100 Tips in Java

By

Vertical Horizons
(http://verticalhorizons.in)

1. All operands are evaluated from left to right.

2. Collection is an Interface where as Collections is a helper class.

3. Map doesn't implement Collection.

4. Only in Hashtable null is not allowed.

5. java.util.Map reject duplicates.

6. wait(long) method can throw InterruptedException, IllegalArgumentException, IllegalMonitorStateException.

7. wait() method can throw InterruptedException, IllegalMonitorStateException.

8. When the thread has slept it doesn't lose the ownership of any object while when wait is called the thread loses the ownership.

9. If runnable interface is implemented by a class and run() is not implemented properly then a compilation error will occur at the class declaration itself.

10. If run() method is called directly instead of calling the start() method, it will behave as a single thread of execution, just like any other method call, i.e., no thread will be created.

11. If interrupt method is invoked on a sleeping thread, the thread moves to ready state. The next time it begins running, it executes the InterruptedException handler.

12. wait, notify and notifyAll methods are not called on Thread, they're called on Object. Because the object is the one which controls the threads in this case. It asks the threads to wait and then notifies when its state changes. It's called a monitor.

13. wait – points to remember
    - calling thread gives up CPU
    - calling thread gives up the lock
    - calling thread goes to monitor's waiting pool

14. notify – points to remember
    - one thread gets moved out of monitor's waiting pool to ready state
    - notifyAll moves all the threads to ready state

15. Class locks control the static methods.

16. Deadly embrace is a scenario, where one thread has a lock on object A and waiting to acquire lock on object B. At the same time, another thread has a lock on object B and waiting to acquire lock on object A.

17. Synchronized methods can be overridden to be non-synchronous. Synchronized behavior affects only the original class.

18. Synchronized keyword can be applied to methods or parts of methods only.

19. If there are multiple threads in the thread pool waiting on the same object, there is no guarantee which thread will get the lock of the object after issuing a notify() or notifyAll() and become runnable.

20. There can be scenarios where a thread never gets a chance to become runnable.

21. Increasing the priority of a thread though not ensures but makes the chances of becoming runnable better once a notify() or notifyAll() is issued.

22. Attributes are not overridden.

23. Private methods are not overridden, so calls to private methods are resolved at compile time and not subject to dynamic method lookup.

24. Regardless of the type of the reference, a method will always access its own fields rather than those of a derived class.

25. A static method can be inherited.

26. A static variable can be overridden by a non-static variable in the child class.

27. Overriding is for non-static methods and hiding is for static methods. So the following statements are the only true statements about hiding and overriding:
    - a static method (in a subclass) may hide another static method (in a superclass)
    - a static method (in a subclass) cannot hide a non-static method (in a superclass)
    - a non-static method (in a subclass) may override another non-static method (in a superclass)
    - a non-static method (in a subclass) cannot override a static method (in a superclass)

28. It is legal for overloaded methods to have same or different return types, as far as their argument list is different.

29. Overridden methods should have the same return type in the subclass. If the name and argument list are the same but the return type is different here, the compilation will fail.

30. An interface can be private or protected.

31. Interface methods are public and abstract while variables are public, static and final.

32. If implementation for all the methods declared in the interface are not provided, the class itself should be declared as abstract, otherwise the class will not compile.

33. The classes defined inside an interface are always public and static and can not call the methods declared inside the interface.

34. If a class implements two interfaces and both the interfaces have identical method declaration, it is totally valid.

35. If a class implements two interfaces and both the interfaces have identical method names and argument lists but return type different, the code will not compile.

36. All the wrapper classes are final.

37. Constructors are not inherited.

38. A constructor can't call the same constructor from within. Compiler will say 'recursive constructor invocation'.

39. Constructor body can have an empty return statement. Though void cannot be specified with the constructor signature, empty return statement is acceptable.

40. The default constructor is lost when a user-defined constructor (with arguments or without arguments) is added to the class.

41. Call to this(…) and super(…) always must be first statement in constructor.

42. If there is no explicit constructor call, compiler automatically inserts a default constructor call of the super class. There is a cascading effect in this case.

43. Only modifiers that a constructor can have are the accessibility modifiers.

44. A class which has all its constructors declared as private cannot be instantiated by any other class and can not be extended.

45. Some of the unchecked exceptions: ArithmeticException, ClassCastException, IllegalArgumentException, IndexOutOfBoundException, NullPointerException, SecurityException.

46. Use throw new xxxException() to throw an exception. If the thrown object is null, a NullPointerException will be thrown at the handler.

47. If there's no code in try block that may throw exceptions specified in the catch blocks, compiler will produce an error. (This is not the case for super-class Exception)

48. A finally block is always executed irrespective of exception is caught of not. The only exception to this rule is when the current thread dies before the finally block, e.g. A call to System.exit().

49. try-catch-finally blocks can be nested.

50. Floating point numbers are double by default.

51. % operators are generally used with integers but they can be used with floating point numbers also and 2.3f%0 won't give any exception like with division of floating points number by 0.

52. All the arithmetic operators produce results of int, long, float or double type only. All the other results should be explicitly type casted before assignment if the variable in the left side is of smaller size.
    e. g. short a = 10 * 5; should be written as short a = (short) 10 * 5;

53. Just like multiple assignment, multiple increment and assignments are also possible.
    e.g. a += b += c; is valid

54. 0 to 2^16-1 is the range of the char data type.

55. long (64 bits) can be assigned to float (32 bits).

56. Default type of a numeric literal with a decimal point is double.

57. Each wrapper class override equals(), toString() and hashCode() method

58. Boolean b1 = new Boolean("true");
    Boolean b2 = new Boolean("true");

    Here b1 and b2 will point to different memory locations as they are different objects altogether, so they will not be equal (==)

59. Boolean b1 = new Boolean("TRUE");
    Boolean b2 = new Boolean("true");

    Here both will be equal (==), as their content is the same. As a wrapper class, Boolean is not case sensitive with the value passed to it as the argument.

60. Boolean b1 = new Boolean("true");
    Boolean b2 = new Boolean("another string");

    Anything apart from true/TRUE, false/FALSE are taken as false in the Boolean wrapper class. So, printing the second variable will simply print false.

61. Break statement can be used with any kind of loop or a switch statement or just a labeled block.

62. Continue must be in a loop(for, do , while). It cannot appear in case constructs.

63. In all the initializers, forward referencing of variables is not allowed. Forward referencing of methods is allowed.

64. An inner class can not have same name as enclosing class.

65. An inner class can be public, private, protected or default.

66. An inner class can be final or abstract also.

67. An inner class can be static or non-static.

68. When static, the enclosing class instance is not required to access an inner class.

69. When non-static, enclosing class instance is required to access an inner class.

70. An inner class can't have static member variables except compile time constants (static final)

71. An inner class variable can shadow an outer class variable. If the inner class is sub-classed within the same outer class, the variable has to be qualified explicitly in the sub-class. To fully qualify the variable, use classname.this.variablename. If we don't correctly qualify the variable, a compiler error will occur. (Note that this does not happen in multiple levels of inheritance where an upper-most super-class's variable is silently shadowed by the most recent super-class variable or in multiple levels of nested inner classes where an innermost class's variable silently shadows an outer-most class's variable. Problem comes only when these two hierarchy chains (inheritance and containment) clash.)

72. The void class is not instantiable.

73. For String constructors valid arguments: literal, byte array, char array, stringbuffer.

74. For Stringbuffer constructors valid arguments : void, int, string

75. X != X comparison involving Nan returns true.

76. + and += operators are not overloaded for stringbuffer class and it will produce compiler error.

77. Equals method is not overridden by stringbuffer class and it woks just as == operator.

78. Comparing or assigning stringbuffer to string will cause compilation error.

79. trim() is not a stringbuffer method.

80. An empty string is NOT the same as a null string

81. Strings are initialized to null, not empty string

82. String objects are always immutable in all the cases. Calling one of the String's data modification methods or reassigning an instance variable of type String.

83. A class without abstract methods can still be declared abstract.

84. Even if a single method of a class is declared as abstract, the class itself has to be declared as abstract

85. If a class is abstract, no instance of that class can be created

86. Arrays, whether local or class-level, are always initialized.

87. File class has NO methods to deal with the contents of the file.

88. Classes, variables and methods can be declared final.

89. Class declared as final can't be sub-classed.

90. If variables declared as final, the value they contain can't be changed.

91. Method declared as final can't be overridden.

92. If an object is declared as final, the reference it stores can't be changed, but the content of the object it is pointing to, can be changed.

93. The transient modifier applies only to class variables.

94. Garbage collector is a low priority thread which runs in background when the system goes out of memory or the CPU load is low, and reclaims the memory taken by unused objects.

95. Calling System.gc() and Runtime.gc() only suggest garbage collection to JVM, but do not guarantee that garbage collection will be run immediately.

96. It is not promised that as soon as there are no references to an object, the garbage collector will be run.

97. Till an object is accessible to any live thread, that object will not be garbage collected.

98. When command line arguments are supplied to main(), “java” and the “name of the program” are not part of the argument list, e.g., when myClass is passed an argument as java myClass red green blue, red will have an index of 0, green will have an index of 1 and blue will have an index of 2. This is different with C and C++.

99. The main method is declared as public static void main (String[] args) {…}. The place for public and static can be interchanged, but void has to be there before the name of the method.

100. You must override the hashCode() in each and every class which overrides equals(). Otherwise it will be a violation of the contract.

Thank you for visiting Vertical Horizons…!!

Please feel free to write to

admin@verticalhorizons.in

for any queries, feedback or suggestions.

For latest updates on Vertical Horizons, you can subscribe @

http://feedburner.google.com/fb/a/mailverify?uri=verticalhorizons/vh