Cuestionarios Java

1. ¿Qué forma es incorrecta cuando se crea un array?

A. `int[] n = {1,2,3,4,5};`
B. `int[10] v = new int;`
C. `int p[] = new int[5];`

2. ¿Qué línea tiene un error?

```
00 int p = 1;
01 byte b;
02 while(p--) {
03 b=(byte) p;
04 System.out.println(b);
05 }
```

A. Línea 2
B. Línea 3
C. Línea 4
D. Línea 5

3. ¿Cuál será el resultado?

```
int k = 5;
int v = 2;
if(k>0 || (++v > 1)) {
  for(int i = 0; i < v; i++) {
       k++;
  }
}
System.out.println(k);
```

A. 7
B. 6
C. 5
D. 8
E. 4

4. Una de las siguientes afirmaciones sobre clases abstractas no es correcta.

A. No es posible crear objetos de una clase abstracta.
B. El modificador abstract no puede ser aplicado sobre un atributo.
C. Una clase abstracta no puede tener constructores definidos explícitamente.
D. Una subclase de una clase abstracta puede ser también abstracta.

5. ¿Cuál es el error de compilación?

A. `int k = 2500;`
B. `float r = 3.7;`
C. `byte p = (byte) k;`

6. a, b y c son variables de tipo int que han sido inicializadas con algún valor. Indicar qué instrucción es incorrecta:

A. `a = c ^b;`
B. `a = b*c++;`
C. `a = b && c;`

7. a, b y c son variables de tipo int que han sido inicializadas con algún valor. Indicar qué instrucción es incorrecta:

A. `if (a=(b+c))`
B. `if (++a>(b-c))`
C. `if ((a+b)==0)`

8. De las siguientes palabras ¿Cuál no se corresponde con tipos básicos del lenguaje?

A. double
B. byte
C. boolean
D. Long
E. int

9. Dos de los siguientes métodos no pueden pertenecer a la misma clase.

A. `public void metodotest(int k) {}`
B. `public int metodotest(){}`
C. `public int metodotest(int k) {}`
D. `public void metodotest(String s) {}`

A. A y B
B. C y D
C. D y A
D. A y C
E. B y C

10. Dada la siguiente clase:

```
class Ejemplo {
     protected void generar(String s, int p) {}
}
```

¿Cuál de los siguientes métodos no podría estar en una subclase de Ejemplo?

A. `void generar(String s, int p) {…}`
B. `public void generar() {…}`
C. `protected int generar(int k) {…}`
D. `public void generar(String s, int p) {…}`

A. A
B. A y B
C. A y C
D. C y D
E. C y A

11. ¿Qué ocurre si compilamos el siguiente código?

```
interface A {
     public void m();
}
class B implements A {
     public void m2() {};
}
```

A. Compila correctamente
B. No compila porque la declaración de la interfaz A no es correcta
C. No compila porque la clase B no implementa los métodos del interfaz A

12.¿Qué se presenta por pantalla al ejecutar este fragmento de código?

```
String str = "Una Cadena";
str.toUpperCase();
System.out.println(str);
```

A. UNA CADENA
B. una cadena
C. Una Cadena

13.¿Qué modificador se emplea para declarar una variable de clase?

A. protected
B. static
C. final

14.Dado el siguiente código:

```
String s1 = "hola";
String s2 = s1;
```

¿Cuál de las siguientes expresiones se evalúa a true?

A. s1.equals(s2)
B. s1==s2
C. Ambas

15.¿Qué literal se emplea para identificar que una variable no tiene asignado ningún objeto?

A. final
B. null
C. 0

16.Para que la clase public MiClase compile, ¿cuál de las siguientes afirmaciones es correcta?

A. MiClase ha de estar definida en el archivo MiClase.java
B. MiClase ha de definir el método main
C. MiClase ha de estar dentro del paquete MiClase

17.¿Qué ocurre al intentar compilar y ejecutar el siguiente programa?

```
class Padre {
     public Padre() {
          System.out.println("Creando Padre");
     }
}
class Hijo extends Padre {
     public Hijo() {
          System.out.println("Creando Hijo");
     }
     public static void main(String [] args) {
          Padre p=new Hijo();
     }
}
```

A. No compila
B. Presenta por pantalla "Creando Hijo"
C. Presenta por pantalla primero "Creando Padre" y luego "Creando Hijo"

18.¿Qué es necesario hacer para que compile el siguiente código?

```
public class A {
     abstract int metodo1();
     void metodo2() {};
}
```

A. Añadir el modificador abstract en la primera línea
B. Añadir el modificador private a la primera línea
C. Añadir el modificador abstract a la tercera línea

19.¿Cuál de las siguientes declaraciones es correcta?

A. `public virtual void metodo();`
B. `public abstract void metodo();`
C. `abstract void metodo() {};`

20. ¿Cuándo está permitida la siguiente declaración de clase?

```
public class A extends B implements C, D { }
```

A. Si B es una clase y C y D son interfaces
B. Si B, C y D son interfaces
C. Nunca porque Java no permite herencia múltiple

21.Siempre que la conversión numérica no sea posible, la JVM lanzará una excepción del tipo:
A. java.lang.NumberFormatException
B. java.lang.FormatException
C. java.math.FormatException
D. java.math.Exception
E. java.math.NULLNumber

22.Usando el constructor:

```
public Boolean(String s);
```

A. Tanto "true" como "TRUE" obtendrán un valor Boolean true.
B. Solamente "true" obtendrá un valor Boolean true.
C. Tanto "true" como 1 obtendrán un valor Boolean true.

23.El Autoboxing/Auto-unboxing se añade en Java 5, permitiendo que las conversiones entre tipos primitivos y sus wrappers se hagan de forma automática.
A. Verdadero
B. Falso

24.Sobre la clase Math:
A. No se puede instanciar.
B. Es una clase pensada para obtener números aleatorios.
C. Contiene funciones matemáticas útiles.

A. Solamente B es correcta.
B. A y C son correctas.
C. A y B son correctas.
D. Todas son correctas.
E. Solamente A es correcta.

25.¿Las clases Integer y Double proporcionan el método longValue()?
A. Verdadero
B. Falso

26.Los nombres de las clases wrapper son: Char, Byte, Short, Int, Long, Float y Double.
A. Verdadero
B. Falso

27.¿Qué mensaje aparecerá en pantalla?

```
String s = new String("Cad nueva");
String p = new String("Cad nueva");
if(s == p) {
      System.out.println("Iguales");
}else {
      System.out.println("No Iguales");
}
```

A. Se produce una excepción
B. Aparecerá "Iguales"
C. Se producirá un error de compilación
D. Aparecerá "No Iguales”

28.¿Qué mensaje aparecerá en pantalla?

```
String s = new String("Cad nueva");
String p = s;
if(s == p) {
      System.out.println("Iguales");
}else {
      System.out.println("No Iguales");
}
```

A. Se produce una excepción
B. Aparecerá "Iguales"
C. Se producirá un error de compilación
D. Aparecerá "No Iguales"

29.¿Qué mensaje aparecerá en pantalla?

```
String s = new String("Cad nueva");
String p = s;
if(s > p) {
      System.out.println("s es mayor");
}else {
      System.out.println("p es mayor");
}
```

A. Se produce una excepción
B. Aparecerá "s es mayor"
C. Se producirá un error de compilación
D. Aparecerá "s es menor"

30.La clase String es una cadena inmutable. ¿cómo entonces se puede modificar su contenido?
A. No se puede.
B. Sus métodos siempre devuelven un String nuevo.
C. Debido a B el uso intensivo de éstos reduce el rendimiento.
D. Todas son correctas.

31. La única diferencia entre StringBuffer y StringBuilder es que StringBuilder tiene sus métodos sincronizados.
A. Verdadero
B. Falso

32. Para crear números aleatorios en Java necesitamos utilizar una librería que no pertenece al JDK.
A. Verdadero
B. Falso

33. Para crear números aleatorios en Java usaremos la clase Random o Math.random() dependiendo el caso en que estemos.
A. Verdadero
B. Falso

34. No se recomienda utilizar Random como generador de números aleatorios para aplicaciones en las que sea crítica la seguridad.
A. Verdadero
B. Falso

35. El principal uso de los wrappers es contener los tipos simples cuando estos se tienen que introducir dentro de una estructura que solo puede contener objetos.
A. Verdadero
B. Falso

36. Todas las clases wrapper proporcionan el método intValue() y byteValue() y toNumberString().
A. Verdadero
B. Falso

37. ¿Qué ocurrirá al ejecutar este código?

```
import java.io.*
public class Ejemplo {
      public void limitador(String[] args) {
            if(args.length() > 10) {
                  throw new IOException();
            }
      }
}
```

A. Se lanza la excepción
B. Fallo de compilación
C. Error en tiempo de ejecución

38. Para enviar el volcado de pila de la excepción desde un catch a la consola, utilizaremos:
A. `System.out.println(e.printStackTrace());`
B. `e.getMessage();`
C. `System.out.println(e.getMessage());`
D. `e.printStackTrace();`

39. Si se produce un Error (no una excepción), casi seguro que se romperá el programa o incluso la propia JVM.
A. Verdadero
B. Falso

40. Qué muestra por pantalla:

```
int s = 0;
int[] m = {1, 3, 5, 7};

try {
     for (int i = 0; i <= m.length; i++) {
          s += m[i];
     }
     System.out.println("El total es: " + s);
} catch (NullPointerException e) {
     System.out.println("Fallo en el array");
} catch (Exception e) {
     System.out.println("Error");
} finally {
     System.out.println(" Total final: " + s);
}
```

A. El total es: 16 Total final: 16
B. Fallo en el array
C. El total es: 16
D. Nada
E. Error Total final: 16

41. Indicar qué afirmación es incorrecta:
A. Si no se ha definido ningún catch para el try, la utilización de finally se hace obligatoria.
B. No es posible definir dos bloques finally en un mismo try.
C. Si se produce una excepción en try y ningún bloque catch la captura, las instrucciones definidas en finally no se ejecutarán.
D. Aunque no se produzca la excepción en try, el bloque finally se ejecutará.

42. Si queremos definir tres bloques catch para controlar excepciones de tipo RuntimeException, SQLException y ClassCastException, ¿cuál debe ser el orden de colocación de los mismos?
A. ClassCastException da igual, pero SQLException debe ir antes de RuntimeException.
B. RuntimeException, ClassCastException y SQLException.
C. SQLException , ClassCastException y RuntimeException.
D. SQLException da igual, pero ClassCastException debe ir antes de RuntimeException.

43. Un programador no puede lanzar un error.
A. Verdadero
B. Falso

44. Típicamente la JVM lanza errores mientras que los programadores lanzan excepciones.
A. Verdadero
B. Falso

45. Si definimos una clase de excepción personalizada que herede de Exception, ¿será de tipo checked?
A. Verdadero
B. Falso

46. Java nos permite crear nuestras propias excepciones heredando de Exception y RuntimeException.
A. Verdadero
B. Falso

47. What is the result of attempting to compile and run this code ?

```
class A extends Exception{}
class B extends A{}
class C extends B{}
public class Test {
  static void aMethod() throws C{ throw new C(); }
  public static void main(String[] args){
     int x = 10;
    try { aMethod(); }
    catch(A e) { System.out.println("Error A");}
    catch(B e) { System.out.println("Error B");}
  }
}
```

A. Compiler error
B. It will print "Error A"
C. It will print "Error B"
D. The exception will go uncaught by both catch blocks

48. Is this code legal ?

```
class ExceptionA extends Exception {}
class ExceptionB extends ExceptionA {}
public class Test{
      void thrower() throws ExceptionB{
            throw new ExceptionB();
      }
    public static void main(String[] args){
            Test t = new Test();
            try{t.thrower();}
            catch(ExceptionA e) {}
            catch(ExceptionB e) {}
      }
}
```

A. Yes
B. No

49. Is this code legal ?

```
class ExceptionA extends Exception {}
class ExceptionB extends ExceptionA {}
public class Test{
      void thrower() throws ExceptionA{
            throw new ExceptionA();
    }
    public static void main(String[] args){
            Test t = new Test();
            try{t.thrower();}
            catch(ExceptionB e) {}
      }
}
```

A. Yes
B. No

50. Is this code legal ?

```
class ExceptionA extends Exception {}
class ExceptionB extends ExceptionA {}
class A{
      void thrower() throws ExceptionA{
            throw new ExceptionA();
     }
}
public class B extends A{
      void thrower() throws ExceptionB{
            throw new ExceptionB();
     }
}
```

A. Yes
B. No

51. Given

```
11. public static void parse(String str) {
12.         try {
13.               float f = Float.parseFloat(str);
14.         } catch (NumberFormatException nfe) {
15.               f = 0;
16.         } finally {
17.               System.out.println(f);
18.         }
19. }
20. public static void main(String[] args) {
21.         parse("invalid");
22. }
```

What is the result?
A. 0.0
B. Compilation fails.
C. A ParseException is thrown by the parse method at runtime.
D. A NumberFormatException is thrown by the parse method at runtime.

52. Given

```
84. try {
85.    ResourceConnection con = rf.getConnection();
86.    r = con.query("GET INFO FROM CUSTOMER");
87.    info = r.getData();
88.    con.close();
89. } catch (ResourceException re) {
90.    errorLog.write(re.getMessage());
91. }
92. return info;
```

Which statement is true if a ResourceException is thrown on line 86?
A. Line 92 will not execute.
B. The connection will not be retrieved in line 85.
C. The resource connection will not be closed on line 88.
D. The enclosing method will throw an exception to its caller.

53. Given

```
11. public static void main(String[] args) {
12.  try {
13.    args = null;
14.    args[0] = "test";
15.    System.out.println(args[0]);
16.  } catch (Exception ex) {
17.    System.out.println("Exception");
18.  } catch (NullPointerException npe) {
19.    System.out.println("NullPointerException");
20.  }}
```

What is the result?
A. test
B. Exception
C. Compilation fails
D. NullPointerException

54. Given

```
11. class A {
12.   public void p(){ System.out.print("A,"); } }
13. class B extends A {
14.   public void p() throws IOException {
15.        super.p();
16.        System.out.print("B,");
17.        throw new IOException();
18.   }
19.   public static void main(String[] args) {
20.     try { new B().p(); }
21.     catch (IOException e)
22.     { System.out.println("Exception"); }}}
```

What is the result?
A. Exception
B. A,B,Exception
C. Compilation fails because of an error in line 20.
D. Compilation fails because of an error in line 14.
E. A NullPointerException is thrown at runtime.

55. Given

```
11. static void test() throws Error {
12.    if (true) throw new AssertionError();
13.    System.out.print("test ");
14. }
15. public static void main(String[] args) {
16.   try { test(); }
17.   catch (Exception ex) {
18.      System.out.print("exception "); }
19.    System.out.print("end ");}
```

What is the result?
A. end
B. Compilation fails.
C. exception end
D. exception test end
E. A Throwable is thrown by main.
F. An Exception is thrown by main.

56. Given

```
33. try {
34.     // some code here
35. } catch (NullPointerException e1) {
36.     System.out.print("a");
37. } catch (RuntimeException e2) {
38.     System.out.print("b");
39. } finally {
40.     System.out.print("c");
41. }
```

What is the result if a NullPointerException occurs on line 34?
A. c
B. a
C. ab
D. ac
E. bc
F. abc

57.¿Qué ocurrirá al ejecutar este código?

```
11. public static Iterator reverse(List list) {
12.   Collections.reverse(list);
13.   return list.iterator();
14. }
15. public static void main(String[] args) {
16.   List ls = new ArrayList();
17.   ls.add(" 1"); ls.add("2"); ls.add("3");
18.   for (Object obj: reverse(ls))
19.      System.out.print(obj + ",");
20. }
```

C. 3,2, 1,
D. 1,2,3,
E. Error de compilación
F. El código se ejecuta pero no genera salida
G. Se lanza una excepción en ejecución

58.¿Qué ocurrirá al ejecutar este código?

```
11. public static Collection get() {
12.   Collection sort = new LinkedList();
13.   sort.add("B"); sort.add("C"); sort.add("A");
14.    return sort;
15. }
16. public static void main(String[] args) {
17.   for (Object obj: get()) {
18.      System.out.print(obj + ", ");
19.   }
20. }
```

A. A, B, C,
B. B, C, A,
C. Error de compilación
D. El código se ejecuta pero no genera salida
E. Se lanza una excepción en ejecución

59.Un programador tiene un algoritmo que requiere el uso de una colección que tenga como característica una eficiente implementación de add(index, Object), pero NO necesita el acceso rápido a elementos. ¿Qué implementación soporta estos requisitos?

A. java.util.Queue
B. java.util.ArrayList
C. java.util.LinearList
D. java.util.LinkedList

60.Dado el siguiente código, selecciona la respuesta correcta.

```
1. public class Person {
2.  private String name;
3.  public Person(String name){this.name = name;}
4.  public boolean equals(Person p) {
5.         return p.name.equals(this.name);
6.  }
7. }
```

A. El método equals no está sobrescrito adecuadamente
B. La compilación falla porque el atributo privado p.name no puede ser accedido desde la línea 5
C. Para trabajar correctamente con estructuras basadas en hash, esta clase debe implementar también el método hashCode
D. Cuando se añaden objetos Person a una colección java.util.Set, el método equals definido en la línea 4 previene duplicados

61.Un HashSet empleará objetos de esta clase, selecciona la respuesta correcta.

```
11. public class Person {
12.   private String name;
13.   public Person(String name) {
14.        this.name = name;
15.   }
16.   public boolean equals(Object o) {
17.        if( !(o instanceof Person )) return false;
18.        Person p = (Person) o;
19.        return p.name.equals(this.name);
20.   }
21. }
```

A. La compilación falla porque el método hashCode() no se ha sobrescrito
B. El HashSet podría contener múltiples objetos Person con el mismo nombre
C. Todos los objetos Person tendrán el mismo hashCode ya que el método hashCode() no ha sido sobrescrito
D. Si un HashSet contiene más de un objeto Person con el nombre "Pepe", cuando se borra otro objeto Person con el mismo nombre todas las instancias serán borradas

62.Un programador está desarrollando una clase Key, que será usada como clave en un HashMap. ¿Qué dos métodos deberían ser sobrescritos para asegurar que la clase key funciona correctamente como clave?

A. public int hashCode()
B. public boolean equals(Key k)
C. public int compareTo(Object o)
D. public boolean equals(Object o)
E. public int compareTo(Key k)

63.HashSet, TreeSet y LinkedHashSet son implementaciones de Set y List:

A. Verdadero
B. Falso

64.Selecciona la respuesta correcta:
A. El método hashCode sirve para testear la igualdad y desigualdad de objetos para una clase
B. El método hashCode se utiliza en SortedSet para ordenar los elementos dentro del conjunto
C. El método hashCode puede ser utilizado para testear desigualdad, pero no para la igualdad
D. El método hashCode es usado por HashSet para agrupar los elementos mediante la función hash

65.Indica cuál será el resultado:
```
11. public static void append(List list) {
12.    list.add("0042"); }
13. public static void main(String[] args) {
14.   List<Integer> ls = new ArrayList<Integer>();
15.   append(ls);
16.   System.out.println(ls.get(0));
17. }
```

A. 42
B. 0042
C. Excepción en tiempo de ejecución
D. La compilación falla en la línea 15
E. La compilación falla en la línea 16

66.Indica cuál será el resultado:
```
1. public class LetterASort {
2.  public static void main(String[] args) {
3.   List<String> ss = new ArrayList<String>();
4.   ss.add("aAaA");
5.   ss.add("AaA");
6.   ss.add("aAa");
7.   ss.add("AAaa");
8.   Collections.sort(ss);
9.   for (String s: ss) {
10.     System.out.print(s + " ");
11.  }
12. }
13.}
```

A. La compilación falla
B. aAaA aAa AAaa AaA
C. AAaa AaA aAa aAaA
D. AaA AAaa aAaA aAa
E. AaA AAaa aAaA aAa
F. Excepción en tiempo de ejecución

67.HashMap almacena elementos en orden:
A. Verdadero
B. Falso

68.Las colecciones evitan que tengamos que crear nuestras propias estructuras de datos en la mayoría de los casos:
A. Verdadero
B. Falso

69.TreeSet almacena elementos en orden:
A. Verdadero
B. Falso

70.Indica cuál será la implementación adecuada para el método hashCode en la clase Person:

```
11. public class Person {
12.  private String name, comm;
13.  private int age;
14.  public Person(String n, int a, String c) {
15.    name = n; age = a; comm = c;
16.  }
17.  public boolean equals(Object o) {
18.    if(! (o instanceof Person)) return false;
19.    Person p = (Person)o;
20.    return age == p.age && name.equals(p.name);
21.  }
22. }
```

A. `return super.hashCode();`
B. `return name.hashCode()+age*7;`
C. `return name.hashCode()+comm.hashCode()/2;`
D. `return name.hashCode()+comm.hashCode()/2-age*3;`