ANSWER GUIDE

JAVA TEST - 15 May 2012

1. What is the most restrictive access modifier that will allow members of one class to have access to members of another class in the same package?

A. public

B. abstract

C. protected

D. default access

Explanation:

`default` access is the "package oriented" access modifier.

Option A and C are wrong because `public` and `protected` are less restrictive. Option B and D are wrong because `abstract` and `synchronized` are not access modifiers.

Read More: Declarations & Access Control

2. You want a class to have access to members of another class in the same package. Which is the most restrictive access that accomplishes this objective?

A. public

B. private

C. protected

D. default access

Explanation:

The only two real contenders are C and D. `Protected` access Option C makes a member accessible only to classes in the same package or subclass of the class. While default access Option D makes a member accessible only to classes in the same package.

Read More: Declarations & Access Control

3. Which is a valid declaration within an interface?

A. public static short stop = 23;

B. protected short stop = 23;

C. transient short stop = 23;

D. final void madness(short stop);

Explanation:

(A) is valid interface declarations.

(B) and (C) are incorrect because interface variables cannot be either `protected` or `transient`. (D) is incorrect because interface methods cannot be `final` or `static`.

Read More: Declarations & Access Control

4. Which two are acceptable types for x?

1. byte
2. long
3. char
4. float
5. Short
6. Long

A. 1 and 3

B. 2 and 4

C. 3 and 5

D. 4 and 6

Explanation:

`Switch` statements are based on integer expressions and since both bytes and chars can implicitly be widened to an integer, these can also be used. Also shorts can be used.`Short` and `Long` are wrapper classes and reference types can not be used as variables.

Read More: Flow Control

5. 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 x: System.out.print("0 ");
                case x-1: System.out.print("1 ");
                case x-2: System.out.print("2 ");
            }
        }
    }
}
```

A. 0 1 2

B. 0 1 2 1 2 2

C. 2 1 0 1 0 0

D. 2 1 2 0 1 2

Explanation:

The case expressions are all legal because `x` is marked `final`, which means the expressions can be evaluated at compile time. In the first iteration of the for loop case`x-2` matches, so 2 is printed. In the second iteration, `x-1` is matched so 1 and 2 are printed (remember, once a match is found all remaining statements are executed until a break statement is encountered). In the third iteration, `x` is matched. So 0 1 and 2 are printed.

Read More: 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.

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.

Read More: Flow Control

7. 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 < 4; z++)
        {
            switch (z)
            {
                case x: System.out.print("0 ");
                default: System.out.print("def ");
                case x-1: System.out.print("1 ");
                          break;
                case x-2: System.out.print("2 ");
            }
        }
    }
}
```

A. 0 def 1

B. 2 1 0 def 1

C. 2 1 0 def def

D. 2 1 0 def 1 def 1

Explanation:

When `z == 0` , case `x-2` is matched. When `z == 1`, case `x-1` is matched and then the break occurs. When `z == 2`, case `x`, then default, then `x-1` are all matched. When `z == 3`, default, then `x-1` are matched. The rules for default are that it will fall through from above like any other case (for instance when `z == 2`), and that it will match when no other cases match (for instance when `z==3`).

Read More: Flow Control

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

A. java.util.HashSet

B. java.util.LinkedHashSet

C. java.util.List

D. java.util.ArrayList

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`.

Read More: Objects & Collections

9. What will be the output of the program?

```
public class Test
{
    public static void main (String args[])
    {
        String str = NULL;
        System.out.println(str);
    }
}
```

A. NULL

B. Compile Error

C. Code runs but no output

D. Runtime Exception

Explanation:

Option B is correct because to set the value of a `String` variable to null you must use "`null`" and not "`NULL`".

Read More: Objects & Collections

10. Which of the following will not directly cause a thread to stop?

A. notify()

B. wait()

C. InputStream access

D. sleep()

Explanation:

Option A is correct. `notify()` - wakes up a single thread that is waiting on this object's monitor.

Option B is wrong. `wait()` causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.

Option C is wrong. Methods of the InputStream class block until input data is available, the end of the stream is detected, or an exception is thrown. Blocking means that a thread may stop until certain conditions are met.

Option D is wrong. `sleep()` - Causes the currently executing thread to sleep (temporarily cease execution) for a specified number of milliseconds. The thread does not lose ownership of any monitors.

Read More: Threads

11. What will be the output of the program?

```
public class SyncTest
{
    public static void main (String [] args)
    {
        Thread t = new Thread()
        {
            Foo f = new Foo();
```

```
            public void run()
            {
                f.increase(20);
            }
        };
    t.start();
    }
}
class Foo
{
    private int data = 23;
    public void increase(int amt)
    {
        int x = data;
        data = x + amt;
    }
}
```

and assuming that data must be protected from corruption, whatâ€”if anythingâ€”can you add to the preceding code to ensure the integrity of data?

A. Synchronize the run method.

B. Wrap a synchronize(this) around the call to `f.increase()`.

C. The existing code will cause a runtime exception.

D. Synchronize the `increase()` method

Explanation:

Option D is correct because synchronizing the code that actually does the increase will protect the code from being accessed by more than one thread at a time.

Option A is incorrect because synchronizing the `run()` method would stop other threads from running the `run()` method (a bad idea) but still would not prevent other threads with other runnables from accessing the `increase()` method.

Option B is incorrect for virtually the same reason as Aâ€”synchronizing the code that calls the `increase()` method does not prevent other code from calling the `increase()` method.

Read More: Threads

12. What will be the output of the program?

```
class Test116
{
static final StringBuffer sb1 = new StringBuffer();
static final StringBuffer sb2 = new StringBuffer();
public static void main(String args[])
{
    new Thread()
    {
        public void run()
        {
            synchronized(sb1)
            {
                sb1.append("A");
                sb2.append("B");
            }
        }
    }.start();

    new Thread()
    {
        public void run()
        {
            synchronized(sb1)
            {
                sb1.append("C");
                sb2.append("D");
            }
        }
    }.start(); /* Line 28 */

    System.out.println (sb1 + " " + sb2);
    }
}
```

A. `main()` will finish before starting threads.

B. `main()` will finish in the middle of one thread

C. `main()` will finish after one thread.

D. Cannot be determined.

Explanation:

Can you guarantee the order in which threads are going to run? No you can't. So how do you know what the output will be? The output cannot be determined.

add this code after line 28:

```
try { Thread.sleep(5000); } catch(InterruptedException e) { }
```

and you have some chance of predicting the outcome.

Read More: Threads

13. Which statement is true?

A. If only one thread is blocked in the wait method of an object, and another thread executes the modify on that same object, then the first thread immediately resumes execution.

B. If a thread is blocked in the wait method of an object, and another thread executes the notify method on the same object, it is still possible that the first thread might never resume execution.

C. If a thread is blocked in the wait method of an object, and another thread executes the notify method on the same object, then the first thread definitely resumes execution as a direct and sole consequence of the notify call.

D. If two threads are blocked in the wait method of one object, and another thread executes the notify method on the same object, then the first thread that executed the wait call first definitely resumes execution as a direct and sole consequence of the notify call.

Explanation:

Option B is correct - The notify method only wakes the thread. It does not guarantee that the thread will run.

Option A is incorrect - just because another thread activates the modify method in A this does not mean that the thread will automatically resume execution

Option C is incorrect - This is incorrect because as said in Answer B notify only wakes the thread but further to this once it is awake it goes back into the stack and awaits execution therefore it is not a "direct and sole consequence of the notify call"

Option D is incorrect - The notify method wakes one waiting thread up. If there are more than one sleeping threads then the choice as to which thread to wake is made by the machine rather than you therefore you cannot guarantee that the notify'ed thread will be the first waiting thread.

Read More: Threads

14. Which statement is true?

A. Memory is reclaimed by calling `Runtime.gc()`.

B. Objects are not collected if they are accessible from live threads.

C. An `OutOfMemory` error is only thrown if a single block of memory cannot be found that is large enough for a particular requirement.

D. Objects that have `finalize()` methods always have their `finalize()` methods called before the program ends.

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.

Read More: Garbage Collections

15. What will be the output of the program (when you run with the -ea option) ?

```
public class Test
{
    public static void main(String[] args)
    {
        int x = 0;
        assert (x > 0) : "assertion failed"; /* Line 6 */
        System.out.println("finished");
    }
}
```

A. finished

B. Compilation fails.

C. An AssertionError is thrown.

D. An AssertionError is thrown and finished is output.

Explanation:

An assertion Error is thrown as normal giving the output "assertion failed". The word "finished" is not printed (ensure you run with the `-ea` option)

Assertion failures are generally labeled in the stack trace with the file and line number from which they were thrown, and also in this case with the error's detail message "assertion failed". The detail message is supplied by the assert statement in line 6.

Read More: Assertions

16. Which line is an example of an inappropriate use of assertions?

A. Line 11

B. Line 12

C. Line 14

D. Line 22

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.

Read More: Assertions

17. Which statement is true?

A. Assertions can be enabled or disabled on a class-by-class basis.

B. Conditional compilation is used to allow tested classes to run at full speed.

C. Assertions are appropriate for checking the validity of arguments in a method.

D. The programmer can choose to execute a return statement or to throw an exception if an assertion fails.

Explanation:

Option A is correct. The assertion status can be set for a named top-level class and any nested classes contained therein. This setting takes precedence over the class loader's default assertion status, and over any applicable per-package default. If the named class is not a top-level class, the change of status will have no effect on the actual assertion status of any class.

Option B is wrong. Is there such a thing as conditional compilation in Java?

Option C is wrong. For private methods - yes. But do not use assertions to check the parameters of a public method. An assert is inappropriate in public methods because the method guarantees that it will always enforce the argument checks. A public method must check its arguments whether or not assertions are enabled. Further, the assert construct does not throw an exception of the specified type. It can throw only an`AssertionError`.

Option D is wrong. Because you're never supposed to handle an assertion failure. That means don't catch it with a `catch` clause and attempt to recover.

Read More: Assertions

18. What will be the output of the program?

```
String x = "xyz";
x.toUpperCase(); /* Line 2 */
String y = x.replace('Y', 'y');
y = y + "abc";
System.out.println(y);
```

A. abcXyZ

B. abcxyz

C. xyzabc

D. XyZabc

Explanation:

Line 2 creates a new `String` object with the value "`XYZ`", but this new object is immediately lost because there is no reference to it. Line 3 creates a new `String` object referenced by `y`. This new `String` object has the value "`xyz`" because there was no "`Y`" in the String object referred to by `x`. Line 4 creates a new `String` object, appends "`abc`" to the value "`xyz`", and refers `y` to the result.

Read More: Java.lang Class

19. What will be the output of the program?

```
class Tree { }
class Pine extends Tree { }
class Oak extends Tree { }
public class Forest1
{
    public static void main (String [] args)
    {
        Tree tree = new Pine();
        if( tree instanceof Pine )
            System.out.println ("Pine");
        else if( tree instanceof Tree )
            System.out.println ("Tree");
        else if( tree instanceof Oak )
            System.out.println ( "Oak" );
        else
            System.out.println ("Oops ");
    }
}
```

A. Pine

B. Tree

C. Forest

D. Oops

Explanation:

The program prints "Pine".

Read More: Java.lang Class

20. What two statements are true about the result obtained from calling `Math.random()`?

1. The result is less than 0.0.
2. The result is greater than or equal to 0.0..
3. The result is less than 1.0.
4. The result is greater than 1.0.
5. The result is greater than or equal to 1.0.

A. 1 and 2

B. 2 and 3

C. 3 and 4

D. 4 and 5

Explanation:

(1) and (2) are correct. The result range for `random()` is `0.0` to < `1.0`; `1.0` is not in range.

Read More: Java.lang Class

REMEMBER: PREPARATION IS KEY