Feel free to contact **santhosh_arena@yahoo.co.in**

Collective Info.

santhosh

SCJP java Programmer certification Exam And Training. This site is entirely independent of Sun Microsystems Inc, The Sun Certified Java Programmers Exam (SCJP) is the internationally recognized java certification exam. I am offering free mock Exam preparation questions, practice tests, tutorials, certification faq and sample code. This page contains a very detailed tutorial organized by topic. At the end of the tutorial you can learn from your mistakes and apply your concepts on the free mock exams java certification practice tests.

Chapter 1
Language Fundamentals

The Java programming language has includes five simple arithmetic operators like are
+ (addition),
- (subtraction),
* (multiplication),
/ (division), and
% (modulo).
The following table summarizes them:

1. Source file's elements (in order)
   Package declaration
   Import statements
   Class definitions

2. Importing packages doesn't recursively import sub-packages.

3. Sub-packages are really different packages, happen to live within an enclosing package. Classes in sub-packages cannot access classes in enclosing package with default access.

4. Comments can appear anywhere. Can't be nested. No matter what type of comments.

5. At most one public class definition per file. This class name should match the file name. If there are more than one public class definitions, compiler will accept the class with the file's name and give an error at the line where the other class is defined.

6. It's not required having a public class definition in a file. Strange, but true. J In this case, the file's name should be different from the names of classes and interfaces (not public obviously).

7. Even an empty file is a valid source file.

8. An identifier must begin with a letter, dollar sign ($) or underscore (_). Subsequent characters may be letters, $, _ or digits.

9. An identifier cannot have a name of a Java keyword. Embedded keywords are OK. true, false and null are literals (not keywords), but they can't be used as identifiers as well.

10. const and goto are reserved words, but not used.

11. Unicode characters can appear anywhere in the source code. The following code is valid.

```
ch\u0061r a = 'a';

char \u0062 = 'b';

char c = '\u0063';
```

12. Java has 8 primitive data types.

Data Type	Size (bits)	Initial Value	Min Value	Max Value
boolean	1	false	false	true
byte	8	0	-128 (-27)	127 (27 - 1)

short	16	0	-215	215 - 1
char	16	'\u0000'	'\u0000' (0)	'\uFFFF' (216 - 1)
int	32	0	-231	231 - 1
long	64	0L	-263	263 - 1
float	32	0.0F	1.4E-45	3.4028235E38
double	64	0.0	4.9E-324	1.7976931348623157E308

13. All numeric data types are signed. char is the only unsigned integral type.

14. Object reference variables are initialized to null.

15. Octal literals begin with zero. Hex literals begin with 0X or 0x.

16. Char literals are single quoted characters or unicode values (begin with \u).

17. A number is by default an int literal, a decimal number is by default a double literal.

18. 1E-5d is a valid double literal, E2d is not (since it starts with a letter, compiler thinks that it's an identifier)

19. Two types of variables.

a. Member variables

Accessible anywhere in the class.
Automatically initialized before invoking any constructor.
Static variables are initialized at class load time.
Can have the same name as the class.

b. Automatic variables method local

· Must be initialized explicitly. (Or, compiler will catch it.) Object references can be initialized to null to make the compiler happy. The following code won't compile. Specify else part or initialize the local variable explicitly.

```
public String testMethod ( int a) {

String tmp;

if ( a > 0 ) tmp = "Positive";

return tmp;

}
```

· Can have the same name as a member variable, resolution is based on scope.

20. Arrays are Java objects. **If you create an array of 5 Strings, there will be 6 objects created.**

21. Arrays should be
- Declared. (int[ ] a; String b[ ]; Object [ ]c; Size should not be specified now)
- Allocated (constructed). (a = new int[10]; c = new String[arraysize])
- Initialized. for (int i = 0; i < a.length; a[i++] = 0)

22. The above three can be done in one step.

```
int a[ ] = { 1, 2, 3 }; (or )

int a[ ] = new int[ ] { 1, 2, 3 };
```
But never specify the size with the new statement.

23. Java arrays are static arrays. Size has to be specified at compile time. Array.length returns array's size. (Use Vectors for dynamic purposes).

24. Array size is never specified with the reference variable, it is always maintained with the array object. It is maintained in array.length, which is a final instance variable.

25. Anonymous arrays can be created and used like this: new int[ ] {1,2,3} or new int[10]

26. Arrays with zero elements can be created. args array to the main method will be a zero element array if no command parameters are specified. In this case args.length is 0.

27. Comma after the last initializer in array declaration is ignored.

```
int[ ] i = new int[2] { 5, 10}; // Wrong

int i[5] = { 1, 2, 3, 4, 5}; // Wrong

int[ ] i[ ] = {{}, new int[ ] {} }; // Correct

int i[ ][ ] = { {1,2}, new int[2] }; // Correct

int i[ ] = { 1, 2, 3, 4, } ; // Correct
```

28. Array indexes start with 0. Index is an int data type.

29. Square brackets can come after datatype or before/after variable name. White spaces are fine. Compiler just ignores them.

30. Arrays declared even as member variables also need to be allocated memory explicitly.

```
static int a[ ];

static int b[ ] = {1,2,3};

public static void main(String s[]) {

        System.out.println(a[0]); // Throws a null pointer exception

        System.out.println(b[0]); // This code runs fine

        System.out.println(a); // Prints 'null'

        System.out.println(b); // Prints a string which is returned by toString

        }
```

31. Once declared and allocated (even for local arrays inside methods), array elements are automatically initialized to the default values.

32. If only declared (not constructed), member array variables default to null, but local array variables will not default to null.

33. Java doesn't support multidimensional arrays formally, but it supports arrays of arrays. From the specification - "The number of bracket pairs indicates the depth of array nesting." So this can perform as a multidimensional array. (no limit to levels of array nesting)

34. In order to be run by JVM, a class should have a main method with the following signature.

```
public static void main(String args[])

static public void main(String[] s)
```

35. args array's name is not important. args[0] is the first argument. args.length gives no. of arguments.

36. main method can be overloaded.

37. main method can be final.

38. A class with a different main signature or w/o main method will compile. **But throws a runtime error.**

39. A class without a main method can be run by JVM, if its ancestor class has a main method. (main is just a method and is inherited)

40. Primitives are passed by value.

41. Objects (references) are passed by reference. The object reference itself is passed by value. So, it can't be changed. But, the object can be changed via the reference.

42. Garbage collection is a mechanism for reclaiming memory from objects that are no longer in use, and making the memory available for new objects.

43. An object being no longer in use means that it can't be referenced by any 'active' part of the program.

44. Garbage collection runs in a low priority thread. It may kick in when memory is too low. No guarantee.

45. It's not possible to force garbage collection. Invoking System.gc may start garbage collection process.

46. The automatic garbage collection scheme guarantees that a reference to an object is always valid while the object is in use, i.e. the object will not be deleted leaving the reference "dangling".

47. There are no guarantees that the objects no longer in use will be garbage collected and their finalizers executed at all. gc might not even be run if the program execution does not warrant it. Thus any memory allocated during program execution might remain allocated after program termination, unless reclaimed by the OS or by other means.

48. There are also no guarantees on the order in which the objects will be garbage collected or on the order in which the finalizers are called. Therefore, the program should not make any decisions based on these assumptions.

49. An object is only eligible for garbage collection, if the only references to the object are from other objects that are also eligible for garbage collection. That is, an object can become eligible for garbage collection even if there are references pointing to the object, as long as the objects with the references are also eligible for garbage collection.

50. Circular references do not prevent objects from being garbage collected.

51. We can set the reference variables to null, hinting the gc to garbage collect the objects referred by the variables. Even if we do that, the object may not be gc-ed if it's attached to a listener. (Typical in case of AWT components) Remember to remove the listener first.

52. All objects have a finalize method. It is inherited from the Object class.

53. finalize method is used to release system resources other than memory. (such as file handles and network connections) The order in which finalize methods are called may not reflect the order in which objects are created. Don't rely on it. This is the signature of the finalize method.

    protected void finalize() throws Throwable { }

    In the descendents this method can be protected or public. Descendents can restrict the exception list that can be thrown by this method.

54. finalize is called only once for an object. If any exception is thrown in finalize, the object is still eligible for garbage collection (at the discretion of gc)

55. gc keeps track of unreachable objects and garbage-collects them, but an unreachable object can become reachable again by letting know other objects of its existence from its finalize method (when called by gc). This 'resurrection' can be done only once, since finalize is called only one for an object.

56. finalize can be called explicitly, but it does not garbage collect the object.

57. finalize can be overloaded, but only the method with original finalize signature will be called by gc.

58. finalize is not implicitly chained. A finalize method in sub-class should call finalize in super class explicitly as its last action for proper functioning. But compiler doesn't enforce this check.

59. System.runFinalization can be used to run the finalizers (which have not been executed before) for the objects eligible for garbage collection.