lgsi('geekathon');

ANSWER GUIDE

JAVA TEST – 14 May 2012

1. Given a method in a protected class, what access modifier do you use to restrict access to that method to only the other members of the same class?

A. final

B. static

C. private

D. protected

Explanation:

The `private` access modifier limits access to members of the same class.

Option A, B, D, and E are wrong because `protected` are the wrong access modifiers, and `final`, `static`, and `volatile` are modifiers but not access modifiers.

Read More: Declarations & Access Control

2. Which two code fragments inserted at end of the program, will allow to compile?

```java
interface DoMath
{
    double getArea(int rad);
}
```

```
interface MathPlus
{
    double getVol(int b, int h);

}
/* Missing Statements ? */
```

1. class AllMath extends DoMath { double getArea(int r); }
2. interface AllMath implements MathPlus { double getVol(int x, int y); }
3. interface AllMath extends DoMath { float getAvg(int h, int l); }
4. class AllMath implements MathPlus { double getArea(int rad); }
5. abstract class AllMath implements DoMath, MathPlus { public double getArea(int rad) { return rad * rad * 3.14; } }

A. 1 only

B. 2 only

C. 3 and 5

D. 1 and 4

Explanation:

((3) are (5) are correct because interfaces and abstract classes do not need to fully implement the interfaces they extend or implement (respectively).

(1) is incorrect because a class cannot extend an interface. (2) is incorrect because an interface cannot implement anything. (4) is incorrect because the method being implemented is from the wrong interface.

Read More: Declarations & Access Control

3. Which three statements are true?

1. The default constructor initialises method variables.
2. The default constructor has the same access as its class.
3. The default constructor invokes the no-arg constructor of the superclass.
4. If a class lacks a no-arg constructor, the compiler always creates a default constructor.
5. The compiler creates a default constructor only when there are no other constructors for the class.

A. 1, 2 and 4

C. 3, 4 and 5

D. 1, 2 and 3

Explanation:

(2) sounds correct as in the example below

```java
class CoffeeCup {
   private int innerCoffee;
   public CoffeeCup() {
   }

   public void add(int amount) {
   innerCoffee += amount;
   }
   //...
 }
```

The compiler gives default constructors the same access level as their class. In the example above, class `CoffeeCup` is `public`, so the default constructor is `public`. If `CoffeeCup` had been given package access, the default constructor would be given package access as well.

(3) is correct. The Java compiler generates at least one instance initialisation method for every class it compiles. In the Java class file, the instance initialisation method is named "`<init>`." For each constructor in the source code of a class, the Java compiler generates one `<init>()` method. If the class declares no constructors explicitly, the compiler generates a default no-arg constructor that just invokes the superclass's no-arg constructor. As with any other constructor, the compiler creates an <init>() method in the class file that corresponds to this default constructor.

(5) is correct. The compiler creates a default constructor if you do not declare any constructors in your class.

Read More: Declarations & Access Control

4. What will be the output of the program?

```java
class Test
{
    public static void main(String [] args)
    {
        int x=20;
```

```
        String sup = (x < 15) ? "small" : (x < 22)? "tiny" : "huge";
        System.out.println(sup);
    }
}
```

A. small

B. tiny

C. huge

D. Compilation fails

This is an example of a nested ternary operator. The second evaluation `(x < 22)` is`true`, so the "`tiny`" value is assigned to `sup`.

Read More: Operators & Assignments

5. Which of the following are legal lines of code?

1. int w = (int)888.8;
2. byte x = (byte)1000L;
3. long y = (byte)100;
4. byte z = (byte)100L;

A. 1 and 2

B. 2 and 3

C. 3 and 4

D. All statements are correct.

Statements (1), (2), (3), and (4) are correct. (1) is correct because when a floating-point number (a double in this case) is cast to an `int`, it simply loses the digits after the decimal.

(2) and (4) are correct because a long can be cast into a `byte`. If the long is over 127, it loses its most significant (leftmost) bits.

(3) actually works, even though a cast is not necessary, because a `long` can store a `byte`.

Read More: Operators & Assignments

6. Which two statements are equivalent?

1. 16*4
2. 16>>2
3. 16/2^2
4. 16>>>2

A. 1 and 2

B. 2 and 4

C. 3 and 5

D. 1 and 3

Explanation:

(2) is correct. 16 >> 2 = 4

(4) is correct. 16 >>> 2 = 4

(1) is wrong. 16 * 4 = 64

(3) is wrong. 16/2 ^ 2 = 10

Read More: Operators & Assignments

7. What will be the output of the program?

```java
public class Test
{
    public static void main(String [] args)
    {
```

```
        int I = 1;
        do while ( I < 1 )
        System.out.print("I is " + I);
        while ( I > 1 ) ;
    }
}
```

A. I is 1

B. I is 1 I is 1

C. No output is produced.

D. Compilation error

There are two different looping constructs in this problem. The first is a do-while loop and the second is a while loop, nested inside the do-while. The body of the do-while is only a single statement-brackets are not needed. You are assured that the while expression will be evaluated at least once, followed by an evaluation of the do-while expression. Both expressions are false and no output is produced.

Read More: Flow Control

8. What will be the output of the program?

```java
public class RTExcept
{
    public static void throwit ()
    {
        System.out.print("throwit ");
        throw new RuntimeException();
    }
    public static void main(String [] args)
    {
        try
        {
            System.out.print("hello ");
            throwit();
```

```
        }
        catch (Exception re )
        {
            System.out.print("caught ");
        }
        finally
        {
            System.out.print("finally ");
        }
        System.out.println("after ");
    }
}
```

A. hello throwit caught

B. Compilation fails

C. hello throwit `RuntimeException` caught after

D. hello throwit caught finally after

Explanation:

The `main()` method properly catches and handles the `RuntimeException` in the catch block, finally runs (as it always does), and then the code returns to normal.

A, B and C are incorrect based on the program logic described above. Remember that properly handled exceptions do not cause the program to stop executing.

Read More: Exceptions

9. What will be the output of the program?

```
public class X
{
    public static void main(String [] args)
    {
        try
        {
            badMethod();
            System.out.print("A");
```

```
        }
        catch (Exception ex)
        {
            System.out.print("B");
        }
        finally
        {
            System.out.print("C");
        }
        System.out.print("D");
    }
    public static void badMethod() {}
}
```

A. AC

B. BC

C. ACD

D. ABCD

Explanation:

There is no exception thrown, so all the code with the exception of the catch statement block is run.

Read More: Exceptions

10. Which of the following are Java reserved words?

1. run
2. import
3. default
4. implement

A. 1 and 2

B. 2 and 3

C. 3 and 4

D. 2 and 4

(2) - This is a Java keyword

(3) - This is a Java keyword

(1) - Is incorrect because although it is a method of `Thread`/`Runnable` it is not a keyword

(4) - This is not a Java keyword the keyword is implements

Read More: Objects & Collections

11. What will be the output of the program?

```java
public class Test
{
    public static void main (String[] args)
    {
        String foo = args[1];
        String bar = args[2];
        String baz = args[3];
        System.out.println("baz = " + baz); /* Line 8 */
    }
}
```

And the command line invocation:

> java Test red green blue

A. baz =

B. baz = null

C. baz = blue

D. Runtime Exception

Explanation:

When running the program you entered 3 arguments "red", "green" and "blue". When dealing with arrays in java you must remember ALL ARRAYS IN JAVA ARE ZERO BASED
therefore `args[0]` becomes "red", `args[1]` becomes "green" and `args[2]` becomes "blue".

When the program entcounters line 8 above at runtime it looks for `args[3]` which has never been created therefore you get an

`ArrayIndexOutOfBoundsException` at runtime.

Read More: Objects & Collections

12. What will be the output of the program?

```java
class Happy extends Thread
{
    final StringBuffer sb1 = new StringBuffer();
    final StringBuffer sb2 = new StringBuffer();

    public static void main(String args[])
    {
        final Happy h = new Happy();

        new Thread()
        {
            public void run()
            {
                synchronized(this)
                {
                    h.sb1.append("A");
                    h.sb2.append("B");
                    System.out.println(h.sb1);
                    System.out.println(h.sb2);
                }
            }
        }.start();

        new Thread()
        {
            public void run()
            {
                synchronized(this)
                {
                    h.sb1.append("D");
                    h.sb2.append("C");
                    System.out.println(h.sb2);
                    System.out.println(h.sb1);
                }
            }
        }.start();
    }
```

```
    }
```

A. ABBCAD

B. ABCBCAD

C. CDADACB

D. Output determined by the underlying platform.

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.

Read More: Threads

13. What will be the output of the program?

```java
class MyThread extends Thread
{
    public static void main(String [] args)
    {
        MyThread t = new MyThread();
        Thread x = new Thread(t);
        x.start(); /* Line 7 */
    }
    public void run()
    {
        for(int i = 0; i < 3; ++i)
        {
            System.out.print(i + "..");
        }
    }
}
```

A. Compilation fails.

B. 1..2..3..

C. 0..1..2..3..

D. 0..1..2..

The thread `MyThread` will start and loop three times (from 0 to 2).

Option A is incorrect because the `Thread` class implements the `Runnable` interface; therefore, in line 7, `Thread` can take an object of type `Thread` as an argument in the constructor.

Option B and C are incorrect because the variable i in the for loop starts with a value of 0 and ends with a value of 2.

Read More: Threads

14. Which statement is true?

A. A static method cannot be synchronized.

B. If a class has synchronized code, multiple threads can still access the nonsynchronized code.

C. Variables can be protected from concurrent access problems by marking them with the `synchronized` keyword.

D. When a thread sleeps, it releases its locks.

Explanation:

B is correct because multiple threads are allowed to enter nonsynchronized code, even within a class that has some synchronized methods.

A is incorrect because static methods can be synchronized; they synchronize on the lock on the instance of class `java.lang.Class` that represents the class type.

C is incorrect because only methodsâ€"not variablesâ€"can be marked synchronized.

D is incorrect because a sleeping thread still maintains its locks.

15. Which statement is true?

A. Calling `Runtime.gc()` will cause eligible objects to be garbage collected.

B. The garbage collector uses a mark and sweep algorithm.

C. If an object can be accessed from a live thread, it can't be garbage collected.

D. If object 1 refers to object 2, then object 2 can't be garbage collected.

Explanation:

This is a great way to think about when objects can be garbage collected.

Option A and B assume guarantees that the garbage collector never makes.

Option D is wrong because of the now famous islands of isolation scenario.

16. Which statement is true about assertions in the Java programming language?

A. Assertion expressions should not contain side effects.

B. Assertion expression values can be any primitive type.

C. Assertions should be used for enforcing preconditions on public methods.

D. An AssertionError thrown as a result of a failed assertion should always be handled by the enclosing method.

Explanation:

Option A is correct. Because assertions may be disabled, programs must not assume that the boolean expressions contained in assertions will be evaluated. Thus these expressions should be free of side effects. That is, evaluating such an expression should not affect any state that is visible after the evaluation is complete. Although it is not illegal for a boolean expression contained in an assertion to

have a side effect, it is generally inappropriate, as it could cause program behaviour to vary depending on whether assertions are enabled or disabled.

Assertion checking may be disabled for increased performance. Typically, assertion checking is enabled during program development and testing and disabled for deployment.

Option B is wrong. Because you assert that something is "true". True is Boolean. So, an expression must evaluate to Boolean, not int or byte or anything else. Use the same rules for an assertion expression that you would use for a while condition.

Option C is wrong. Usually, enforcing a precondition on a public method is done by condition-checking code that you write yourself, to give you specific exceptions.

Option D is wrong. "You're never supposed to handle an assertion failure"

Not all legal uses of assertions are considered appropriate. As with so much of Java, you can abuse the intended use for assertions, despite the best efforts of Sun's Java engineers to discourage you. For example, you're never supposed to handle an assertion failure. That means don't catch it with a catch clause and attempt to recover. Legally, however, AssertionError is a subclass of Throwable, so it can be caught. But just don't do it! If you're going to try to recover from something, it should be an exception. To discourage you from trying to substitute an assertion for an exception, the AssertionError doesn't provide access to the object that generated it. All you get is the String message.

Read More: Assertions

17. What will be the output of the program?

```java
public class ObjComp
{
    public static void main(String [] args )
    {
        int result = 0;
        ObjComp oc = new ObjComp();
        Object o = oc;

        if (o == oc)
            result = 1;
        if (o != oc)
            result = result + 10;
        if (o.equals(oc) )
            result = result + 100;
        if (oc.equals(o) )
            result = result + 1000;

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

```
        }
}
```

A. 1

B. 10

C. 101

D. 1101

Even though o and oc are reference variables of different types, they are both referring to the same object. This means that == will resolve to true and that the defaultequals() method will also resolve to true.

Read More: Java.lang Class

18. What will be the output of the program?

```java
public class Test178
{
    public static void main(String[] args)
    {
        String s = "foo";
        Object o = (Object)s;
        if (s.equals(o))
        {
            System.out.print("AAA");
        }
        else
        {
            System.out.print("BBB");
        }
        if (o.equals(s))
        {
            System.out.print("CCC");
        }
        else
        {
            System.out.print("DDD");
        }
    }
```

```
    }
```

B. AAADDD

C. BBBCCC

D. BBBDDD

Explanation:

Read More: Java.lang Class

19. What will be the output of the program?

```
int i = 1, j = 10;
do
{
    if(i++ > --j) /* Line 4 */
    {
        continue;
    }
} while (i < 5);
System.out.println("i = " + i + "and j = " + j); /* Line 9 */
```

A. i = 6 and j = 5

B. i = 5 and j = 5

C. i = 6 and j = 6

D. i = 5 and j = 6

Explanation:

This question is not testing your knowledge of the continue statement. It is testing your knowledge of the order of evaluation of operands. Basically the prefix and postfix unary operators have a higher

order of evaluation than the relational operators. So on line 4 the variable i is incremented and the variable j is decremented before the greater than comparison is made. As the loop executes the comparison on line 4 will be:

if(i > j)

if(2 > 9)

if(3 > 8)

if(4 > 7)

if(5 > 6) at this point i is not less than 5, therefore the loop terminates and line 9 outputs the values of i and j as 5 and 6 respectively.

The continue statement never gets to execute because i never reaches a value that is greater than j.

Read More: Java.lang Class

20. What will be the output of the program?

```java
public class ExamQuestion7
{
    static int j;
    static void methodA(int i)
    {
        boolean b;
        do
        {
            b = i<10 | methodB(4); /* Line 9 */
            b = i<10 || methodB(8);   /* Line 10 */
        }while (!b);
    }
    static boolean methodB(int i)
    {
        j += i;
        return true;
    }
    public static void main(String[] args)
    {
        methodA(0);
        System.out.println( "j = " + j );
    }
}
```

A. j = 0

B. j = 4

C. j = 8

D. The code will run with no output

Explanation:

The lines to watch here are lines 9 & 10. Line 9 features the non-shortcut version of the OR operator so both of its operands will be evaluated and therefore methodB(4) is executed.

However line 10 has the shortcut version of the OR operator and if the 1st of its operands evaluates to true (which in this case is true), then the 2nd operand isn't evaluated, so methodB(8) never gets called.

The loop is only executed once, b is initialized to false and is assigned true on line 9. Thus j = 4.

Read More: Java.lang Class

REMEMBER: PREPARATION IS KEY