.

Option C is incorrect because the expression after the colon must have a value.

Option D is incorrect because assertions throw errors and not exceptions, and assertion errors do cause program termination and should not be handled.

17. 
```
publicclassTest2
{
    publicstaticint x;
    publicstaticint foo(int y)
    {
        return y * 2;
    }
    publicstaticvoid main(String [] args)
    {
        int z = 5;
        assert z >0; /* Line 11 */
        assert z >2: foo(z); /* Line 12 */
        if ( z <7 )
            assert z >4; /* Line 14 */

        switch (z)
        {
            case4: System.out.println("4 ");
            case5: System.out.println("5 ");
            default: assert z <10;
        }

        if ( z <10 )
            assert z >4: z++; /* Line 22 */
        System.out.println(z);
    }
}
```
which line is an example of an inappropriate use of assertions?

    A.    Line 11

    B.    Line 12

    C.    Line 14

    D.    Line 22

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

**Explanation:**

Assert statements should not cause side effects. Line 22 changes the value of z if the assert statement is false.

Option A is fine; a second expression in an assert statement is not required.

Option B is fine because it is perfectly acceptable to call a method with the second expression of an assert statement.

Option C is fine because it is proper to call an assert statement conditionally.

18. What will be the output of the program?

```
publicclassNFE
{
  publicstaticvoid main(String [] args)
  {
  String s = "42";
    try
    {
      s = s.concat(".5");  /* Line 8 */
      double d = Double.parseDouble(s);
      s = Double.toString(d);
      int x = (int) Math.ceil(Double.valueOf(s).doubleValue());
      System.out.println(x);
    }
    catch (NumberFormatException e)
    {
      System.out.println("bad number");
    }
  }
}
```

A.   42

B.   42.5

C.   43

D.   bad number

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

**Explanation:**

All of this code is legal, and line 8 creates a new String with a value of "42.5". Lines 9 and 10 convert the String to a double and then back again. Line 11 is funâ€'Math.ceil()'s argument expression is evaluated first. We invoke the valueOf()method that returns an anonymous Double object (with a value of 42.5). Then thedoubleValue() method is called (invoked on the newly created Double object), and returns a double primitive (there and back again), with a value of (you guessed it) 42.5. The ceil() method converts this to 43.0, which is cast to an int and assigned to x.

19. What will be the output of the program?

```
publicclassTest138
{
  publicstaticvoidstringReplace (String text)
  {
    text = text.replace ('j' , 'c'); /* Line 5 */
  }
```

```
publicstaticvoidbufferReplace (StringBuffer text)
{
    text = text.append ("c");  /* Line 9 */
}
publicstaticvoid main (String args[])
{
    String textString = new String ("java");
    StringBuffertextBuffer = newStringBuffer ("java"); /* Line 14 */
    stringReplace(textString);
    bufferReplace(textBuffer);
    System.out.println (textString + textBuffer);
}
}
```

    A.    java

    B.    javac

    C.    javajavac

    D.    Compile error

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

**Explanation:**

A string is immutable, it cannot be changed, that's the reason for the StringBufferclass.
The stringReplace method does not change the string declared on line 14, so this remains set to "java".

Method parameters are always passed by value - a copy is passed into the method - if the copy changes, the original remains intact, line 5 changes the reference i.e. text points to a new String object, however this is lost when the method completes. ThetextBuffer is a StringBuffer so it can be changed.

This change is carried out on line 9, so "java" becomes "javac", the text reference on line 9 remains unchanged. This gives us the output of "javajavac"

20. What will be the output of the program (in jdk1.6 or above)?

```
publicclassBoolTest
{
    publicstaticvoid main(String [] args)
    {
        Boolean b1 = new Boolean("false");
        boolean b2;
        b2 = b1.booleanValue();
        if (!b2)
        {
            b2 = true;
            System.out.print("x ");
        }
        if (b1 & b2) /* Line 13 */
```

```
      {
         System.out.print("y ");
      }
      System.out.println("z");
   }
}
```

A.    z

B.    x z

C.    y z

D.    Compilation fails.

**Answer:** Option **B**