lgsi('geekathon');

ANSWER GUIDE

JAVA TEST – 16 May 2012

1. Which causes a compiler error?

A. int[ ] scores = {3, 5, 7};

B. int [ ][ ] scores = {2,7,6}, {9,3,45};

C. String cats[ ] = {"Fluffy", "Spot", "Zeus"};

D. boolean results[ ] = new boolean [] {true, false, true};

Explanation:

Option B generates a compiler error: <identifier> expected. The compiler thinks you are trying to create two arrays because there are two array initialisers to the right of the equals, whereas your intention was to create one 3 x 3 two-dimensional array.

To correct the problem and make option B compile you need to add an extra pair of curly brackets:

int [ ] [ ] scores = { {2,7,6}, {9,3,45} };

Read More: Declarations & Access Control

2. Which three are valid method signatures in an interface?

1. private int getArea();
2. public float getVol(float x);
3. public void main(String [] args);
4. public static void main(String [] args);
5. boolean setFlag(Boolean [] test);

A. 1 and 2

B. 2, 3 and 5

C. 3, 4, and 5

D. 2 and 4

Explanation:

(2), (3), and (5). These are all valid interface method signatures.

(1), is incorrect because an interface method must be `public`; if it is not explicitly declared `public` it will be made public implicitly. (4) is incorrect because interface methods cannot be `static`.

Read More: Declarations & Access Control

3. What will be the output of the program?

```
class Super
{
    public Integer getLength()
    {
        return new Integer(4);
    }
}

public class Sub extends Super
{
    public Long getLength()
    {
        return new Long(5);
    }

    public static void main(String[] args)
    {
        Super sooper = new Super();
        Sub sub = new Sub();
        System.out.println(
        sooper.getLength().toString() + "," + sub.getLength().toString() );
    }
}
```

A. 4, 4

B. 4, 5

C. 5, 4

D. Compilation fails.

Explanation:

Option D is correct, compilation fails - The return type of `getLength( )` in the super class is an object of reference type Integer and the return type in the sub class is an object of reference type `Long`. In other words, it is not an override because of the change in the return type and it is also not an overload because the argument list has not changed.

Read More: Declarations & Access Control

4. What will be the output of the program?

```
class Test
{
    public static void main(String [] args)
    {
        int x= 0;
        int y= 0;
        for (int z = 0; z < 5; z++)
        {
            if (( ++x > 2 ) && (++y > 2))
            {
                x++;
            }
        }
        System.out.println(x + " " + y);
    }
}
```

A. 5 2

B. 5 3

C. 6 3

D. 6 4

Explanation:

In the first two iterations `x` is incremented once and `y` is not because of the short circuit`&&` operator. In the third and forth iterations `x` and `y` are each incremented, and in the fifth iteration `x` is doubly incremented and `y` is incremented.

Read More: Operators & Assignments

5. What will be the output of the program?

```
int i = 1, j = -1;
switch (i)
{
    case 0, 1: j = 1; /* Line 4 */
    case 2: j = 2;
    default: j = 0;
}
System.out.println("j = " + j);
```

A. j = -1

B. j = 0

C. j = 1

D. Compilation fails.

Explanation:

The case statement takes only a single argument. The case statement on line 4 is given two arguments so the compiler complains.

Read More: Flow Control

6. What will be the output of the program?

```
for(int i = 0; i < 3; i++)
{
    switch(i)
    {
        case 0: break;
```

```
        case 1: System.out.print("one ");
        case 2: System.out.print("two ");
        case 3: System.out.print("three ");
    }
}
System.out.println("done");
```

A. done

B. one two done

C. one two three done

D. one two three two three done

Explanation:

The variable i will have the values 0, 1 and 2.

When `i` is `0`, nothing will be printed because of the break in `case 0`.

When `i` is `1`, "one two three" will be output because `case 1`, `case 2` and `case 3` will be executed (they don't have break statements).

When `i` is `2`, "two three" will be output because `case 2` and `case 3` will be executed (again no break statements).

Finally, when the for loop finishes "done" will be output.

Read More: Flow Control

7. What will be the output of the program?

```
boolean bool = true;
if(bool = false) /* Line 2 */
{
    System.out.println("a");
}
else if(bool) /* Line 6 */
{
    System.out.println("b");
}
```

```
else if(!bool) /* Line 10 */
{
    System.out.println("c"); /* Line 12 */
}
else
{
    System.out.println("d");
}
```

A. a

B. b

C. c

D. d

Explanation:

Look closely at line 2, is this an equality check (==) or an assignment (=). The condition at line 2 evaluates to false and also assigns false to `bool`. `bool` is now false so the condition at line 6 is not true. The condition at line 10 checks to see if `bool` is not true (`if !(bool == true)`), it isn't so line 12 is executed.

Read More: Flow Control

8. What will be the output of the program?

```
class Exc0 extends Exception { }
class Exc1 extends Exc0 { } /* Line 2 */
public class Test
{
    public static void main(String args[])
    {
        try
        {
            throw new Exc1(); /* Line 9 */
        }
        catch (Exc0 e0) /* Line 11 */
        {
```

```
            System.out.println("Ex0 caught");
        }
        catch (Exception e)
        {
            System.out.println("exception caught");
        }
    }
}
```

A. Ex0 caught

B. exception caught

C. Compilation fails because of an error at line 2.

D. Compilation fails because of an error at line 9.

Explanation:

An exception `Exc1` is thrown and is caught by the catch statement on line 11. The code is executed in this block. There is no finally block of code to execute.

Read More: Exceptions

9. Which answer most closely indicates the behavior of the program?

```
public class MyProgram
{
    public static void throwit()
    {
        throw new RuntimeException();
    }
    public static void main(String args[])
    {
        try
        {
            System.out.println("Hello world ");
            throwit();
            System.out.println("Done with try block ");
        }
        finally
```

```
        {
            System.out.println("Finally executing ");
        }
    }
}
```

A. The program will not compile.

B. The program will print Hello world, then will print that a `RuntimeException` has occurred, then will print Done with try block, and then will print Finally executing.

C. The program will print Hello world, then will print that a `RuntimeException` has occurred, and then will print Finally executing.

D. The program will print Hello world, then will print Finally executing, then will print that a `RuntimeException` has occurred.

Explanation:

Once the program throws a `RuntimeException` (in the `throwit()` method) that is not caught, the finally block will be executed and the program will be terminated. If a method does not handle an exception, the finally block is executed before the exception is propagated.

Read More: Exceptions

10. Which interface provides the capability to store objects using a key-value pair?

A. Java.util.Map

B. Java.util.Set

C. Java.util.List

D. Java.util.Collection

Explanation:

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

Read More: Objects & Collections

11. Which of the following statements about the `hashcode()` method are incorrect?

1. The value returned by `hashcode()` is used in some collection classes to help locate objects.
2. The `hashcode()` method is required to return a positive `int` value.
3. The `hashcode()` method in the `String` class is the one inherited from `Object`.
4. Two new empty String objects will produce identical hashcodes.

A. 1 and 2

B. 2 and 3

C. 3 and 4

D. 1 and 4

Explanation:

(2) is an incorrect statement because there is no such requirement.

(3) is an incorrect statement and therefore a correct answer because the hashcode for a string is computed from the characters in the string.

Read More: Objects & Collections

12. What will be the output of the program?

```
public class TestObj
{
    public static void main (String [] args)
    {
        Object o = new Object() /* Line 5 */
        {
            public boolean equals(Object obj)
            {
                return true;
            }
        }       /* Line 11 */

        System.out.println(o.equals("Fred"));
```

```
    }
}
```

A. It prints "true".

B. It prints "Fred".

C. An exception occurs at runtime.

D. Compilation fails

Explanation:

This code would be legal if line 11 ended with a semicolon. Remember that line 5 is a statement that doesn't end until line 11, and a statement needs a closing semicolon!

Read More: Inner Classes

13. Which of the following will directly stop the execution of a Thread?

A. wait()

B. notify()

C. notifyall()

D. exits synchronized code

Explanation:

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

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

Option C is wrong. `notifyAll()` - wakes up all threads that are waiting on this object's monitor.

Option D is wrong. Typically, releasing a lock means the thread holding the lock (in other words, the thread currently in the synchronized method) exits the synchronized method. At that point, the lock is free until some other thread enters a synchronized method on that object. Does entering/exiting

synchronized code mean that the thread execution stops? Not necessarily because the thread can still run code that is not synchronized. I think the word directly in the question gives us a clue. Exiting synchronized code does not directly stop the execution of a thread.

Read More: Threads

14. Which class or interface defines the `wait()`, `notify()`,and `notifyAll()` methods?

A. Object

B. Thread

C. Runnable

D. Class

Explanation:

The `Object` class defines these thread-specific methods.

Option B, C, and D are incorrect because they do not define these methods. And yes, the Java API does define a class called `Class`, though you do not need to know it for the exam.

Read More: Threads

15. What will be the output of the program?

```
public class ThreadDemo
{
    private int count = 1;
    public synchronized void doSomething()
    {
        for (int i = 0; i < 10; i++)
            System.out.println(count++);
    }
    public static void main(String[] args)
    {
        ThreadDemo demo = new ThreadDemo();
        Thread a1 = new A(demo);
        Thread a2 = new A(demo);
```

```
        a1.start();
        a2.start();
    }
}
class A extends Thread
{
    ThreadDemo demo;
    public A(ThreadDemo td)
    {
        demo = td;
    }
    public void run()
    {
        demo.doSomething();
    }
}
```

A. It will print the numbers 0 to 19 sequentially

B. It will print the numbers 1 to 20 sequentially

C. It will print the numbers 1 to 20, but the order cannot be determined

D. The code will not compile.

Explanation:

NA

Read More: Threads

16. Which two can be used to create a new Thread?

1. Extend `java.lang.Thread` and override the `run()` method.
2. Extend `java.lang.Runnable` and override the `start()` method.
3. Implement `java.lang.Thread` and implement the `run()` method.
4. Implement `java.lang.Runnable` and implement the `run()` method.
5. Implement `java.lang.Thread` and implement the `start()` method.

A. 1 and 2

B. 2 and 3

C. 1 and 4

D. 3 and 4

Explanation:

There are two ways of creating a thread; extend (sub-class) the `Thread` class and implement the `Runnable` interface. For both of these ways you must implement (override and not overload) the `public void run()` method.

(1) is correct - Extending the Thread class and overriding its run method is a valid procedure.

(4) is correct - You must implement interfaces, and runnable is an interface and you must also include the run method.

(2) is wrong - Runnable is an interface which implements not Extends. Gives the error: (No interface expected here)

(3) is wrong - You cannot implement `java.lang.Thread` (This is a Class). (Implements Thread, gives the error: Interface expected). Implements expects an interface.

(5) is wrong - You cannot implement `java.lang.Thread` (This is a class). You Extend classes, and Implement interfaces. (Implements Thread, gives the error: Interface expected)

Read More: Threads

17. The following block of code creates a Thread using a Runnable target:

```
Runnable target = new MyRunnable();
Thread myThread = new Thread(target);
```

Which of the following classes can be used to create the target, so that the preceding code compiles correctly?

A. public class MyRunnable extends Runnable{public void run(){}}

B. public class MyRunnable extends Object{public void run(){}}

C. public class MyRunnable implements Runnable{public void run(){}}

D. public class MyRunnable implements Runnable{void run(){}}

Explanation:

The class correctly implements the Runnable interface with a legal `public void run()` method.

Option A is incorrect because interfaces are not extended; they are implemented.

Option B is incorrect because even though the class would compile and it has a valid`public void run()` method, it does not implement the `Runnable` interface, so the compiler would complain when creating a Thread with an instance of it.

Option D is incorrect because the `run()` method must be `public`.

Read More: Threads

18. Select how you would start the program to cause it to print: `Arg is 2`

```
public class Myfile
{
    public static void main (String[] args)
    {
        String biz = args[1];
        String baz = args[2];
        String rip = args[3];
        System.out.println("Arg is " + rip);
    }
}
```

A. java Myfile 222

B. java Myfile 1 2 2 3 4

C. java Myfile 1 3 2 2

D. java Myfile 0 1 2 3

Explanation:

Arguments start at array element `0` so the fourth arguement must be `2` to produce the correct output.

Read More: Java.lang Class

19. What will be the output of the program?

```
int i = (int) Math.random();
```

A. i = 0

B. i = 1

C. value of i is undetermined

D. Statement causes a compile error

Explanation:

`Math.random()` returns a `double` value greater than or equal to `0` and less than `1`. Its value is stored to an `int` but as this is a narrowing conversion, a cast is needed to tell the compiler that you are aware that there may be a loss of precision.

The value after the decimal point is lost when you cast a `double` to `int` and you are left with `0`.

Read More: Java.lang Class

20. What will be the output of the program?

```
class A
{
    public A(int x){}
}
class B extends A { }
public class test
{
    public static void main (String args [])
    {
        A a = new B();
        System.out.println("complete");
    }
}
```

A. It compiles and runs printing nothing

B. Compiles but fails at runtime

C. Compile Error

D. Prints "complete"

Explanation:

No constructor has been defined for `class B` therefore it will make a call to the default constructor but since `class B` extends `class A` it will also call the `Super()` default constructor.

Since a constructor has been defined in `class A` java will no longer supply a default constructor for `class A` therefore when `class B` calls `class A`'s default constructor it will result in a compile error.

Read More: Java.lang Class

REMEMBER: PREPARATION IS KEY