lgsi('geekathon');

ANSWER GUIDE

JAVA TEST – 11 May 2012

1. You want subclasses in any package to have access to members of a superclass. Which is the most restrictive access that accomplishes this objective?

A. public

B. private

C. protected

D. transient

Explanation:

Access modifiers dictate which classes, not which instances, may access features.

Methods and variables are collectively known as members. Method and variable members are given access control in exactly the same way.

`private` makes a member accessible only from within its own class

`protected` makes a member accessible only to classes in the same package or subclass of the class

`default` access is very similar to protected (make sure you spot the difference) default access makes a member accessible only to classes in the same package.

`public` means that all other classes regardless of the package that they belong to, can access the member (assuming the class itself is visible)

`final` makes it impossible to extend a class, when applied to a method it prevents a method from being overridden in a subclass, when applied to a variable it makes it impossible to reinitialise a variable once it has been initialised

`abstract` declares a method that has not been implemented.

`transient` indicates that a variable is not part of the persistent state of an object.

`volatile` indicates that a thread must reconcile its working copy of the field with the master copy every time it accesses the variable.

After examining the above it should be obvious that the access modifier that provides the most restrictions for methods to be accessed from the subclasses of the class from another package is C - `protected`. A is also a contender but C is more restrictive, B would be the answer if the constraint was the "same package" instead of "any package" in other words the subclasses clause in the question eliminates default.

Read More: Declarations & Access Control

2. Which two code fragments will compile?

```
interface Base
{
    boolean m1 ();
    byte m2(short s);
}
```

1. `interface Base2 implements Base {}`
2. `abstract class Class2 extends Base { public boolean m1(){ return true; }}`
3. `abstract class Class2 implements Base {}`
4. `abstract class Class2 implements Base { public boolean m1(){ return (7 > 4); }}`
5. `abstract class Class2 implements Base { protected boolean m1(){ return (5 > 7) }}`

A. 1 and 2

B. 2 and 3

C. 3 and 4

D. 1 and 5

Explanation:

(3) is correct because an abstract class doesn't have to implement any or all of its interface's methods. (4) is correct because the method is correctly implemented (`(7 > 4)` is a boolean).

(1) is incorrect because interfaces don't implement anything. (2) is incorrect because classes don't extend interfaces. (5) is incorrect because interface methods are implicitly`public`, so the methods being implemented must be `public`.

Read More: Declarations & Access Control

3. Which is valid in a class that extends class A?

```
class A
{
    protected int method1(int a, int b)
    {
        return 0;
    }
}
```

A. public int method1(int a, int b) {return 0; }

B. private int method1(int a, int b) { return 0; }

C. public short method1(int a, int b) { return 0; }

D. static protected int method1(int a, int b) { return 0; }

Explanation:

Option A is correct - because the class that extends `A` is just simply overriding `method1`.

Option B is wrong - because it can't override as there are less access privileges in the subclass `method1`.

Option C is wrong - because to override it, the return type needs to be an integer. The different return type means that the method is not overriding but the same argument list means that the method is not overloading. Conflict - compile time error.

Option D is wrong - because you can't override a method and make it a class method i.e. using `static`.

Read More: Declarations & Access Control

4. What will be the output of the program?

```
class Test
{
    public static void main(String [] args)
    {
        Test p = new Test();
        p.start();
    }

    void start()
    {
        boolean b1 = false;
        boolean b2 = fix(b1);
        System.out.println(b1 + " " + b2);
    }

    boolean fix(boolean b1)
    {
        b1 = true;
        return b1;
    }
}
```

A. true true

B. false true

C. true false

D. false false

Explanation:

The boolean `b1` in the `fix()` method is a different boolean than the `b1` in the `start()` method. The `b1` in the `start()` method is not updated by the `fix()` method.

Read More: Operators & Assignments

5. What will be the output of the program?

```
class PassS
{
    public static void main(String [] args)
    {
        PassS p = new PassS();
        p.start();
    }

    void start()
    {
        String s1 = "slip";
        String s2 = fix(s1);
        System.out.println(s1 + " " + s2);
    }

    String fix(String s1)
    {
        s1 = s1 + "stream";
        System.out.print(s1 + " ");
        return "stream";
    }
}
```

A. slip stream

B. slipstream stream

C. stream slip stream

D. slipstream slip stream

Explanation:

When the `fix()` method is first entered, `start()`'s s1 and `fix()`'s `s1` reference variables both refer to the same String object (with a value of "slip"). `Fix()`'s `s1` is reassigned to a new object that is created when the concatenation occurs (this second String object has a value of "slipstream"). When the program returns to `start()`, another `String` object is created, referred to by `s2` and with a value of "stream".

Read More: Operators & Assignments

6. What will be the output of the program?

```
class Test
{
    static int s;
    public static void main(String [] args)
    {
        Test p = new Test();
        p.start();
        System.out.println(s);
    }

    void start()
    {
        int x = 7;
        twice(x);
        System.out.print(x + " ");
    }

    void twice(int x)
    {
        x = x*2;
        s = x;
    }
}
```

A. 7 7

B. 7 14

C. 14 0

D. 14 14

Explanation:

The `int x` in the `twice()` method is not the same `int x` as in the `start()` method.`Start()`'s `x` is not affected by the `twice()` method. The instance variable `s` is updated by `twice()`'s `x`, which is 14.

Read More: Operators & Assignments

7. Which two of the following statements, inserted independently, could legally be inserted into missing section of this code?

```
import java.awt.*;
class Ticker extends Component
{
    public static void main (String [] args)
    {
        Ticker t = new Ticker();
        /* Missing Statements ? */
    }
}
```

1. boolean test = (Component instanceof t);
2. boolean test = (t instanceof Ticker);
3. boolean test = t.instanceof(Ticker);
4. boolean test = (t instanceof Component);

A. 1 and 4

B. 2 and 3

C. 1 and 3

D. 2 and 4

Explanation:

(2) is correct because class type `Ticker` is part of the class hierarchy of `t`; therefore it is a legal use of the `instanceof` operator. (4) is also correct because `Component` is part of the hierarchy of `t`, because `Ticker extends Component`.

(1) is incorrect because the syntax is wrong. A variable (or null) always appears before the `instanceof` operator, and a type appears after it. (3) is incorrect because the statement is used as a method (`t.instanceof(Ticker);`), which is illegal.

Read More: Operators & Assignments

8. Which statement is true?

```
public void test(int x)
{
    int odd = 1;
    if(odd) /* Line 4 */
    {
        System.out.println("odd");
    }
    else
    {
        System.out.println("even");
    }
}
```

A. Compilation fails.

B. "odd" will always be output.

C. "even" will always be output.

D. "odd" will be output for odd values of x, and "even" for even values.

Explanation:

The compiler will complain because of incompatible types (line 4), the if expects a boolean but it gets an integer.

Read More: Flow Control

9. What will be the output of the program?

```
int i = 1, j = 10;
do
{
    if(i > j)
    {
        break;
    }
    j--;
} while (++i < 5);
```

```
System.out.println("i = " + i + " and j = " + j);
```

A. i = 6 and j = 5

B. i = 5 and j = 5

C. i = 6 and j = 4

D. i = 5 and j = 6

Explanation:

This loop is a do-while loop, which always executes the code block within the block at least once, due to the testing condition being at the end of the loop, rather than at the beginning. This particular loop is exited prematurely if `i` becomes greater than `j`.

The order is, test `i` against `j`, if bigger, it breaks from the loop, decrements `j` by one, and then tests the loop condition, where a pre-incremented by one `i` is tested for being lower than 5. The test is at the end of the loop, so `i` can reach the value of 5 before it fails. So it goes, start:

1, 10

2, 9

3, 8

4, 7

5, 6 loop condition fails.

Read More: Flow Control

10. What will be the output of the program?

```
public class Test
{
    public static void main(String args[])
    {
        int i = 1, j = 0;
        switch(i)
        {
            case 2: j += 6;
            case 4: j += 1;
```

```
            default: j += 2;
            case 0: j += 4;
        }
        System.out.println("j = " + j);
    }
}
```

A. 0

B. 2

C. 4

D. 6

Explanation:

Because there are no break statements, the program gets to the default case and adds2 to `j`, then goes to `case 0` and adds `4` to the new `j`. The result is `j = 6`.

Read More: Flow Control

11. Which of these is true?

```
import java.io.*;
public class MyProgram
{
    public static void main(String args[])
    {
        FileOutputStream out = null;
        try
        {
            out = new FileOutputStream("test.txt");
            out.write(122);
        }
        catch(IOException io)
        {
            System.out.println("IO Error.");
        }
        finally
        {
            out.close();
```

```
            }
        }
}
```

and given that all methods of class `FileOutputStream`, including `close()`, throw an`IOException.`

A. This program will compile successfully.

B. This program fails to compile due to an error at line 4.

C. This program fails to compile due to an error at line 6.

D. This program fails to compile due to an error at line 13.

Explanation:

Any method (in this case, the `main()` method) that throws a checked exception (in this case, `out.close()`) must be called within a try clause, or the method must declare that it throws the exception. Either `main()` must declare that it throws an exception, or the call to `out.close()` in the finally block must fall inside a (in this case nested) try-catch block.

Read More: Exceptions

12. Which statement is true?

A. `catch(X x)` can catch subclasses of `X` where `X` is a subclass of `Exception`.

B. The `Error` class is a `RuntimeException`.

C. Any statement that can throw an `Error` must be enclosed in a try block.

D. Any statement that can throw an `Exception` must be enclosed in a try block.

Explanation:

Option A is correct. If the class specified in the catch clause does have subclasses, any exception object that subclasses the specified class will be caught as well.

Option B is wrong. The error class is a subclass of `Throwable` and not `Runtime`Exception.

Option C is wrong. You do not catch this class of error.

Option D is wrong. An exception can be thrown to the next method higher up the call stack.

Read More: Exceptions

13. Which is valid declaration of a float?

A. float f = 1F;

B. float f = 1.0;

C. float f = "1";

D. float f = 1.0d;

Explanation:

Option A is valid declaration of float.

Option B is incorrect because any literal number with a decimal point u declare the computer will implicitly cast to double unless you include "F or f"

Option C is incorrect because it is a `String`.

Option D is incorrect because "d" tells the computer it is a double so therefore you are trying to put a double value into a float variable i.e there might be a loss of precision.

Read More: Objects & Collections

14. What will be the output of the program?

```
import java.util.*;
class H
{
    public static void main (String[] args)
    {
        Object x = new Vector().elements();
        System.out.print((x instanceof Enumeration)+",");
        System.out.print((x instanceof Iterator)+",");
        System.out.print(x instanceof ListIterator);
```

```
    }
}
```

A. Prints: false,false,false

B. Prints: false,false,true

C. Prints: false,true,false

D. Prints: true,false,false

Explanation:

The `Vector.elements` method returns an `Enumeration` over the elements of the vector. `Vector` implements the `List` interface and extends `AbstractList` so it is also possible to get an `Iterator` over a `Vector` by invoking the `iterator` or `listIterator`method.

Read More: Objects & Collections

15. What will be the output of the program?

```
TreeSet map = new TreeSet();
map.add("one");
map.add("two");
map.add("three");
map.add("four");
map.add("one");
Iterator it = map.iterator();
while (it.hasNext() )
{
    System.out.print( it.next() + " " );
}
```

A. one two three four

B. four three two one

C. four one three two

D. one two three four one

Explanation:

`TreeSet` assures no duplicate entries; also, when it is accessed it will return elements in natural order, which typically means alphabetical.

Read More: Objects & Collections

16. Which statement, inserted at line 10, creates an instance of Bar?

```
class Foo
{
    class Bar{ }
}
class Test
{
    public static void main (String [] args)
    {
        Foo f = new Foo();
        /* Line 10: Missing statement ? */
    }
}
```

A. Foo.Bar b = new Foo.Bar();

B. Foo.Bar b = f.new Bar();

C. Bar b = new f.Bar();

D. Bar b = f.new Bar();

Explanation:

Option B is correct because the syntax is correct-using both names (the enclosing class and the inner class) in the reference declaration, then using a reference to the enclosing class to invoke new on the inner class.

Option A, C and D all use incorrect syntax. A is incorrect because it doesn't use a reference to the enclosing class, and also because it includes both names in the new.

C is incorrect because it doesn't use the enclosing class name in the reference variable declaration, and because the new syntax is wrong.

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

Read More: Inner Classes

17. Which three guarantee that a thread will leave the running state?

1. yield()
2. wait()
3. notify()
4. notifyAll()
5. sleep(1000)
6. aLiveThread.join()
7. Thread.killThread()

A. 1, 2 and 4

B. 2, 5 and 6

C. 3, 4 and 7

D. 4, 5 and 7

Explanation:

(2) is correct because `wait()` always causes the current thread to go into the object's wait pool.

(5) is correct because `sleep()` will always pause the currently running thread for at least the duration specified in the sleep argument (unless an interrupted exception is thrown).

(6) is correct because, assuming that the thread you're calling `join()` on is alive, the thread calling `join()` will immediately block until the thread you're calling `join()` on is no longer alive.

(1) is wrong, but tempting. The `yield()` method is not guaranteed to cause a thread to leave the running state, although if there are runnable threads of the same priority as the currently running thread, then the current thread will probably leave the running state.

(3) and (4) are incorrect because they don't cause the thread invoking them to leave the running state.

(7) is wrong because there's no such method.

Read More: Threads

18. Which will contain the body of the thread?

A. run();

B. start();

C. stop();

D. main();

Explanation:

Option A is Correct. The `run()` method to a thread is like the `main()` method to an application. Starting the thread causes the object's run method to be called in that separately executing thread.

Option B is wrong. The `start()` method causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

Option C is wrong. The `stop()` method is deprecated. It forces the thread to stop executing.

Option D is wrong. Is the main entry point for an application.

Read More: Threads

19. 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");
    }
}
```

A. finished

B. Compilation fails.

C. An `AssertionError` is thrown and finished is output.

D. An `AssertionError` is thrown with the message "assertion failed."

Explanation:

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

Read More: Assertions

20. What will be the output of the program?

```
public class BoolTest
{
    public static void main(String [] args)
    {
        int result = 0;

        Boolean b1 = new Boolean("TRUE");
        Boolean b2 = new Boolean("true");
        Boolean b3 = new Boolean("tRuE");
        Boolean b4 = new Boolean("false");

        if (b1 == b2)  /* Line 10 */
            result = 1;
        if (b1.equals(b2) ) /* Line 12 */
            result = result + 10;
```

```
        if (b2 == b4)  /* Line 14 */
            result = result + 100;
        if (b2.equals(b4) ) /* Line 16 */
            result = result + 1000;
        if (b2.equals(b3) ) /* Line 18 */
            result = result + 10000;

        System.out.println("result = " + result);
    }
}
```

A. 0

B. 1

C. 10

D. 10010

Explanation:

Line 10 fails because b1 and b2 are two different objects. Lines 12 and 18 succeed because the `Boolean` String constructors are case insensitive. Lines 14 and 16 fail because true is not equal to false.

Read More: Java.lang Class

REMEMBER: PREPARATION IS KEY