INDICE

ABSTRACCION 2
ENCAPSULACIÓN 4
INTERFAZ E IMPLEMENTACION 5
CONSTRUCTOR 6
OPERACIONES (METODOS) 7
FORMA CANONICA 8
HERENCIA 9
POLIMORFISMO 10
GENERICIDAD 11
VISIBILIDAD 13
AGREGACION Y COMPOSICION 14
FRAMEWORKS Y LIBRERIAS 15
REFLEXION 17
GESTION DE ERRORES 18
REFACTORIZACION 21
SOBRECARGA 22
ATRIBUTO 23
CLASE 24
METODOS 25
DOWNCASTING 26
OTRAS 27
MAS 29

ABSTRACCION

Expresa las características esenciales de un objeto, las cuales distinguen al objeto de los demás. Una clase que hereda de una clase abstracta puede ser no abstracta.		
1	De una clase abstracta no se pueden crear instancias, excepto si se declara explícitamente algún constructor	F
2	De una clase abstracta se pueden crear referencias a objetos de la clase	V
3	Destacamos 3 Niveles de Abstracción	V
4	En Java los métodos abstractos pueden tener implementación pero siempre harán abstracta a la clase que contenga	F
5	En Java, si un método no abstracto f() definido en una superclase se sobrescribe en una de sus subclases, se puede invocar el método de la superclase desde la subclase mediante la instrucción super.f()	V
6	La abstracción es una supresión intencionada (u ocultación) de algunos detalles de un proceso o artefacto, con el fin de destacar más claramente otros aspectos, detalles o estructuras	V
7	La clase abstracta se caracteriza por declarar al menos un método abstracto	V
8	Los constructores de las clases abstractas siempre son métodos abstractos	F
9	Los constructores de las clases abstractas son métodos con enlace dinámico	F
10	Los métodos abstractos siempre tienen enlace dinámico	V
11	Los métodos abstractos son métodos con enlace dinámico	V
12	Los métodos virtuales son métodos abstractos	F
13	Ser abstracto implica tener enlace dinámico	V
14	Un método abstracto es un método con polimorfismo puro	F
15	Un método con enlace dinámico siempre es abstracto	F
16	Una clase abstracta es una clase que no permite instancias de ella	V
17	Una clase abstracta no puede tener constructores	F
18	Una clase abstracta se caracteriza por declarar al menos un métodos abstracto	V
19	Una clase abstracta se caracteriza por declarar todos sus métodos de instancia como abstractos	F
20	Una clase abstracta se caracteriza por no tener atributos	F
21	Una clase abstracta se caracteriza por no tener definido ningún constructor	F
22	Una clase abstracta siempre tiene como clase base una clase interfaz	F
23	Una clase abstracta siempre tiene que tener alguna clase que derive de ella	F

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

ENCAPSULACIÓN

Ocultación del estado o de los datos miembro de un objeto, de forma que sólo es posible modificar los mismos mediante los métodos definidos para dicho objeto. Garantiza la integridad de los datos que contiene un objeto. Existen tres niveles de acceso para elementos de esa clase: *público* (las funciones de todas las clases pueden acceder a los datos o métodos de una clase pública), *protegido* (acceso a datos restringido a las funciones de clases heredadas), *privado* (acceso a datos restringido a métodos de esa clase en particular).		
1	Hablamos de encapsulación cuando agrupamos datos junto con las operaciones que pueden realizarse sobre esos datos	V
2	Hablamos de encapsulación cuando diferenciamos entre interfaz e implementación	V
3	La encapsulación es un mecanismo que permite separar de forma estricta interfaz e implementación	V

INTERFAZ E IMPLEMENTACION

Una interfaz es una lista de acciones que puede llevar a cabo un determinado objeto.

1	ArrayList es una implementación en el Java Collection Framework de la interfaz List	V
2	De una clase interfaz no se pueden crear instancias. De una clase abstracta si	F
3	El principio de segregación de interfaz indica que el código cliente no debe ser forzado a depender de interfaces que no utiliza	V
4	El principio de segregación de interfaz indica que el código cliente no debe ser forzado a depender de interfaces que no utiliza	V
5	El siguiente código en Java define una interfaz: interface S{}	V
6	El usuario de un framework implementa el comportamiento declarado en los interfaces del framework mediante herencia de interfaz	V
7	La clase class S{ public Object obj; } constituye una interfaz en Java	F
8	La herencia de interfaz se implementa mediante herencia pública	V
9	La siguiente sentencia class S{public final void f() {}} constituye una interfaz en JAVA	F
10	Mediante la herencia de implementación heredamos la implementación de los métodos de la clase base pero no la interfaz de esta	F
11	Una clase abstracta siempre tiene como clase base una clase abstracta	F
12	Una clase interfaz no debe tener atributos de instancia. Una clase abstracta si puede tenerlos	V
13	Una clase interfaz no puede tener atributos de instancia. Una clase abstracta si puede tenerlos	V
14	Una clase interfaz no puede tener instancias	V
15	Una interfaz en Java obliga a que las clases no abstractas que la implementan definan todos los métodos que la interfaz declara	V
16	Una interfaz es la definición de un protocolo para cierto comportamiento, sin especificar la implementación de dicho comportamiento	V
17	Una interfaz es una clase abstracta con al menos un método de instancia abstracto	F
18	Una interfaz no puede tener atributos de instancia. Una clase abstracta si puede tenerlos	V
19	Una interfaz puede implementar otra interfaz	F

CONSTRUCTOR

Un Constructor es una función, método, etc. de las clases, la cual es llamada automáticamente cuando se crea un objeto de esa clase. No puede ser Heredado.		
1	Cuando se crea un objeto de la clase D que deriva de una clase B, el orden de ejecución de los constructores es siempre B() D().	V
2	El constructor de copia permite argumentos tanto por referencia como por valor	F
3	El objetivo de un constructor es crear e inicializar objetos	V
4	En cuanto a los constructores son funciones miembro constantes	V
5	En la misma clase podemos definir constructores con distinta visibilidad	V
6	Es conveniente definir siempre un Constructor por Defecto que permita la inicialización sin parámetros de un objeto, donde los atributos se inicializan con valores por defecto	V
7	Es posibles definir un constructor de copia invocando en su cuerpo al operador de asignación	V
8	Los constructores siempre deben tener visibilidad publica	F
9	Los constructores siempre son métodos virtuales	F
10	Los constructores siempre tienen enlace dinámico	F
11	Si en una clase no se declara, implícita o explícitamente, un constructor por defecto, no se pueden crear instancias de esa clase	V
12	Un constructor de copia acepta cualquier tipo de modificador (static, const, public, etc).	F

OPERACIONES (METODOS)

Función miembro, método o servicio de la clase. Acción que puede realizar un objeto en respuesta a un mensaje. Definen el comportamiento del objeto. Tienen asociada una visibilidad, pueden ser de clase o de instancia. Pueden modificar el estado del sistema.		
1	En el cuerpo de una operación de clase no se puede acceder a ningún atributo/operación del objeto receptor del mensaje	V
2	En el cuerpo de una operación de clase no se puede acceder a ningún atributo/operación de instancia de los objetos pasados como parámetros	F
3	En el cuerpo de una operación de clase se puede acceder a únicamente a atributo/operación de clase	V
4	En el cuerpo de una operación de clase se puede acceder únicamente a atributos/operaciones de clase	V
5	Las operaciones de clase no son funciones miembro de la clase	F
6	Las operaciones de clase solo pueden manejar atributos de clase	F
7	Las operaciones de clase solo pueden ser invocadas por objetos constantes	F
8	Una operación de clase no es una función miembro de la clase	F
9	Una operación de clase no puede tener enlace dinámico	V
10	Una operación de clase solo puede acceder directamente a atributos de clase	V
11	Una operación de clase sólo puede ser invocada mediante objetos constantes	F
12	Una operación de instancia puede acceder directamente a atributos de clase y de instancia	V

FORMA CANONICA

Conjunto de métodos que toda clase debería definir independientemente de su funcionalidad.		
1	En JAVA la forma canónica incluye constructor, equials, hashCode, toString y clone	F
2	Implementar la forma canónica ortodoxa de una clase es una condición necesaria (aunque no suficiente) para controlar que el valor de un atributo de clase que cuenta el número de instancias de dicha clase esté siempre en un estado consistente	F
3	La forma canónica de la clase está formada por el constructor, el constructor copia, el destructor y el operador de asignación	V

HERENCIA

Mecanismo que permite la definición de una clase a partir de la definición de otra ya existente. Permite compartir automáticamente métodos y datos entre clases, subclases y objetos.		
1	Dado una clase genérica, no puede ser utilizada como clase base en herencia múltiple	F
2	En la herencia pública la clase derivada podrá acceder a los atributos privados de la clase base de la que hereda	F
3	Independientemente del tipo de herencia la clase base siempre podrá acceder a lo público, protegido y default heredado pero no a lo privado	F
4	La herencia de implementación siempre implica herencia de interfaz	F
5	La herencia de interfaz se implementa mediante herencia pública	V
6	La herencia es más flexible en cuanto a posibles cambios en la naturaleza de los objetos que la composición	F
7	La herencia múltiple se produce cuando de una misma clase base se heredan varias clases derivadas	F
8	La herencia protegida permite a los métodos de la clase derivada acceder a las propiedades privadas de la clase base	F
9	La herencia pública implica herencia de implementación y de interfaz mientras que la herencia privada o protegida implica herencia de implementación pero no de interfaz	V
10	La herencia pública permite a los métodos definidos en una clase derivada acceder a las propiedades privadas de la clase base	F
11	La relación de herencia es una relación de clases no persistentes	F
12	Mediante la herencia de implementación heredamos la implementación de los métodos de la clase base pero no la interfaz de esta	F
13	Sean dos clases Base e Hija. La clase Hija hereda de Base. En JAVA, cuando asignamos un objeto de la clase Hija a una referencia a Base haciendo conversión de tipo explícita estamos haciendo object slicing	F
14	Si la conversión, downcasting, es fuera de la jerarquía de herencia JAVA dará error de ejecución	F
15	Tanto composición como herencia son mecanismos de reutilización del software	V
16	Tanto la herencia protegida como la privada permiten a una clase derivada acceder a las propiedades privadas de la clase base	F
17	Tanto la herencia protegida como la privada permiten a una clase derivada acceder a las propiedades privadas de la clase base	F
18	Una de las principales fuentes de problemas cuando utilizamos herencia múltiple es que las clases bases hereden de un ancestro común	V

POLIMORFISMO

Habilidad de una función, método, variable u objeto de poseer varias formas distintas. Un mismo identificador comparte varios significados diferentes. Su propósito es implementar un estilo de programación llamado envío de mensajes en el que los objetos interactúan entre ellos mediante estos mensajes, que son más que llamadas a distintas funciones. Las 4 formas principales de polimorfismo son: P. de asignación (el más relacionado con el enlace dinámico), P. puro, Sobrecarga, P. de inclusión.

1	El cambio de una condicional por el uso de polimorfismo es un ejemplo de refactorización	V
2	El cambio de una sentencia condicional por el uso de polimorfismo es un ejemplo de refactorización	V
3	El polimorfismo debido a la sobrecarga de funciones siempre se da en relaciones de herencia	F
4	El polimorfismo es un tipo de genericidad	F
5	El polimorfismo es una forma de reflexión	F
6	El puntero this no es una variable polimórfica porque es constante y no se puede cambiar su valor	F
7	En Java los métodos de instancia con polimorfismo puro pero no abstractos tienen enlace dinámico	V
8	En JAVA los métodos de instancia con polimorfismo puro pero no abstracto tienen enlace dinámico	V
9	En JAVA los métodos de instancia con polimorfismo puro pero no abstractos tienen enlace dinámico por defecto	V
10	La genericidad es un tipo de polimorfismo	V
11	La sobrescritura es una forma de polimorfismo	V
12	this es un ejemplo de variable polimórfica en JAVA	V
13	Un método abstracto es un método con polimorfismo puro	F
14	Un método tiene polimorfismo puro cuando devuelve una variable polimórfica	F
15	Un método tiene polimorfismo puro cuanto tiene como argumentos al menos una variable polimórfica	V
16	Una variable polimórfica puede hacer referencia a diferentes tipos de objetos en diferentes instantes de tiempo	V

GENERICIDAD

Clases parametrizadas con tipos de datos. Permite escribir código reutilizable.

1	Dada una clase genérica, no puede ser utilizada como clase base en herencia múltiple	F
2	Dada una clase genérica, no se pueden derivar de ella clases genéricas	F
3	Dada una clase genérica, se pueden derivar de ella clases no genéricas	V
4	El polimorfismo es un tipo de genericidad	F
5	En JAVA no podemos crear interfaces genéricas	F
6	En Java no se pueden derivar clases genéricas de otras genéricas	F
7	En Java, los tipos genéricos sólo se pueden aplicar a clases e interfaces	F
8	En Java, una clase genérica puede ser parametrizada empleado más de un tipo	V
9	En la genericidad restringida sólo podemos usar los métodos definidos en Object	F
10	En la genericidad restringida solo podremos usar los métodos Object al implementar los métodos	F
11	En los métodos genéricos sólo podemos usar los métodos definidos en Object	V
12	Hay dos tipos de genéricos: Clases Genéricas y Métodos Genéricos	V
13	La genericidad es un tipo de polimorfismo	V
14	La genericidad es una propiedad que permite definir a una clase o una función sin tener que especificar el tipo de datos o algunos de sus miembros o argumentos.	V
15	La genericidad restringida permite indicar que los tipos genéricos pertenezcan a una determinada jerarquía de herencia	V
16	La genericidad se considera una característica opcional de los lenguajes orientados a objetos	V
17	La genericidad se considera una característica opcional de los lenguajes	V
18	La genericidad se considera una característica opcional de los lenguajes orientados a objetos	V
19	La genericidad se considera una característica opcional de los lenguajes	V
20	Las funciones genéricas no se pueden sobrecargar	F
21	Los métodos definidos en una clase genérica son a su vez genéricos	V
22	Los métodos genéricos no se pueden sobrecargar ni sobrescribir	F
23	No se puede derivar de una clase no genérica a una genérica	F

24	No se puede derivar una clase genérica de otra no genérica	F
25	Se pueden derivar clases genéricas y no genéricas de otras genéricas	V
26	Si para una clase genérica llamada Pila<T> declaramos las siguientes funciones: virtual void apilar(T* pt); virtual void apilar(T t); En caso de sobrescritura ya que el tipo devuelto es el mismo en ambos métodos	F

VISIBILIDAD

La visibilidad puede ser:
Pública: se puede acceder al miembro de la clase desde cualquier lugar
JAVA: public. UML: +
Protegida: sólo se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella
JAVA: protected. UML: #
Privada: sólo se puede acceder al miembro de la clase desde la propia clase.
JAVA: private. UML: -

1	En la misma clase, podemos definir constructores con distinta visibilidad	V
2	Los constructores siempre deben tener visibilidad pública	F
3	Un atributo de clase debe tener visibilidad pública para poder ser accedido por los objetos de la clase	F
4	Un atributo declarado con visibilidad protegida en una clase A es accesible desde clases definidas en el mismo espacio de nombres donde se definió A	F

AGREGACION Y COMPOSICION

La agregación es un tipo de asociación que indica que una clase es parte de otra clase. Los componentes pueden ser compartidos.
La composición es una forma fuerte de composición donde la vida de la clase contenida debe coincidir con la vida de la clase contenedor.

1	En una agregación, la existencia de un objeto parte depende de la existencia del objeto todo que lo contiene	F
2	La agregación es una relación mucho más restrictiva que la herencia	V
3	La agregación y la asociación pueden ser bidireccionales	F
4	Las relaciones todo-parte (tiene-un) son la agregación y la composición	V
5	Ni la asociación ni la agregación manejan la creación/destrucción de objetos tipo parte	V
6	En una composición de un objeto componente puede formar parte de más de un compuesto	F
7	En una composición un objeto componente puede formar parte de más de un objeto compuesto	F
8	La Composición maneja la creación/destrucción de los objetos tipo parte	V
9	La herencia es más flexible en cuanto a posibles cambios en la naturaleza de los objetos que la composición	F
10	Tanto composición como herencia son mecanismos de reutilización del software	V

FRAMEWORKS Y LIBRERIAS

Un framework es un molde sobre el cual se puede construir algo
Las librerías son importadas ya que son necesarias para usar funciones en el programa

1	Arraylist es una implementación en el Java Collection Framework de la interfaz List	V
2	El usuario de un framework implementa al componente declarado de los interfaces de framework mediante herencia de interfaz	V
3	El usuario de un framework implementa al componente declarado de los interfaces de framework mediante herencia de implementación	F
4	El usuario de un framework implementa el comportamiento	F
5	El usuario de un framework implementa el comportamiento declarado en los interfaces del framework mediante herencia de interfaz	V
6	El usuario de un framework implementa el comportamiento declarado en los interfaces del framework mediante herencia de implementación	F
7	En Java el acceso a bases de datos se hace con librerías propietarias de SGBD cuyos interfaces no tienen ningún estándar	F
8	Hibernate es un framework para congelar en memoria el estado de nuestra aplicación	F
9	JCF (Java Collection Framework) tiene un conjunto de clases en el JDK representando tipos de datos abstractos	V
10	JDBC (Java Data Base Collection) es un framework que permite conectar Java con Bases de Datos	V
11	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
12	La inversión de control en los frameworks es posible gracias al enlace dinámico de métodos	V
13	Los Frameworks no contienen implementación alguna, únicamente un conjunto de interfaces que debes ser implementados por el usuario del framework	F
14	Para poder utilizar un framework, es necesario crear clases que implementen todas las interfaces declaradas en el framework	V
15	Un framework es un conjunto de clases cuyos métodos invocamos para que realicen varias tareas a modo de caja negra	F
16	Un Framework es un esqueleto para fines específicos que debemos completar y personalizar mediante la implementación de interfaces, herencia de clases abstractas y ficheros de configuración	V
17	Un framework invoca mediante enlace dinámico a nuestra implementación de interfaces propios de framework	V
18	Un frameworks es un conjunto de clases cuyos métodos invocamos para que realicen unas tareas a modo de caja negra	F

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

REFLEXION

El programa tiene la capacidad de observar y modificar su estructura de manera dinámica		
1	Con el uso de la reflexión solo podemos invocar métodos de instancia	F
2	Con el uso de reflexión solo podemos acceder a métodos de instancia.	F
3	Dado un objeto, mediante reflexión no se puede obtener la lista de todos los métodos declarados en su clase, tanto públicos como privados	F
4	El polimorfismo es una forma de reflexión	F
5	La API de reflexión de JAVA incluye métodos para obtener la signatura de todos los métodos	V
6	La instanciación mediante reflexión de un objeto de la clase Rectángulo del paquete pack se realiza así: Class.forName(“pack”, “Rectangulo”);	F
7	La reflexión es demasiado compleja para usarla en aplicaciones de propósito general	F
8	La Reflexión es una infraestructura del lenguaje que permite a un programa conocerse y manipularse a sí mismo en tiempo de ejecución	V
9	La reflexión no puede usarse en aplicaciones certificadas con el estándar 100% Pure Java.	F
10	La reflexión permite que un programa obtenga información sobre sí mismo en tiempo de ejecución	V
11	La Reflexión puede usarse para construir nuevos objetos y arrays, acceder y modificar atributos de instancia, invocar métodos de instancia y estáticos	V
12	La reflexión reduce el rendimiento de las aplicaciones	F
13	La reflexión sólo es útil para trabajar con componentes JavaBeans	F
14	Mediante reflexión no podemos saber cuál es el método que se está ejecutando en un determinado momento	V
15	Mediante reflexión podemos saber cuáles son las clases derivadas de una clase dada.	V
16	Para usar reflexión en Java hemos de conocer el nombre de las clases en tiempo de compilación	F
17	Podemos usar reflexión para encontrar un método heredado (solo hacia arriba) y reducir código condicional	V

GESTION DE ERRORES

Durante el desarrollo pueden darse dos tipos de errores: los de tiempo de compilación y los de tiempo de ejecución. En general es preferible que los lenguajes de compilación estén diseñados de tal manera que la compilación pueda detectar el máximo número posible de errores. Es preferible que los errores de tiempo de tiempo de ejecución se deban a situaciones inesperadas y no a descuidos del programador. Errores de tiempo de ejecución siempre habrá, y su gestión a través de excepciones es fundamental en cualquier lenguaje de programación actual

1	Cuando se captura una excepción y ésta pertenece a una jerarquía de clases, el primer bloque catch debe comenzar con la clase del nivel más alto de la jerarquía	F
2	Desde un método de una clase derivada nunca puede invocarse un método implementado con idéntica signatura de una de sus clase base	F
3	El bloque finally solo se ejecutará si se produce alguna excepción en el bloque try al que esté asociado	F
4	El estado de un objeto es el conjunto de valores de los atributos y métodos que han sido invocados sobre él	F
5	El orden de las excepciones en los bloques catch no es relevante	F
6	En JAVA si no se captura una excepción lanzada por un método da error la compilación	F
7	En JAVA, la instrucción throw no se puede usar dentro de un bloque catch	F
8	En JAVA, siempre es obligatorio especificar que excepciones verificadas (checked exceptions) lanza un método mediante una cláusula throws tras la lista de argumentos	V
9	Es una forma sistemática de introducir mejoras en el código que minimiza la posibilidad de introducir errores en él.	V
10	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.	F
11	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.	F
12	La cláusula throws de un método incluirá todas las excepciones unchequed que puedan producirse en esta y estén dentro del bloque try catch que las capture	F
13	La instrucción throw en JAVA solo permite lanzar objetos que son de la clase throwable o clases derivadas de esta.	V
14	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
15	La instrucción throw permite lanzar como excepción cualquier tipo de dato.	V
16	Las excepciones si se ignoran, se produce un error de compilación	F

17	Las excepciones si se ignoran, el compilador genera mensajes de advertencia (warnings)	F
18	Las excepciones si se ignoran, el programa no advierte que ha ocurrido algún error y continua su ejecución normalmente	F
19	Las instrucciones para el manejo de excepciones nos permiten mezclar el código que describe el funcionamiento normal de un programa con el código encargado del tratamiento de errores	F
20	Los bloques try-catch no se pueden anidar	F
21	Los constructores de las clases abstractas son siempre métodos abstractos	F
22	Los métodos con enlace dinámico son abstractos	F
23	No se puede definir un bloque catch sin su correspondiente bloque try	V
24	Podemos poner en bloque finally sin poner bloques catch	V
25	Si en el bloque try de un catch, que captura chequed exception, es imposible que se produzca dicha excepción el código no compilará, indicando que el bloque catch es inalcanzable.	V
26	Si en la sentencia thorws de un método especificamos una excepción que es imposible que se lance en el cuerpo de dicho método el compilador no compilará, indicando que el método nunca lanzará la excepción.	F
27	Si la conversión, downcasting, es fuera de la jerarquía de herencia JAVA dará error de ejecución	F
28	Si no se captura una excepción lanzada por un método, el programa no advierte que ha ocurrido error y continúa su ejecución normalmente	F
29	Si no se captura una excepción por un método, el programa no advierte que ha ocurrido error y continúa su ejecución que llamo a este	F
30	Si se produce una excepción el método que la provoca se cancela y se continúa la ejecución en el método que llamo a este.	F
31	Si se produce una excepción en un constructor el objeto se construirá con los valores por defecto.	F
32	Todas las excepciones son checked exception salvo las runtime que son uncheked exception	V
33	Todo espacio de nombres define su propio ámbito, distinto de cualquier otro ámbito	V
34	Tras la ejecución de un bloque catch, termina la ejecución del programa.	F
35	Un método sobrecargado es aquel que tiene más de una implementación, diferenciando cada uno por el ámbito en el que se declara, o por el número, orden y tipo de argumentos que admite	V

36	Un objeto se caracteriza por poseer un estado, un comportamiento y una identidad	V
37	Uno de los objetivos del tratamiento de errores mediante excepciones es el manejo de errores del resto del código	V

REFACTORIZACION

	La refactorización consiste en tomar una pieza de código y modificarla (re-factorizarla) de tal forma que haga exactamente lo mismo, pero su diseño mejore	
1	El cambio de una sentencia condicional por el uso de polimorfismo es un ejemplo de refactorización.	V
2	El código duplicado es un caso de código sospechoso en el que se aconseja el uso de técnicas de refactorización para eliminarlo.	V
3	El principio de segregación de interfaz indica que el código cliente no debe ser forzado a depender de interfaces que no utiliza	V
4	En la refactorización se permite que cambie la estructura interna de un sistema software aunque varíe su comportamiento externo.	F
5	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	V
6	Hacer el código más fácil de entender no es un motivo suficiente para refactorizarlo	F
7	Hacer que el código sea más fácil de entender no es un motivo suficiente para refactorizarlo.	F
8	La existencia de una sólida colección de pruebas unitarias es una precondición fundamental para la refactorización	V
9	La refactorización debe hacerse siempre apoyándonos en un conjunto de tests completo y robusto	V
10	La refactorización ha sido identificada como una de las importantes innovaciones en el campo del software	V
11	La refactorización nunca produce cambios en las interfaces de las clases	F
12	Los métodos grandes (con muchas instrucciones) son estructuras que sugieren la posibilidad de una refactorización.	V
13	Refactorizar es una forma sistemática y segura de realizar cambios en una aplicación con el fin de hacerla más fácil de comprender	V
14	Un ejemplo de refactorización sería mover un método arriba o abajo en la jerarquía de herencia	V
15	Una clase con un gran número de métodos y atributos es candidata a ser refactorizada	V

SOBRECARGA

Varios métodos con el mismo nombre pero con diferente lista de tipos de parámetros		
1	En JAVA, gracias a la sobrecarga de operadores podemos crear nuevos operadores en el lenguaje	F
2	En la sobrecarga basada en ámbito los métodos pueden diferir únicamente en el tipo devuelto	V
3	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
4	En la sobrecarga de operadores como función miembro, el operando de la izquierda puede ser un objeto de la clase o cualquier otro tipo de objeto, mientras que en las funciones amigas siempre es un objeto de la clase	F
5	La sobrecarga basada en ámbito permite definir el mismo método en dos clases diferentes	V
6	Un método sobrecargado es aquel que recibe como argumento al menos una variable polimórfica	F

ATRIBUTO

1	A los atributos de instancia si son constantes se les asigna su valor inicial fuera de su clase	F
2	En JAVA, un atributo de clase debe declararse dentro de la clase con el modificador static	V
3	En un atributo de clase se reserva espacio en memoria para una copia de él por cada objeto de su clase creado	F
4	Un atributo de clase debe declararse dentro de la clase con el modificador const	F
5	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
6	Un atributo estático ocupa una zona de memoria que es compartida por todos los objetos de la clase en la que se define, aunque no por los objetos de cualquier clase derivada de ella	F
7	Un atributo privado en la clase base no es directamente accesible en la clase derivada, independientemente del tipo de herencia utilizado	V
8	Un atributo protegido es la clase base es también protegido en cualquier clase que derive de dicha base, independientemente del tipo de herencia utilizado	F

CLASE

1	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
2	Una clase derivada puede añadir nuevos métodos/atributos propios de la clase derivada, pero no modificar los métodos heredados de la clase base	F
3	Una clase es una especificación abstracta de una estructura de datos y de las operaciones que se pueden realizar con ella	F

METODOS

1	Desde un método de una clase derivada nunca puede invocar un método implementado con identidad signatura de su clase	F
2	Desde un método de una clase derivada nunca puede invocar un método implementado en esta con el mismo nombre que en la clase base	F
3	Desde un método de una clase derivada nunca puede invocarse a un método implementado con la identidad signatura de una clase de sus clases base	F
4	El método invocado por un objeto en respuesta a un mensaje viene siempre determinado, entre otras cosas por la clase del objeto receptor en tiempo de compilación	F
5	En java es obligatorio indicar que un método de una clase derivada sobrescribe un método de la clase base con la misma signatura	F
6	La llamada a un método sobrescrito se resuelve en tiempo de compilación	F
7	La signatura de tipo de un método incluye el tipo devuelto por el método	V
8	Los métodos definidos en una clase derivada nunca pueden acceder a las propiedades privadas de una clase base	V
9	Los métodos que usan referencias a clases base deben ser capaces de usar objetos de clases derivadas sin saberlo, es una posible formulación del principio de inversión de dependencias	F
10	Los métodos virtuales siempre tienen enlace dinámico	F
11	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
12	Un método de clase (estático) no puede tener enlace dinámico	V

DOWNCASTING

Acto de convertir una referencia de una clase base en una de sus clases derivadas		
1	El downcasting estático siempre es seguro	F
2	El downcasting implica deshacer el principio de sustitución	V
3	El downcasting siempre es seguro.	F
4	En JAVA el downcasting siempre se realiza en tiempo de ejecución	V

OTRAS

1	Con el diseño OO es perfectamente posible y deseable hacer uso de variables globales	F
2	Cuando usamos la varianza estamos haciendo un uso inseguro de la herencia de implementación	V
3	Declarar un dato miembro de una clase como private indica que sólo puede acceder a ese atributo desde las funciones miembro de la clase	F
4	El enlace de la invocación a un método sobrescrito se produce en tiempo de ejecución en función del tipo del receptor del mensaje	V
5	El principio abierto-cerrado indica que un componente software debe estar abierto a su extensión y cerrado a su modificación	V
6	El principio de segregación de interfaz indica que el código cliente no debe ser forzado a depender de interfaces que no utilice	V
7	El principio de sustitución implica una coerción entre tipos de una misma jerarquía de clases	V
8	El puntero this es un puntero constante al objeto que recibe el mensaje	V
9	El recolector de basura es un mecanismo de liberación de recursos presente en todos los lenguajes OO	F
10	En el diseño mediante tarjetas CRC utilizamos una tarjeta para cada clase	V
11	En el diseño mediante tarjetas CRC, utilizamos una tarjeta por cada jerarquía de herencia.	F
12	En el diseño por contrato son dos componentes fundamentales las pre y pos condiciones	V
13	En el paradigma orientado a objetos, un objeto siempre es instancia de alguna clase	V
14	En el paradigma orientado a objetos, un programa es un conjunto de objetos que se comunican mediante el paso de mensajes	V
15	En el principio de sustitución implica una coerción entre tipos de una misma jerarquía de clases	V
16	En el proceso de diseño de un sistema de software se debería intentar aumentar el acoplamiento y la cohesión	F
17	En JAVA el concepto meta-clase se presenta con la clase Class	V
18	En JAVA el enlace por defecto de métodos de instancia es estático	F
19	En JAVA las sentencias de un bloque ‘finally’ solamente se ejecutan cuando se ha producido una excepción y no la hemos capturado en un bloque ‘catch’	F

20	Hablamos de shadowing cuando el método a invocar se decide en tiempo de compilación	V
21	La existencia de una relación todo-parte entre dos clases implica necesariamente que el objeto todo maneja la creación/destrucción de los objetos parte	F
22	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
23	La robustez de un sistema software es un parámetro de claridad intrínseco	F
24	Los TAD's representan una perspectiva orientada a datos, mientras que los objetos reflejan una perspectiva orientada a servicios	V
25	Si utilizamos los mecanismos de manejo de excepciones disminuye la eficiencia del programa incluso si no se llega a lanzar nunca una excepción	V
26	Toda sentencia que aparece después del punto del programa en el que ocurre una excepción, en ningún caso se ejecuta	F
27	Todo espacio de nombre define su propio ámbito, distinto de cualquier otro	V
28	Un buen diseño OO se caracteriza por un alto acoplamiento y una baja cohesión	F
29	Un espacio de nombres es un ámbito con nombre.	V
30	Un mensaje tiene cero o más argumentos. Por el contrario, una llamada o procedimiento/función tiene uno o más argumentos	F
31	Un sistema de tipos de un lenguaje asocia a cada tipo una expresión.	V
32	Una colaboración describe como un grupo de objetos trabaja conjuntamente para realizar una tarea	V
33	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
34	Una forma de mejorar el diseño de un sistema es reducir su acoplamiento y aumentar su cohesión	V
35	Una operación constante sólo puede ser invocada por un objeto constante	F
36	Una tarjeta CRC contiene el nombre de una clase, su lista de responsabilidades y si lista de colaboradores	V

MAS

1	El paradigma se define como la forma de entender y representar la realidad	V
2	Extensibilidad, reutilización y mantenibilidad son parámetros de calidad intrínsecos	V
3	Hablamos de enlace dinámico cuando nos referimos a tiempo de compilación	F
4	Hablamos de enlace estático cuando nos referimos a tiempo de ejecución	F
5	Hablamos de enlace estático para referirnos a tiempo de compilación y dinámico para referirnos a tiempo de ejecución	V
6	La corrección y la robustez son parámetros de calidad extrínsecos (fiabilidad)	V
7	La perspectiva de datos los paquetes y tipos abstractos de datos	V
8	La perspectiva de servicios los objetos (TAD, herencia, polimorfismo)	V
9	La perspectiva funcional tiene el ensamblador, procedimientos y módulos	V
10	La robustez de un sistema software es un parámetro de calidad intrínseco	F
1	En el cuerpo de una operación de clase no se puede acceder a ningún atributo/operación del objeto receptor del mensaje	V
2	En el cuerpo de una operación de clase no se puede acceder a ningún atributo/operación de instancia de los objetos pasados como parámetros	F
3	En el cuerpo de una operación de clase se puede acceder únicamente a atributos/operaciones de clase	V
4	Un atributo de clase ocupa una zona de memoria que es compartida por todos los objetos de la clase en la que se define, aunque no por los objetos de la clase derivada	F
1	El objetivo de un constructor es crear e inicializar objetos	V
2	En la misma clase podemos definir un constructor con distinta visibilidad	V
3	Un constructor acepta cualquier tipo de modificación (static, const, public, etc)	F
1	El método Clone() proporciona un Deep Copy en Java, que copia bit a bit el contenido del objeto pasado por parámetro	F
2	La única relación no persistente es la relación de uso, ya que no se materializa mediante referencias	V
3	Las relaciones todo –parte son relaciones persistentes como la herencia	V
4	Todo son relaciones de clase excepto la relación de uso que es relación de objeto	F

1	La sentencia derivada d = d2.clone(); es correcta si d y d2 son de tipo Derivada	V
2	Un atributo public en base es accesible en main desde un objeto derivado sea cual sea el tipo de herencia	F
1	Definir dos métodos que se llamen igual pero tengan distinto tipo se llama sobrecarga	F
2	En la sobrecarga nunca se podrá cambiar el tipo devuelto por un método	F
3	Los métodos de sobrecarga son el refinamiento y el remplazo	F
4	Para sobrecargar dos métodos dentro del mismo ámbito es necesario que difieran en el número, tipo y orden de sus argumentos	F
5	Un método sobrecargado es aquel que recibe como argumento una variable polimórfica	F
1	La principal utilidad de la genericidad es la de agrupar variables cuyo tipo base no está predeterminado	V
2	Un método sobrecargado es aquel que recibe como argumento una variable polimórfica	F
1	Con el objeto method podemos obtener su nombre y lista de parámetros e invocarlo	V
2	Con el objeto method podemos obtener un método a partir de una signatura, u obtener una lista de todos los métodos con esa signatura	V
3	Toda la clase que se carga en la JVM tiene asociado un objeto de tipo Class	V
1	Apache Lucene es un motor de búsqueda para la recuperación de información	V
2	GWT (Google Web Toolkit) es una generación de aplicaciones ricas web mediante la generación de JavaScript a partir de Java	V
3	Hibernate está asociado con objetos en Bases de Datos	V
4	Set, list and map son las interfaces del JCF, y set<list>, hashmap… son implementaciones de esas interfaces	V
1	La refactorización es un proceso que maneja la estructura interna de un sistema de software de forma que su comportamiento no varia	V
2	La refactorización es una forma sistemática de introducir mejoras en el código que minimiza la posibilidad de introducir errores en el	V
1	El puntero this es un puntero que no puede cambiar de valor y que contiene la dirección del objeto receptor del mensaje, además existe en cualquier método definido dentro de la clase	F