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

11. En Java los métodos de instancia con polimorfismo puro pero no abstracto tienen enlace dinámico. V

12. Una operación de clase solo puede acceder directamente a atributos de clase. V

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

15. En la misma clase podemos definir constructores de con distinta visibilidad. V

19. La genericidad es un tipo de polimorfismo. V

24. El downcasting implica deshacer el principio de sustitución. V

26. La instrucción throw en JAVA solo permite lanzar objetos que son de la clase throwable o clases derivada de esta. V

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

29. En JAVA. Siempre es obligatorio especificar que excepciones verificadas (checked exceptions) lanza un método mediante una clausula throws tras la lista de argumentos. V

32. Todas las excepciones son checked exception salvo las runtime que son uncheked exception. V

35. Podemos poner un bloque finally sin poner bloques catch. V

38. La genericidad se considera una característica opcional de los lenguajes. V

39. La genericidad se considera una característica opcional de los lenguajes OO. V

42. En los métodos genéricos solo podremos usar los métodos definidos en Object. V

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

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

47. En JAVA el concepto de meta-clase se presenta con la clase Class. V

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

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

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

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

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

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

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

64. 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. V

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

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

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

73. Cuando diseñamos sistema OO las interfaces de las clases que diseñamos deberían estar abiertas a la extensión y cerradas a la modificación. V

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

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

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

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

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

81. En el diseño por contrato son dos componentes fundamentes las pre y pos condiciones. V

82. Los métodos definidos en una clase derivada nunca pueden acceder a las propiedades privadas de una clase base. V

87. Los métodos abstractos son métodos con enlace dinámico. V

92. En C++ no podemos hacer sobrecarga de operadores para tipos predefinidos. V

94. En la misma clase, podemos definir constructores con distinta visibilidad. V

96. La instrucción throw permite lanzar como excepción cualquier tipo de dato. V

98. Dada una clase genérica, se pueden derivar de ella clases no genéricas. V

100. Una clase interfaz no puede tener atributos de instancia. Una clase abstracta sí puede tenerlos. V

101. La herencia de interfaz se implementa mediante herencia pública. V

103. Una clase abstracta se caracteriza por declarar al menos un método abstracto. V

105. Los métodos abstractos siempre tienen enlace dinámico. V

107. Un método tiene polimorfismo puro cuando tiene como argumentos al menos una variable polimórfica. V

109. Una de las características básicas de una lengua orientada a objetos es que todos los objetos de la misma clase pueden recibir los mismos mensajes. V

111. En C++, si no se define ningún constructor, el compilador proporciona por defecto uno sin argumentos. V

112. Una clase interfaz no puede tener instancias. V

114. No se puede definir un bloque catch sin su correspondiente bloque try. V

116. Una variable polimórfica puede hacer referencia a diferentes tipos de objetos en diferentes instantes de tiempo. V

118. El principio de sustitución implica una coerción entre tipos de una misma jerarquía de clases. V

119. La sobrecarga basada en ámbito permite definir el mismo método en dos clases diferentes. V

120. Una operación de clase no puede tener enlace dinámico. V

122. La siguiente clase en C++: class S {public: virtual ~S()=0;}; define una interfaz. V

125. En C++, un atributo de la clase debe declararse dentro de la clase con el modificador static. V

128. Una de las características básicas de un lenguaje orientado a objetos es que todos los objetos de la misma clase pueden recibir los mismos mensajes. V

129. Hablamos de encapsulación cuando agrupamos datos junto con las operaciones que pueden realizarse sobre esos datos. V

132. Una clase interfaz no debe tener atributos de instancia. Una clase abstracta sí puede tenerlos. V

134. La interpretación de un mismo mensaje puede variar en función del receptor del mismo y/o del tipo de información adicional que lo acompaña. V

136. Una forma de mejorar el diseño de un sistema es reducir su acoplamiento y aumentar su cohesión. V

141. Cuando se crea un objeto de la clase D de que deriva de una clase B, el orden de ejecución de los constructores es siempre B() D(). V

142. Una clase abstracta es una clase que no permite instancias de ella. V

144. Un método de clase (estático) no puede tener enlace dinámico. V

147. Los métodos definidos en una clase genérica son a su vez genéricos. V

152. Los TAD’s representan una perspectiva orientada a datos, mientas que los objetos reflejan una perspectiva orientada a servicios. V

153. Un atributo privado en la clase base no es directamente accesible en la clase derivada, independientemente del tipo de herencia utilizado. V

157. El puntero this es un puntero constante al objeto que recibe el mensaje. V

160. La forma canónica de la clase está formada por el constructor, el constructor de copia, el destructor y el operador de asignación. V

162. Tanto composición como herencia son mecanismos de reutilización del software. V

164. Un atributo de clase público puede ser accedido desde fuera de la clase a través de un objeto de la clase, un puntero o referencia al mismo o mediante el nombre de la clase seguido del operador de ámbito. V

166. Una interfaz es la definición de un protocolo para cierto comportamiento, sin especificar la implementación de dicho comportamiento. V

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

168. En el diseño mediante tarjetas CRC, utilizamos una tarjeta por cada clase. V

169. La sobreescritura es una forma de polimorfismo. V

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

177. En el paradigma orientado a objetos, un objeto siempre es instancia de alguna clase. V

179. En el paradigma orientado a objetos, un programa es un conjunto de objetos que se comunican mediante el paso de mensajes. V

180. El siguiente código en Java define una interfaz: interface S {} V

181. Sea un método llamado glue(), sin argumentos, implementado en una superclase y sobrescrito en una de sus subclases. Siempre podremos invocar a la implementación del método en la superclase desde la implementación del método en la subclase usando la instrucción super.glue(); V

182. Los métodos abstractos siempre tienen enlace dinámico. V

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

186. Las sentencias ‘switch’ son un caso de código sospechoso (código con mal olor). V

187. El código duplicado es un caso de código sospechoso en el que se aconseja el uso de técnicas de refactorización para eliminarlo. V

188. La existencia de una sólida colección de pruebas unitarias es una precondición fundamental para la refactorización. V

190. El enlace de la invocación a un método sobrescrito se produce en tiempo de ejecución en función del tipo del receptor del mensaje. V

191. this es un ejemplo de variable polimórfica en Java. V

192. En Java el downcasting siempre se realiza en tiempo de ejecución. V

193. En Java, un atributo de clase debe declararse dentro de la clase con el modificador static. V

195. Si en una clase no se declara, implícita o explícitamente, un constructor por defecto, no se pueden crear instancias de esa clase. V

196. Una de las características básicas de unos lenguajes orientados a objetos es que todos los objetos de la misma clase pueden recibir los mismos mensajes. V

197. La instrucción throw en Java sólo permite lanzar objetos que son instancias de la clase java.lang.Throwable o de clases derivadas de ésta. V

209. Un objeto se caracteriza por poseer un estado, un comportamiento y una identidad. V

213. La siguiente clase: class S {public: virtual ~S()=0; virtual void f()=0;}; constituye una interfaz en C++. V

218. Un método sobrecargado es aquel que tiene más de una implementación, diferenciando cada una por el ámbito en el que se declara, o por el número, orden y tipo de argumentos que admite. V

220. Todo espacio de nombres define su propio ámbito, distinto de cualquier otro ámbito. V

221. En la sobrecarga de operadores binarios para objetos de una determinada clase, si se sobrecarga como función miembro, el operando de la izquierda es siempre un objeto de la clase. V

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

223. Hablamos de encapsulación cuando diferenciamos entre interfaz e implementación. V

227. En C++, la cláusula throw() tras la declaración de una función indica q ésta no lanza ninguna excepción. V

230. Cuando usamos la varianza estamos haciendo un uso inseguro de la herencia de implementación. V

232. Cuando creamos un objeto en C++ mediante una variable automática el constructor se autoinvoca. También se autoinvoca el destructor del mismo al salir del ámbito de la función donde se creó. V

234. De una clase abstracta se pueden crear referencias a objetos. V

243. Si utilizamos los mecanismos de manejo de excepciones disminuye la eficiencia del programa incluso si no se llega a lanzar nunca una excepción. V

245. Hablamos de shadowing cuando el método a invocar se decide en tiempo de compilación. V

246. A diferencia de otros lenguajes de programación en C++ la sobreesctritura en relaciones de herencia se debe indiciar de forma explícita en la clase padre. V

245. La encapsulación es un mecanismo que permite separar de forma estricta interfaz e implementación. V

246. La interpretación de un mismo mensaje puede variar en función del receptor del mismo y/o del tipo de información adicional que lo acompaña. V

247. Un atributo de clase público puede ser accedido desde fuera de la clase a través de un objeto de la clase, un puntero o referencia al mismo o mediante el nombre de la clase seguido del operador de ámbito. V

248. Es posibles definir un constructor de copia invocando en su cuerpo al operador de asignación. V

251. La signatura de tipo de un método incluye el tipo devuelto por el método. V

252. En el principio de sustitución implica una coerción entre tipos de una misma jerarquía de clases. V

1. La herencia de interfaz se implementa mediante herencia pública. V

2. Una interfaz no puede tener atributos de instancia. Una clase abstracta si puede tenerlos. V

3. No se puede definir un bloque catch sin su correspondiente bloque try. V

4. Una variable polimórfica puede hacer referencia a diferentes tipos de objetos en diferentes instantes de tiempo. V

6. La sobrecarga basada en ámbito permite definir el mismo método en dos clases diferentes. V

8. Un espacio de nombres es un ámbito con nombre. V

9. Un sistema de tipos de un lenguaje asocia a cada tipo una expresión. V

12.En Java, una clase genérica puede ser parametrizada empleado más de un tipo. V

14.Una de las principales fuentes de problemas cuando utilizamos herencia múltiple es que las clases bases hereden de un ancestro común. V

18. En la sobrecarga basada en ámbito los métodos pueden diferir únicamente en el tipo devuelto. V

21.El cambio de una condicional por el uso de polimorfismo es un ejemplo de refactorización. V

23.El principio de segregación de interfaz indica que el código cliente no debe ser forzado a depender de interfaces que no utilice. V