BÁSICOS.

1. La forma canónica en C++ incluye el constructor copia, el destructor, el operador asignación, el constructor por defecto y cualquier otro constructor parametrizado.

2. En JAVA la forma canónica incluye constructor, equals, hashCode, toString y clone.

3. Tanto la herencia protegida como la privada permiten a una clase derivada acceder a las propiedades privadas de la clase base.

4. Independientemente del tipo de herencia la clase base siempre podrá acceder a lo público, protegido y default heredado pero no a lo privado.

5. Una clase abstracta se caracteriza por declarar todos sus métodos de instancia como abstractos.

6. En JAVA, la clase deberá contener al menos un método abstracto para ser declarada como tal, en C++ no.

7. La siguiente sentencia class S{ public final void f() {}} constituye una interfaz en JAVA.

8. La siguiente sentencia interface S{ void f(); } constituye una interfaz en C++.

9. Desde un método de una clase derivada nunca puede invocarse un método implementando con idéntica signatura de su clase.

10. En Java los métodos de instancia con polimorfismo puro pero no abstractos tienen enlace dinámico.

11. Una operación de clase solo puede acceder directamente a atributos de clase.

12. Una operación de instancia puede acceder directamente a atributos de clase y de instancia.

13. Una operación de clase no es una función miembro de la clase.

14. En la misma clase podemos definir constructores con distinta visibilidad.

15. El modificador static es correcto para JAVA pero no para C++ cuando lo usamos con constructores.

16. Un buen diseño orientado a objetos se caracteriza por un alto acoplamiento y una baja cohesión.

17. Mediante la herencia de implementación heredamos la implementación de los métodos de la clase base pero no la interfaz de esta.

18. La genericidad es un tipo de polimorfismo.

19. El polimorfismo es un tipo de genericidad.

20. En JAVA y en C++ todas las clases son polimorficas.

21. El downcasting en JAVA y en C++ es siempre dinámico.

22. Si la conversión, downcasting, es fuera de la jerarquía de herencia JAVA dará error de ejecución.

23. El downcasting implica deshacer el principio de sustitución.

GESTIÓN DE ERRORES.

24. En Java, si no se captura una excepción lanzada por un método da error de compilación.

25. La instrucción throw en JAVA solo permite lanzar objetos que son de la clase throwable o clases derivadas de esta.

26. Uno de los objetivos del tratamiento de errores mediante excepciones es el manejo de errores del resto del código.

27. Si no se captura una excepción lanzada por un método, el programa no advierte que ha ocurrido error y continúa su ejecución normalmente.

28. En JAVA, siempre es obligatorio especificar que excepciones verificadas (checked exceptions) lanza un método mediante una cláusula throws tras la lista de argumentos.

29. Si se produce una excepción el método que la provoca se cancela y se continúa la ejecución en el método que llamo a este.

30. Si se produce una excepción en un constructor el objeto se construirá con los valores por defecto.

31. Todas las excepciones son checked exception salvo las runtime que son uncheked exception.

32. La cláusula throws de un método incluirá todas las excepciones uncheked exception que puedan producirse en este y no estén dentro del bloque try catch que las capture.

33. El orden de las excepciones en los bloques catch no es relevante.

34. Podemos poner un bloque finally sin poner bloques catch.

35. El bloque finally solo se ejecutará si se produce alguna excepción en el bloque try al que esté asociado.

36. IllegalArgumentException, ArrayIndexOutOfBoundsException, ClassCastException y IOException son excepciones del tipo runtimeException y por tanto no es necesario capturarlas ni indicar throws en el método en el que se provoquen.

GENERICIDAD.

37. La genericidad se considera una característica opcional de los lenguajes.

38. La genericidad se considera una característica opcional de los lenguajes orientados a objetos.

39. No se puede derivar una clase genérica de una clase no genérica.

40. En JAVA no podemos crear interfaces genéricas.

41. En los métodos genéricos solo podremos usar los métodos definidos en Object.

42. En la genericidad restringida solo podremos usar los métodos de Object al implementar los métodos.

43. Los métodos genéricos no se pueden sobrecargar ni sobrescribir.

REFLEXIÓN.

44. La API de reflexión de JAVA incluye métodos para obtener la signatura de todos los métodos.

45. La reflexión permite que un programa obtenga información sobre si mismo en tiempo de ejecución.

46. Para usar reflexión en Java hemos de conocer el nombre de las clases en tiempo de compilación.

47. En Java el concepto de meta-clase se representa con la clase Class.

48. Con el uso de la reflexión solo podemos invocar métodos de instancia.

49. La reflexión sólo es útil para trabajar con componentes JavaBeans.

50. La reflexión es demasiado compleja para usarla en aplicaciones de propósito general.

51. La reflexión reduce el rendimiento de las aplicaciones.

52. La reflexión no puede usarse en aplicaciones certificadas con el estándar 100% Pure Java.

53. Mediante reflexión podemos saber cuales son las clases derivadas de una clase dada.

54. Mediante reflexión no podemos saber cual es el método que se está ejecutando en un determinado momento.

55. Podemos usar reflexión para encontrar un método heredado (solo hacia arriba) y reducir código condicional.

REFACTORIZACIÓN.

56. La refactorización debe hacerse siempre apoyandonos en un conjunto de tests completo y robusto.

57. Una clase con un gran número de métodos y atributos es candidata a ser refactorizada.

58. Los métodos grandes (con muchas instrucciones) son estructuras que sugieren la posibilidad de una refactorización.

59. En la refactorización se permite que cambie la estructura interna de un sistema software aunque varíe su comportamiento externo.

60. Un ejemplo de refactorización sería mover un método arriba o abajo en la jerarquía de herencia.

61. El cambio de una sentencia condicional por el uso de polimorfismo es un ejemplo de refactorización.

62. Hacer el código más fácil de entender no es un motivo suficiente para refactorizarlo.

63. Existe un catálogo de refactorizaciones comunes, de forma que el programador no se ve obligado a usar su propio criterio y metodología para refactorizar el código.

64. El principio de segregación de interfaz indica que el código cliente no debe ser forzado a depender de interfaces que no utiliza.

65. Con el uso de reflexión solo podemos acceder a métodos de instancia.

FRAMEWORKS.

66. Los frameworks no contienen implementación alguna, únicamente un conjunto de interfaces que deben ser implementados por el usuario del framework.

67. Un framework invoca mediante enlace dinámico a nuestra implementación de interfaces propios de framework.

68. Hibernate es un framework para congelar en memoria el estado de nuestra aplicación.

69. JDBC es un framework de Java que usan los fabricantes de sistemas de gestión de bases de datos para ofrecer un acceso estandarizado a las bases de datos.

70. El usuario de un framework implementa al componente declarado de los interfaces de framework mediante herencia de implementación.

71. Un frameworks es un conjunto de clases cuyos métodos invocamos para que realicen unas tareas a modo de caja negra.

72. En Java el acceso a bases de datos se hace con librerías propietarias de SGBD cuyos interfaces no tienen ningún estándar.

73. El usuario de un framework implementa el comportamiento declarado en los interfaces del framework mediante herencia de interfaz.

74. Para poder utilizar un framework, es necesario crear clases que implementen todas las interfaces declaradas en el framework.

OTROS.

75. Una librería de clases proporciona una funcionalidad completa, es decir, no requiere que el usuario implemente o herede nada.

76. El polimorfismo es una forma de reflexión.

77. En el proceso de diseño de un sistema de software se debería intentar aumentar el acoplamiento y la cohesión.

78. Cuando diseñamos sistemas orientados a objetos las interfaces de las clases que diseñamos deberían estar abiertas a la extensión y cerradas a la modificación.

79. Todo espacio de nombre define su propio ámbito, distinto de cualquier otro.

80. Una colaboración describe como un grupo de objetos trabaja conjuntamente para realizar una tarea.

81. En el diseño mediante tarjetas CRC utilizamos una tarjeta para cada clase.

82. Una tarjeta CRC contiene el nombre de una clase, su lista de responsabilidades y su lista de colaboradores.

83. La robustez de un sistema software es un parámetro de calidad intrínseco.

84. El principio abierto-cerrado indica que un componente software debe estar abierto a su extensión y cerrado a su modificación.

85. Con el diseño orientado a objetos es perfectamente posible y deseable hacer uso de variables globales.

86. En el diseño por contrato son dos componentes fundamentales las pre y pos condiciones.