

## TEMA 1

¿Cuántos objetos *Fecha* se crean aquí (C++)?

```
Fecha f1, f2(12,9,2016);  
Fecha *f3 = new Fecha(f2);  
Fecha *f4 = f3;  
Fecha& f5 = f2;
```

Seleccione una:

- ☐ 1
- ☐ 2
- ☐ 4
- ☒ 3

Correcto, se trata de los objetos f1, f2 y f3.

- ☐ 5

### Retroalimentación

#### Respuesta correcta

Debes contar objetos 'automáticos', que se crean en la pila (stack) y objetos creados con 'new', que se crean en el 'heap'. También debes saber cuando una asignación entre objetos, punteros o referencias realiza una copia de un objeto (y por tanto crea uno nuevo) y cuando simplemente estamos copiando la dirección de un objeto, sin duplicarlo.

#### Enunciado de la pregunta

¿En. C++, porqué debemos usar 'delete' con objetos creados mediante 'new'?

Seleccione una:

- ☐ Porque nos obliga C++.
- ☐ No es necesario, solo es conveniente hacerlo.
- ☒ Porque es responsabilidad del programador administrar la memoria dinámica del programa.
- ☐ Porque si no lo hacemos provocaremos tarde o temprano un error de 'violación de segmento' ('segmentation fault')

### Retroalimentación

#### Respuesta correcta

#### Enunciado de la pregunta

```
Fecha f1(12,9,2016);  
Fecha *f5;  
if ( ... ) {  
    Fecha f2(f1);
```

```
...
} else if ( ... ) {
    Fecha f3(f1);
    ...
} else if ( ... ) {
    Fecha *f4 = new Fecha(f1);
    f5=f4;
    delete f5; f5=NULL;
}
// ¿Cuántos objetos 'Fecha' hay en este punto del código?
```

Respuesta:

### Retroalimentación

Sólo f1 ha sobrevivido. f2 y f3 han sido destruidos al terminar el bloque donde se habían definido. f4 y f5 son dos punteros apuntando al mismo objeto, que es destruido al hacer 'delete'.

### Enunciado de la pregunta

¿Dónde se guarda el valor de un atributo de instancia de un objeto que está en memoria?

Seleccione una:

- ☐ En la memoria que se reservó para la clase del objeto al crearlo.
- ☐ En el disco duro.
- ☐ En la memoria estática.
- ☒ En la memoria que se reservó para el objeto al crearlo.
- ☐ En una zona especial de la memoria destinada al almacenamiento de atributos, distinta de donde se guardan los objetos y sus punteros

### Retroalimentación

Respuesta correcta

### Enunciado de la pregunta

¿Por qué ocultamos los datos (atributos) en una clase?

Seleccione una o más de una:

- ☐ Para poder tener atributos que sólo usamos internamente en los métodos (funciones) de la clase pero que el código cliente no necesita.
- ☐ Para poder usar los nombres de atributos que queramos sin tener que publicarlos.
- ☒ Para ocultar los detalles de implementación de nuestra clase al código cliente.
- ☒ Para evitar que el código cliente pueda manipular el estado de nuestros objetos de forma que queden en un estado inconsistente o corrupto (p. ej., que el día de una fecha sea negativo o mayor que 31).

### Retroalimentación

### Respuesta correcta

#### Enunciado de la pregunta

Necesitamos una forma de crear objetos Fecha que se inicialicen con unos valores fijos (por ejemplo, a 1 de Enero de 2000) sin necesidad de indicar una fecha concreta. Tenemos que incluir en la clase

Seleccione una:

- ☐ Un método de instancia que lo haga, como por ejemplo Fecha::inicializar()
- ☐ Sobrecargar el operador de entrada
- ☒ Un constructor por defecto

El constructor por defecto nos permite inicializar los atributos de Fecha al crear el objeto. Creación del objeto e inicialización de sus propiedades son dos acciones que los lenguajes como C++ o Java realizan de forma conjunta mediante los constructores.

- ☐ Un constructor de oficio

### Retroalimentación

### Respuesta correcta

#### Enunciado de la pregunta

¿Cómo podemos crear objetos Fecha con valores iniciales diferentes?

Seleccione una:

- ☐ Implementando un constructor de copia.
- ☐ Sobrecargando el operador de entrada.
- ☒ Implementando un constructor sobrecargado.

Aunque Fecha::set(int dia, int mes, int anyo) hace prácticamente el mismo trabajo que el constructor sobrecargado (de hecho, podríamos llamar al método desde el

constructor), éste lo hace justo en el momento en el que se crea el objeto, garantizando que el objeto 'nace' con esos valores iniciales, antes de que podamos hacer uso de él de alguna forma, cosa que no está garantizada con el método set(...).

- ☐ Implementando un método de instancia, p. ej. Fecha::set(int dia, int mes, int anyo).

Retroalimentación

Respuesta correcta

Enunciado de la pregunta

¿Cómo podemos crear fechas que sean copia de otras?

Seleccione una:

- ☐ Implementando un método que haga la copia, como Fecha::copiar(Fecha f)
- ☐ Implementando un constructor sobrecargado.
- ☐ Implementando un constructor sobrecargado que luego usamos para copiar la fecha : Fecha f2(f1.getDía(), f1.getMes(), f1.getAnyo()).
- ☒ Implementando un constructor de copia.

Retroalimentación

Respuesta correcta

Enunciado de la pregunta

Cuando decimos que un objeto que hemos creado 'ha sido eliminado', 'ya no existe', ¿qué estamos diciendo exactamente?

Seleccione una:

- ☐ Que hemos hecho 'delete'.
- ☒ Que el objeto ha sido eliminado de la memoria de nuestro programa y ya no es accesible.
- ☐ Que la variable mediante el cual nos referimos a él ya no existe porque hemos salido del ámbito donde fue definida.

Retroalimentación

Respuesta correcta

Enunciado de la pregunta

¿Qué pasa cuando un objeto 'desaparece'?

Seleccione una o más de una:

- ☒ Es eliminado de la memoria de nuestro programa.
- ☐ Todas las variables que hacen referencia a él (punteros, referencias) ya no se pueden usar.
- ☒ Se ejecuta el destructor de la clase sobre ese objeto (C++)

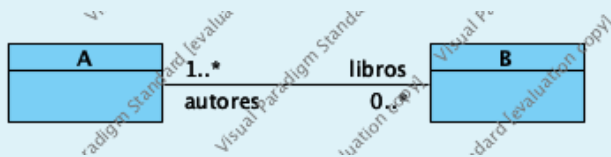
Retroalimentación

Respuesta correcta

## TEMA 2

In UML roles have a specific location. In this diagram, A and B are fictitious class names. They really should be called Author and Book.

Indicate whether A should be an Author and therefore B must be a Book or on the contrary A should be a Book and therefore B must be an Author, so that a book has been written by one or several authors and an author has been able to write none or some books.



Seleccione una:

- ☒ a. A represents the Author class and B represents Book
- ☐ b. A represents the Book class and B represents Author but the roles (authors, books) are misplaced, they should exchange their positions
- ☐ c. A represents the class Author and B represents Book but the roles (authors, books) are misplaced, they should exchange their positions
- ☐ d. A represents the Book class and B represents Author

Enunciado de la pregunta

Match every UML diagram with a relationship between classes.



Respuesta 1

Inheritance



Respuesta 2

Dependency



Respuesta 3

Aggregation



Respuesta 4

Composition



Respuesta 5

Association



Respuesta 6

Realization/implementation

## Enunciado de la pregunta

Choose the arity that best describes the relationships in the following UML diagrams:



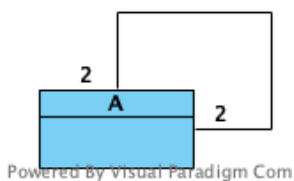
Respuesta 1

Binary



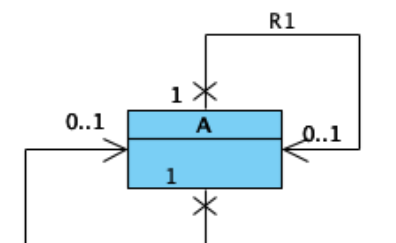
Respuesta 2

Binary



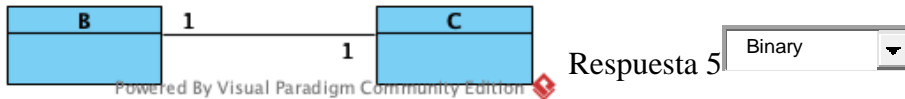
Respuesta 3

Unary



Respuesta 4

Unary



### Enunciado de la pregunta

Choose the most correct option.



It is difficult to implement this relationship because when we add an instance of B to another A we cannot know if that instance of B is already related to another A.



When an instance of A is destroyed, the instance of B it contains are available for use by another instance of A.



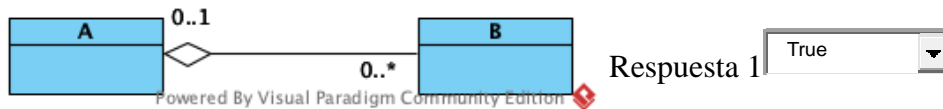
When an instance of A is destroyed, those instances of B contained in A are also destroyed.



Nonsense option.

### Enunciado de la pregunta

In the following diagrams, select TRUE when instances of B can access the object of class A that contains it



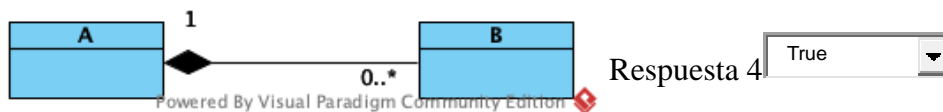
Respuesta 1



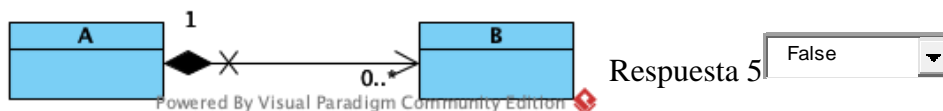
Respuesta 2



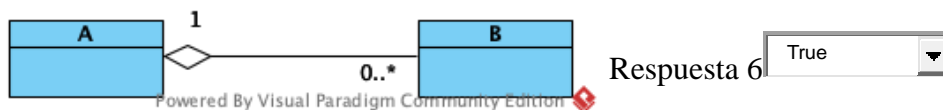
Respuesta 3



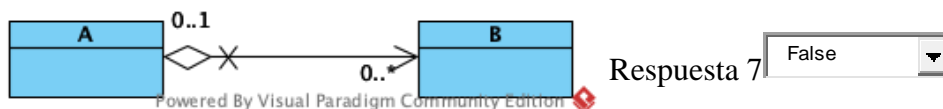
Respuesta 4



Respuesta 5



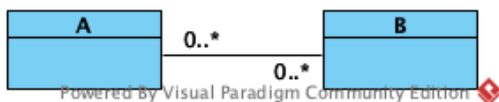
Respuesta 6



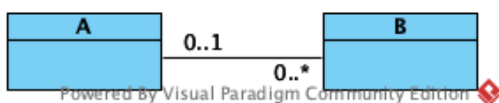
Respuesta 7

## Enunciado de la pregunta

Choose the correct option



Each instance of A is optionally related to several instances of B, which may be related to a variable number of instances of A, among which that instance of A may not be...

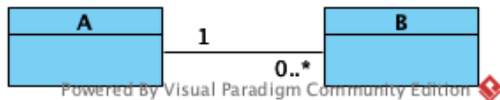




Each instance of A is optionally related to several instances of B, which may be related to that instance of A.



Each instance of A is always related to another instance of B, which in turn is optionally related to that instance of A.



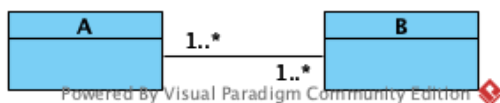
Each instance of A is optionally related to several instances of B, which are necessarily related to that instance of A.



Each instance of A is always related to another instance of B, which in turn is always related to that instance of A.



Each instance of A is always related to another instance of B. It has not yet been specified how many instances of A each instance of B will be related with.



Each instance of A is related to several instances of B, at least one, which may be related to a variable number of instances of A, at least one.

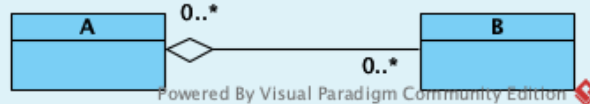
Enunciado de la pregunta

Choose the correct option.

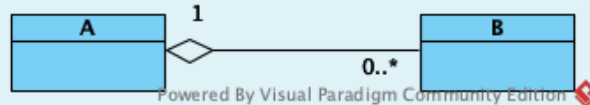
Question 1:



Question 2:



Question 3:



Question 1: Deep copy will be used in the copy constructor of A.

Question 2: Shallow copy will be used in the copy constructor of A.

Question 3: Shallow copy cannot be used in the copy constructor of A due to B's arity (B can only be related to one A).

### **TEMA 3**

Rellena con las palabras clave correctas, escribe la cadena VACIO si no es necesario que aparezca ninguna palabra clave

```
class Ejemplo {  
    private void F(int x) Respuesta 
```

Respuesta

```
{  
    if (x < 0) {  
        Respuesta 
```

Respuesta

```
Exception();  
    }  
}
```

Enunciado de la pregunta

Rellena con las palabras clave correctas, no escribas nada si no es necesario

```
class MiExcepcion extends Exception {  
    private int causante;
```

```

    public MiExcepcion(int x) {
        this.causante = x;
    }
    public int getCausante() {
        return causante;
    }
}

class Ejemplo {
    private void F(int x) Respuesta 

    MiExcepcion {
        if (x <= 0) {
            Respuesta 

            Respuesta 

            Respuesta 

            (Respuesta 

        );
    }
    // haz alguna cosa si no x es positiva
}

```

### Enunciado de la pregunta

Rellena con las palabras clave correctas, escribe la cadena VACIO si no es estrictamente necesario añadir otra palabra clave

```

class MiExcepcion extends RuntimeException {
    private int causante;
    public MiExcepcion(int x) {
        this.causante = x;
    }
    public int getCausante() {
        return causante;
    }
}

```

```

class Ejemplo {
    private void F(int x) Respuesta 

```

```

{
    if (x <= 0) {
        Respuesta 

    Respuesta 

    Respuesta 

    (x);
        }
        // haz alguna cosa si no x es positiva
    }
}

```

### Enunciado de la pregunta

Rellena con las palabras clave correctas, escribe la cadena VACIO si no es estrictamente necesario añadir otra palabra clave

```

class MiExcepcion extends Exception {
    private int causante;
    public MiExcepcion(int x) {
        this.causante = x;
    }
    public int getCausante() {
        return causante;
    }
}

```

```

class Ejemplo {
    private void F(int x) Respuesta 

    Respuesta 

    {
        if (x <= 0) {
            Respuesta 

            Respuesta 

            Respuesta 

```

```
(x);
    }
    // haz alguna cosa si no x es positiva
}
```

```
private void G(int y) Respuesta throws
```

```
Respuesta MiExcepcion
```

```
{
    if (y * y > 100) {
        F(x);
    }
}
```

### Enunciado de la pregunta

Rellena con las palabras clave correctas, escribe la cadena VACIO si no es estrictamente necesario añadir otra palabra clave

```
class MiExcepcion extends Exception {
    private int causante;
    public MiExcepcion(int x) {
        this.causante = x;
    }
    public int getCausante() {
        return causante;
    }
}
```

```
class Ejemplo {
    private void F(int x) Respuesta throws
```

```
Respuesta MiExcepcion
```

```
{
    if (x <= 0) {
        Respuesta throw
```

```
Respuesta new
```

```
Respuesta MiExcepcion
```

```

(x);
    }
    // haz alguna cosa si no x es positiva
}

private void G(int y) Respuesta VACIO

Respuesta VACIO

{
    Respuesta try

{
        if (y * y > 100) {
            F(y);
        }
    } catch (Respuesta MiExcepcion

e) {
        System.err.println("El valor introducido debe ser
positivo y es: " + e.Respuesta getCausante

));
    }
}

```

Rellena con las palabras clave correctas, escribe la cadena VACIO si no es estrictamente necesario añadir otra palabra clave

```

class MiExcepcion extends Exception {
    private int causante;
    public MiException(int x) {
        this.causante = x;
    }
    public int getCausante() {
        return causante;
    }
}

```

```

class Ejemplo {
    int [] v;

    public Ejemplo(int tamanyo) {
        v = new int[tamanyo];
    }
}

```

```

        private void F(int indice, int valor) Respuesta throws
        {
            MiExcepcion {
                try {
                    v[indice] = valor;
                } catch (Respuesta
                    {
                        e) { // capturamos que el índice se sale del array
                            Respuesta
                                {
                                    Respuesta
                                        {
                                            Respuesta
                                                {
                                                    (indice);
                                                }
                                            }
                                        }
                                    // haz alguna cosa si no x es positiva
                                }
                            private void G(int indice, int y) Respuesta throws
                            {
                                Respuesta
                                    {
                                        {
                                            try
                                                {
                                                    if (y * y > 100) {
                                                        F(indice, y);
                                                    }
                                                }
                                            } catch (Respuesta
                                                {
                                                    e) {
                                                        int indiceMal = e.Respuesta
                                                            {
                                                                ();
                                                                System.err.println("El valor introducido debe ser positivo y es: " +
                                                                indiceMal);
                                                                throw new MiExcepcion(indice);
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

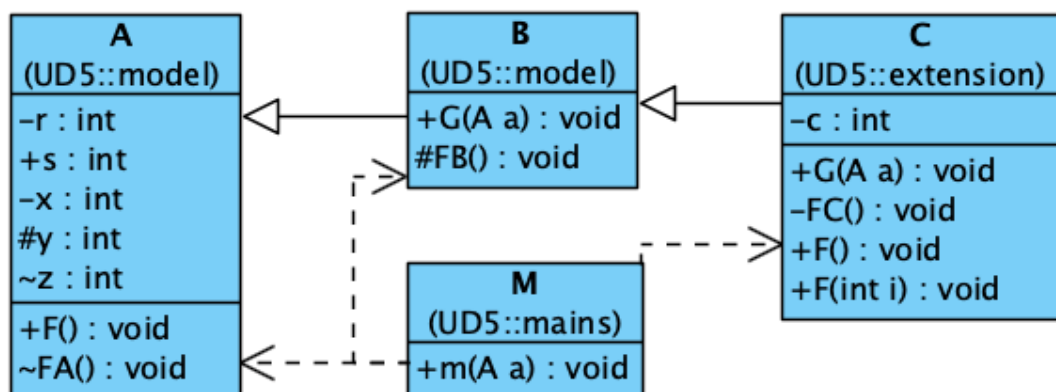
```

## Enunciado de la pregunta

Rellena con las palabras clave correctas, escribe la cadena VACIO si no es estrictamente necesario añadir otra palabra clave

```
public class Ejemplo {  
    public static void main(String[] args) Respuesta VACIO  
  
    Respuesta VACIO  
  
    {  
        FileOutputStream fos = null;  
  
        try {  
            fos = new FileOutputStream("unfichero.txt");  
  
            fos.write(65);  
            fos.flush();  
  
        } catch (IOException Respuesta ex  
  
    ) {  
        System.out.print("Ha ocurrido un error de E/S: " + ex.getMessage());  
    } Respuesta finally  
  
    {  
        if(fos!=null) {  
            fos.close();  
        }  
    }  
}
```

## TEMA 4





Usando este diagrama, selecciona la opción correcta. En el código, donde aparecen tres puntos (...) estamos indicando que hay más código que no mostramos.

```
package mains;
```

```
import model.A;  
import extension.C;
```

```
public class M {  
  
    public void m(A a) {  
        A c = new C();  
        c.F();  
    }  
}
```

Respuesta 1

Ejecuta la implementación de F() en C

```
package mains;
```

```
import model.A;  
import model.B;
```

```
public class M {  
  
    public void m(A a) {  
        B b = new B();  
        b.G(a);  
        b.F();  
    }  
}
```

Respuesta 2

Ejecuta la implementación de F() en A

```
package mains;
```

```
import model.A;  
import model.B;
```

```
public class M {  
  
    public void m(A a) {  
  
        A b = new B();  
        b.G(a);  
        b.F();  
    }  
}
```

Respuesta 3

No compila

```
package mains;
```

```
import model.*;  
import extension.C;
```

```
public class M {  
  
    public void m(A a) {  
        a = new C();  
    }  
}
```

Respuesta 4

Ejecuta la implementación de F() en A

```
        A a2 = new B();
        a = a2;
        a.F();
    }
}
```

```
package mains;
import model.A;
import model.B;
```

```
public class M {
    public void m(A a) {
        if (a instanceof B) {
            a.G(new A());
        }
    }
}
```

Respuesta 5

No compila

```
package mains;
import model.A;
public class M {
    public void m(A a) {
        a.FA();
    }
}
```

Respuesta 6

No compila

```
package mains;
import model.A;
import model.B;
```

```
public class M {
    public void m(A a) {
        if (a instanceof B) {
            ((B)a).G(new A());
        }
    }
}
```

Respuesta 7

Correcto

```
package mains;
```

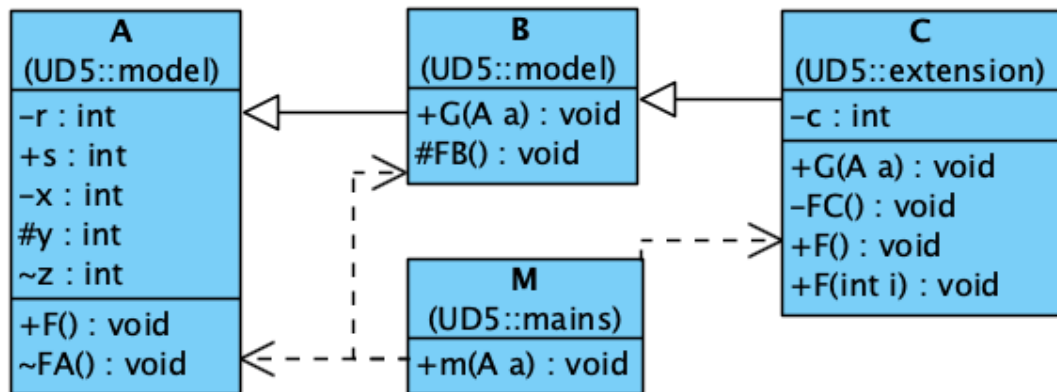
```
import model.A;
import extension.C;
```

```
public class M {
    public void m(A a) {
        A c = new C();
        c.F(4);
    }
}
```

Respuesta 8

No compila

Enunciado de la pregunta



Usando este diagrama, selecciona la opción correcta. En el código, donde aparecen tres puntos (...) estamos indicando que hay más código que no mostramos.

```

package model;
public class B extends A {
    ...

    public void G(A a) {
        this.z = a.y;
    }
    ...
}
  
```

Respuesta 1

Correcto

```

package model;
public class A {
    ...

    void FA() {
        r = s = x = y =
z;
    }
}
  
```

Respuesta 2

Correcto

```

package extension;
import model.B;
public class C extends B {
    ...

    public void G(A a) {
        this.c = a.z;
    }
}
  
```

Respuesta 3

No compila porque el acceso es de paquete

```
    }  
    ...  
}
```

---

```
package model;  
public class B extends A {  
    ...  
    public void G(A a) {  
        this.y = a.s;  
    }  
    ...  
}
```

Respuesta 4

Correcto

```
package extension;  
import model.B;  
public class C extends B {  
    ...  
    private void FC() {  
        FB();  
    }  
    ...  
}
```

Respuesta 5

Correcto

```
package extension;  
import model.B;  
public class C extends B {  
    ...  
    private void FC() {  
        FA();  
    }  
    ...  
}
```

Respuesta 6

No compila porque el acceso es de paquete

---

```
package model;

public class B extends A {

    ...

    public void G(A a) {

        int cont = a.r;

    }

    ...

}
```

Respuesta 7

No compila porque el acceso es privado

---

```
package extension;

import model.B;

public class C extends B {

    ...

    public void G(A a) {

        this.c = a.y;

    }

    ...

}
```

Respuesta 8

Correcto

---

```
package model;

public class B extends A {

    ...

    public void G(A a) {

        this.r = a.s;

    }

    ...

}
```

Respuesta 9

No compila porque el acceso es privado

---

```
package model;

public class B extends A {

    ...

    public void G(A a) {
```

Respuesta 10

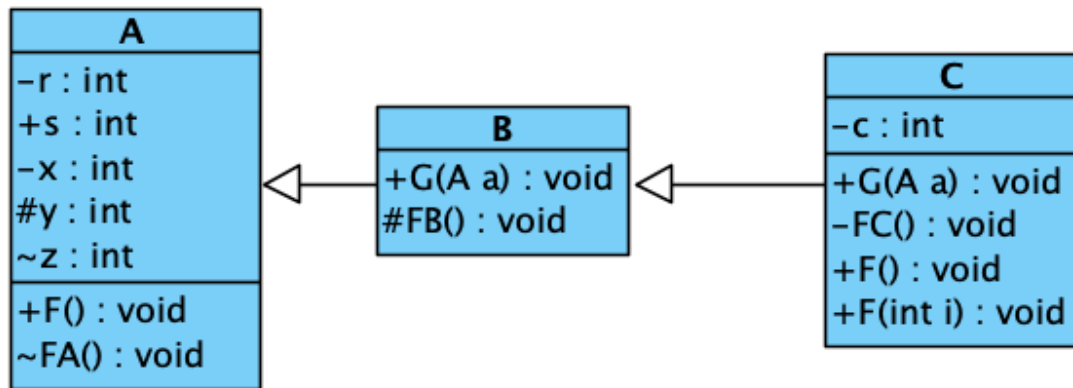
Correcto

```

        FA();
    }
    ...
}

```

### Enunciado de la pregunta



Usando como referencia el diagrama de clases, rellena con las palabras clave correctas el código. Si no hay que escribir nada, introduce la cadena VACIO.

```
package model;
```

```
public class A {
```

```
    Respuesta
```

```
private
```

```
    int r;
```

```
    Respuesta
```

```
public
```

```
    int s;
```

```
    Respuesta
```

```
private
```

```
    int x;
```

```
    Respuesta
```

```
protected
```

```
    int y;
```

```
    Respuesta
```

```
VACIO
```

```
    int z;
```

```
    Respuesta
```

```
public
```

```
void F() {  
    }  
    Respuesta VACIO
```

```
void FA() {  
    }  
}
```

---

```
package model;
```

```
public B Respuesta extends
```

```
Respuesta A
```

```
{  
    Respuesta public
```

```
void G(A a) {  
    }
```

```
    Respuesta protected
```

```
void FB() {  
    }  
}
```

---

```
package extension;  
import model.B;
```

```
public C Respuesta extends
```

```
Respuesta B
```

```
{  
    Respuesta private
```

```
int c;
```

Respuesta

public

```
void G(A a) {  
}
```

Respuesta

private

```
void FC() {  
}
```

Respuesta

public

```
void F() {  
}
```

Respuesta

public

```
void F(int i) {  
}  
}
```

Si a partir de una clase Vehiculo, creo mediante herencia la clase Coche y la clase Motocicleta, estoy haciendo

Seleccione una o más de una:

- ☒ a. Una especialización
- ☐ b. Una variación
- ☐ c. Una excepción
- ☐ d. Una generalización

Retroalimentación

Respuesta correcta

Piensa en las clases como conjuntos que contienen a otros conjuntos (subclases).

La respuesta correcta es: Una especialización

Enunciado de la pregunta

Cuando a partir de una superclase, creamos subclases, decimos que éstas últimas

Seleccione una o más de una:

- ☐ a. reemplazan el comportamiento de la superclase



- ☒ b. extienden el comportamiento de la superclase
- ☐ c. refinan el comportamiento de la superclase
- ☒ d. heredan las propiedades de la superclase
- ☐ e. restringen el comportamiento de la superclase

#### Retroalimentación

Respuesta correcta: heredan las propiedades de la superclase y pueden añadir propiedades (extensión de la superclase).

Las respuestas correctas son: heredan las propiedades de la superclase, extienden el comportamiento de la superclase

#### Enunciado de la pregunta

Si mi clase 'MiExcepcion' hereda directamente de la clase 'Exception' y esta hereda a su vez de la clase 'Object' entonces

Seleccione una o más de una:

- ☐ a. 'Object' es una subclase de 'MiExcepcion'
- ☐ b. 'MiExcepcion' es una superclase de 'Object'
- ☒ c. 'Object' es una superclase de 'MiExcepcion'
- ☒ d. 'MiExcepcion' es una subclase de 'Object'

#### Retroalimentación

Respuesta correcta. La herencia es transitiva.

Las respuestas correctas son: 'MiExcepcion' es una subclase de 'Object', 'Object' es una superclase de 'MiExcepcion'

#### Enunciado de la pregunta

La herencia de implementación...

Seleccione una o más de una:

- ☒ a. nos permite reutilizar código, ya que heredamos tanto la interfaz como la implementación de la superclase.
- ☐ b. nos permite reutilizar conceptos, es decir, la interfaz de la clase, pero no código (implementación).
- ☐ c. sólo nos permite heredar la implementación de la superclase, no su interfaz.

### Retroalimentación

#### Respuesta correcta

La respuesta correcta es: nos permite reutilizar código, ya que heredamos tanto la interfaz como la implementación de la superclase.

#### Enunciado de la pregunta

Decimos que estamos haciendo herencia simple de implementación...

Seleccione una o más de una:

- ☐ a. cuando las instancias de las subclases representan a todas las instancias de la superclase.
- ☒ b. cuando una clase sólo tiene una superclase.
- ☐ c. cuando una clase sólo tiene una subclase.
- ☐ d. cuando hay instancias de la superclase que no están representadas por alguna subclase.

### Retroalimentación

#### Respuesta correcta

La respuesta correcta es: cuando una clase sólo tiene una superclase.

#### Enunciado de la pregunta

Las subclases...

Seleccione una o más de una:

- ☐ a. pueden añadir métodos nuevos pero no atributos.
- ☒ b. pueden modificar la implementación de los métodos heredados.
- ☐ c. no pueden modificar la implementación de los métodos heredados.
- ☒ d. pueden añadir atributos y métodos nuevos.

### Retroalimentación

Respuesta correcta: La subclase puede extender a la superclase (añadir métodos o atributos nuevos) o modificar su comportamiento (la implementación de los métodos heredados).

Las respuestas correctas son: pueden modificar la implementación de los métodos heredados., pueden añadir atributos y métodos nuevos.

### Enunciado de la pregunta

Si 'B' es una subclase de 'A', un método de instancia de 'B'...

Seleccione una o más de una:

- ☒ a. siempre puede acceder a las propiedades de 'A' que tienen visibilidad pública.
- ☐ b. siempre puede acceder a las propiedades de 'A' que tienen visibilidad privada.
- ☒ c. siempre puede acceder a las propiedades de 'A' que tienen visibilidad protegida.
- ☐ d. siempre puede acceder a las propiedades de 'A' que tienen visibilidad de paquete (package).

### Retroalimentación

Respuesta correcta. A las propiedades privadas de 'A' sólo puede acceder 'A'. A las propiedades 'package' sólo las clases que están en el mismo paquete que 'A'.

Las respuestas correctas son: siempre puede acceder a las propiedades de 'A' que tienen visibilidad protegida., siempre puede acceder a las propiedades de 'A' que tienen visibilidad pública.

### Enunciado de la pregunta

Si la clase 'B' está en el mismo paquete que 'A', un método de instancia de 'B'...

Seleccione una o más de una:

- ☒ a. siempre puede acceder a las propiedades de 'A' que tienen visibilidad pública.
- ☒ b. siempre puede acceder a las propiedades de 'A' que tienen visibilidad protegida.
- ☒ c. siempre puede acceder a las propiedades de 'A' que tienen visibilidad de paquete (package).
- ☐ d. siempre puede acceder a las propiedades de 'A' que tienen visibilidad privada.

### Retroalimentación

Respuesta correcta. A las propiedades privadas de 'A' sólo puede acceder 'A'. A las propiedades 'package' sólo las clases que están en el mismo paquete que 'A'.

Las respuestas correctas son: siempre puede acceder a las propiedades de 'A' que tienen visibilidad protegida., siempre puede acceder a las propiedades de 'A' que tienen visibilidad de paquete (package)., siempre puede acceder a las propiedades de 'A' que tienen visibilidad pública.

### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    public A() {}  
    public A(int x) {}  
}  
  
class B extends A {  
    public B() {}  
    public B(int y) {}  
}
```

Seleccione una o más de una:

- ☒ a. B.B(int) invoca implícitamente a A.A()
- ☒ b. B.B() invoca implícitamente a A.A()
- ☐ c. B.B(int) invoca implícitamente a A.A(int)
- ☐ d. B.B(int) invoca implícitamente a A.A() y luego a A.A(int)

### Retroalimentación

Respuesta correcta. El único constructor de la superclase que se invoca implícitamente desde la subclase es el constructor por defecto.

Las respuestas correctas son: B.B() invoca implícitamente a A.A(), B.B(int) invoca implícitamente a A.A()

### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    public A() {}  
    public A(int x) {}  
}  
  
class B extends A {  
    public B() {}  
    public B(int y) {}  
}
```

cuando hacemos 'new B(3)'...

Seleccione una o más de una:

- ☐ a. Primero se ejecuta B.B(int) y luego A.A(int)
- ☐ b. Primero se ejecuta A.A(int) y luego B.B(int)
- ☐ c. Primero se ejecuta B.B(int) y luego A.A()

- ☒ d. Primero se ejecuta A.A() y luego B.B(int)

#### Retroalimentación

Respuesta correcta. Primero se ejecuta el constructor de la superclase.

La respuesta correcta es: Primero se ejecuta A.A() y luego B.B(int)

#### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    private int a;  
    public A() { a=0; }  
    public A(int x) { a= x; }  
}
```

```
class B extends A {  
    private int b;  
    public B() { b=0; }  
    public B(int y) { b=y; }  
}
```

Si tenemos una referencia 'A objeto;'...

Seleccione una o más de una:

- ☐ a. podemos asignarle cualquier tipo de objeto. : 'objeto = new Object();'
- ☐ b. sólo podemos asignarle objetos de tipo 'A': 'objeto = new A();'
- ☒ c. podemos asignarle objetos de tipo 'A' directamente ('objeto = new A();') y objetos de tipo 'B' mediante upcasting explícito: 'objeto = (A) new B();'
- ☒ d. podemos asignarle objetos de tipo 'A' y 'B'. : 'objeto = new A();' y : 'objeto = new B();'

#### Retroalimentación

Respuesta correcta. El upcasting siempre se puede hacer, implícita o explícitamente.

Las respuestas correctas son: podemos asignarle objetos de tipo 'A' y 'B'. : 'objeto = new A();' y : 'objeto = new B();', podemos asignarle objetos de tipo 'A' directamente ('objeto = new A();') y objetos de tipo 'B' mediante upcasting explícito: 'objeto = (A) new B();'

#### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    private int a;  
    public A() { a=0; }  
    public A(int x) { a= x; }  
}
```

```
public int getA() { return a; }  
}
```

```
class B extends A {  
    private int b;  
    public B() { b=0; }  
    public B(int y) { b=y; }  
    public int getB() { return b; }  
}
```

Si tenemos una referencia 'A objeto = new B();'

Seleccione una o más de una:

- ☐ a. no podemos invocar a 'objeto.getA()'
- ☒ b. no podemos invocar a 'objeto.getB()'
- ☒ c. podemos invocar a 'objeto.getA()'
- ☐ d. podemos invocar a 'objeto.getB()'

### Retroalimentación

Respuesta correcta. Con una referencia de tipo superclase ('A') sólo podemos usar la interfaz de la superclase.

Las respuestas correctas son: podemos invocar a 'objeto.getA()', no podemos invocar a 'objeto.getB()'

### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    private int a;  
    public A() { a=0; }  
    public A(int x) { a= x; }  
    public int getA() { return a; }  
}
```

```
class B extends A {  
    private int b;  
    public B() { b=0; }  
    public B(int y) { b=y; }  
    public int getA() { return 1; }  
    public int getB() { return b; }  
}
```

Si tenemos una referencia 'A objeto = new B(3);', el resultado de llamar a 'objeto.getA()' es

Seleccione una o más de una:

- ☐ a. 0
- ☒ b. 1



c. 3



d. Un error de compilación.

#### Retroalimentación

Respuesta correcta. Se ejecuta la implementación de la subclase, por que el objeto 'en tiempo de ejecución' al que apunta 'objeto' es de tipo subclase.

La respuesta correcta es: 1

#### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    protected int a;  
    public A() { a=0; }  
    public A(int x) { a= x; }  
    public int getA() { return a; }  
}  
  
class B extends A {  
    private int b;  
    public B() { b=0; }  
    public B(int y) { b=y; }  
    public int getA() { return a+3; }  
    public int getB() { return b; }  
}
```

Si tenemos una referencia 'A objeto = new B(3);', el resultado de llamar a 'objeto.getA()' es

Seleccione una o más de una:



a. 0



b. 3



c. 1



d. Un error de compilación.

#### Retroalimentación

Respuesta correcta. 'B' puede acceder a las propiedades protegidas de 'A' y se ejecuta la implementación de 'B'.

La respuesta correcta es: 3

### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    protected int a;  
    public A() { a=0; }  
    public A(int x) { a= x; }  
    public int getA() { return a; }  
}
```

```
class B extends A {  
    private int b;  
    public B() { b=0; }  
    public B(int y) { b=y; }  
  
    public int getB() { return b; }  
}
```

Si tenemos una referencia '**B objeto = new B(3);**', el resultado de llamar a 'objeto.getA()' es

Seleccione una o más de una:

- ☐ a. 3
- ☐ b. 1
- ☒ c. 0
- ☐ d. Un error de compilación.

### Retroalimentación

Respuesta correcta.

'B' ha heredado getA() de 'A'.

La respuesta correcta es: 0

### Enunciado de la pregunta

Dadas estas clases:

```
class A {  
    protected int a;
```



```
public A() { a=0; }  
public A(int x) { a= x; }  
public int getA() { return a; }  
}  
  
class B extends A {  
    private int b;  
    public B() { b=0; }  
    public B(int y) { b=y; }  
    public int getA() { return a+3; }  
    public int getB() { return b; }  
}
```

Si tenemos una referencia '**B objeto = new A(3);**', el resultado de llamar a 'objeto.getA()' es

Seleccione una o más de una:

- ☒ a. Un error de compilación.
- ☐ b. 1
- ☐ c. 3
- ☐ d. 0

#### Retroalimentación

Respuesta correcta

La respuesta correcta es: Un error de compilación.

### **TEMA 5**

En el siguiente código

```
class A {}  
class B extends A {}  
  
...  
A obj = new B();
```

Seleccione una:

- ☐ a. El tipo en tiempo de compilación de 'obj' es A y su tipo en tiempo de ejecución también es A.

- ☒ b. El tipo en tiempo de compilación de 'obj' es A y su tipo en tiempo de ejecución es B.
- ☐ c. El tipo en tiempo de compilación de 'obj' es B y su tipo en tiempo de ejecución es A.
- ☐ d. El tipo en tiempo de compilación de 'obj' es B y su tipo en tiempo de ejecución también es B.

Enunciado de la pregunta

Indica en que fase del proceso de compilación se realiza la comprobación de tipos.  
Seleccione una:

- ☐ a. Análisis sintáctico
- ☐ b. Análisis léxico
- ☐ c. Generación de código
- ☒ d. Análisis semántico

Enunciado de la pregunta

Indica la opción correcta para la máquina virtual de Java (JVM):

Seleccione una:

- ☐ a. En la pila se almacenan cualquier variable local u objeto si el método que se está ejecutando es un método estático.
- ☐ b. Las variables locales (tipos primitivos, referencias a objetos) se almacenan en el heap y los objetos en la pila
- ☒ c. Las variables locales (tipos primitivos, referencias a objetos) se almacenan en la pila y los objetos en el heap
- ☐ d. En el heap se almacenan cualquier variable local u objeto si el método que se está ejecutando es un método de instancia.

Enunciado de la pregunta

Los atributos estáticos

Seleccione una:

- ☐ a. se almacenan en la memoria asignada a cada instancia de la clase donde se definen
- ☐ b. se almacenan en la pila
- ☒ c. se almacenan en la memoria asignada al objeto Class de la clase donde se definen

### Enunciado de la pregunta

Dado este código:

```
public class Animal {  
    public void come() {  
        System.out.println("Animal, ¡come!");  
    }  
}
```

```
public class Serpiente extends Animal {  
    public void come() {  
        System.out.println("Serpiente, ¡come!");  
    }  
    public void reptar() {  
        System.out.println("¡Repta!");  
    }  
}
```

```
public class Ave extends Animal {  
    public void mudaPlumas() {  
        System.out.println("Ave, ¡muda las plumas!");  
    }  
    public void come() {  
        System.out.println("Ave, ¡come!");  
    }  
}
```

```
public class AveVoladora extends Ave {  
    public void vuela() {  
        System.out.println("¡Vuela!");  
    }  
}
```

```
public class Pinguino extends Ave {  
    public void come() {  
        System.out.println("Pingüino, ¡come!");  
    }  
}
```

```
public class Cliente {
```

```

    public Animal F(char c) {
        Animal animal = null;

        switch (c) {
            case 'a':
                animal = new Ave();
                break;
            case 'v':
                animal = new AveVoladora();
                break;
            case 'p':
                animal = new Pinguino();
                break;
            case 's':
                animal = new Serpiente();
                break;
            default:
                animal = new Animal();
        }

        return animal;
    }
}

```

Indica la opción correcta

```

public class Main {
    public static void
main(String[] args) {
    Cliente m = new
Cliente();

    Animal a = null;
    if
(Math.random()>0.5)
        a = m.F('s');
    else
        a = m.F('p');

    Serpiente s =
(Serpiente) a;
    s.repta();
}
}

```

Respuesta 1

Podría fallar en tiempo de ejecución

```

public class Main {
    public static void
main(String[] args) {
    Cliente m = new
Cliente();

    Animal a =
m.F('s');
    a.repta();
}
}

```

Respuesta 2

No compila

```

    }
}
public class Main {
    public static void
main(String[] args) {

        Cliente m = new
Cliente();

        Animal a =
m.F('s');

        if (a instanceof
Serpiente) {

            Serpiente s
= (Serpiente) a;

            s.repta();

        }

    }
}

```

Respuesta 3

Se está haciendo un downcasting seguro

```

public class Main {
    public static void
main(String[] args) {

        Animal a = new
Serpiente();

        a.come();

    }
}

```

Respuesta 4

Se está haciendo un upcasting

## Enunciado de la pregunta

Dado este código:

```

public class Animal {
    public void come() {
        System.out.println("Animal, ¡come!");
    }
}

```

```

public class Serpiente extends Animal {
    public void come() {
        System.out.println("Serpiente, ¡come!");
    }
    public void reptar() {
        System.out.println("¡Repta!");
    }
}

```

```

public class Ave extends Animal {
    public void mudaPlumas() {
        System.out.println("Ave, ¡muda las plumas!");
    }
    public void come() {
        System.out.println("Ave, ¡come!");
    }
}

```

```
}
```

```
public class AveVoladora extends Ave {  
    public void vuela() {  
        System.out.println("¡Vuela!");  
    }  
}
```

```
public class Pinguino extends Ave {  
    public void come() {  
        System.out.println("Pingüino, ¡come!");  
    }  
}
```

```
public class Main {  
    public void F(char c) {  
        Animal animal = null;  
  
        switch (c) {  
            case 'a':  
                animal = new Ave();  
                break;  
            case 'v':  
                animal = new AveVoladora();  
                break;  
            case 's':  
                animal = new Serpiente();  
                break;  
            case 'p':  
                animal = new Pinguino();  
                break;  
            default:  
                animal = new Animal();  
        }  
  
        animal.come();  
    }  
}
```

Indica lo que se imprime por pantalla al llamar al método F(char c) dadas las siguientes opciones

c='s' Respuesta 1

c = 'p' Respuesta 2

c = 'a' Respuesta 3

c = 'v' Respuesta 4

Ave, ¡come!

### Enunciado de la pregunta

Indica si las siguientes afirmaciones son correctas.

Sabemos que este código de Javascript no da errores sintácticos. Este lenguaje usa un sistema de comprobación de tipos dinámico. Por tanto, puede funcionar porque tiene un sistema de tipos débil.

```
js> x="5" <-- String
5
js> x=x*5 <-- Int
25
```

Respuesta 1

Cierto

El siguiente código es correcto en una **comprobación de tipos dinámica**

```
class A {}
class B extends A { int b; }
a = new B();
a.b;
```

Respuesta 2

Cierto

Todo lenguaje que use una comprobación de tipos estática usa tipado fuerte

Respuesta 3

Falso

Sabemos que este código de Python da error. Este lenguaje usa un sistema de comprobación de tipos dinámico. Por tanto, el error lo da porque Python usa un tipado fuerte.

```
x = "Hola"
y = 2019
print x + y
```

Respuesta 4

Cierto

Todo lenguaje que use una comprobación de tipos dinámica usa tipado débil

Respuesta 5

Falso

El siguiente código es correcto en una **comprobación de tipos estática**

```
class A {}
class B extends A { int b; }
A a = new B();
a.b;
```

Respuesta 6

Falso

### Enunciado de la pregunta

Siguiendo el orden de los números en los comentarios, sigue la traza de ejecución del código desde el programa principal y escribe las palabras correctas en los huecos.

Las palabras válidas son:

stack  
heap  
compilación  
ejecución

NOTA: se han omitido algunos artículos en las oraciones para evitar dar pistas adicionales a las respuestas.

```
public class Personaje {  
    protected String nombre;  
    public Personaje(String nombre) {this.nombre = nombre;}  
    // 19º. Se añade a Respuesta
```

stack

el constructor de Personaje

```
    // 20º. Se reserva en Respuesta
```

stack

memoria para el parámetro 'nombre'

```
    // 21º. Se asigna en Respuesta
```

heap

la referencia 'this.nombre' al valor del parámetro 'nombre'

```
    // 22º Cuando acaba este constructor de Personaje se elimina de  
Respuesta
```

stack

```
    public String getNombre() {return this.nombre;}  
    public String expresate() {return "";}  
}
```

```
public class Animal extends Personaje {  
    Extremidad[] extremidades; // composicion  
    protected Animal(String nombre, Extremidad [] extremidades) {  
    // 17º. Se añade a Respuesta
```

stack

el constructor de Animal

```
    // 18º. Se reserva en Respuesta
```

stack

memoria para los parámetros 'nombre' y 'extremidades'

```
        super(nombre);
```

```
        this.extremidades = new Extremidad[extremidades.length];
```

```
    // 23º. Se reserva en Respuesta
```

heap

la memoria para un array de Extremidades

```
    // 24º. Se asigna en Respuesta
```

heap



```
la referencia 'this.extremidades' al objeto anterior
    for (int i=0; i < extremidades.length; i++) {
        this.extremidades[i] = extremidades[i].clona();
        // En tiempo de Respuesta compilación
```

```
se comprueba que la clase Extremidad
    // tiene un método clona() que retorna un tipo
Extremidad o compatible
    // Cuando i=0, como el tipo de extremidades[i] en
tiempo de Respuesta ejecución
```

```
    // es Brazo, se invoca al método clona() de Brazo

    //Para i=1 se repiten los pasos 25º, 26º y 27º
    //Para i=2 y luego para i=3 se repiten los pasos
28º, 29º y 30º
    }
}
// 31º. Se elimina de Respuesta stack
```

```
el constructor de Animal
}
```

```
public class Arbol extends Personaje {
    public Arbol(String nombre) {super(nombre);}
}
```

```
public class Humano extends Animal {
    private ArrayList frases; // composición
    protected Collection mascotas; // agregación

    public Humano(String nombre, Collection frases) {
        // 10º. Se añade a Respuesta stack
```

```
el constructor de Humano
```

```
    // 11º. Se añade a Respuesta stack
```

```
los parámetros nombre y frases
```

```
    super(nombre,
    new Extremidad[] {
        // 13º. Se reserva en Respuesta heap
```

```
la memoria para un objeto de tipo array Extremidad[]
```

```
    // 14º. Se reserva en Respuesta stack
```

la memoria para una referencia al objeto anterior

```
new Brazo(),  
new Brazo(),  
new Pierna(),  
new Pierna()}
```

); // 15º. Se reserva en Respuesta

heap

la memoria de dos instancias de Brazo y dos de Pierna

// 16º. Se enlazan en Respuesta

heap

las referencias a estas cuatro instancias dentro del objeto array.

// El tipo en tiempo de Respuesta

compilación

de esas cuatro referencias es Extremidad

// El tipo en tiempo de Respuesta

ejecución

de las dos primeras referencias es Brazo y de las otras dos es Pierna

this.frases = new ArrayList<>(frases); // no necesitamos  
duplicar las cadenas porque son objetos inmutables

// 32º. Se reserva en Respuesta

heap

la memoria para un objeto de tipo ArrayList

// 33º. Se añade a Respuesta

stack

el constructor de ArrayList a partir de una colección de String

// Se ejecutará internamente ese constructor de ArrayList

// 34º. Se asigna en Respuesta

heap

la referencia 'this.frases' a la nueva instancia de ArrayList

mascotas = new ArrayList<>();

// 35º. Se reserva en Respuesta

heap

la memoria para un objeto de tipo ArrayList

// 36º. Se añade a Respuesta

stack

el constructor de ArrayList por defecto

// Se ejecutará internamente ese constructor de ArrayList

// 37º. Se asigna en Respuesta

heap

la referencia 'this.mascotas' a la nueva instancia de ArrayList

```
// 38°. Se elimina de Respuesta
```

stack

el constructor de Humano

```
public String expresate() {
    String frase = frases.get((int) (Math.random() *
frases.size()));
    StringBuilder sb = new StringBuilder();
    sb.append(frase);
    for (Animal mascota: mascotas) {
        sb.append(',');
        sb.append(mascota.expresate());
    }
    return sb.toString();
}

public void addMascota(Animal animal) {
    mascotas.add(animal);
}
}
```

```
public class Perro extends Animal {
    protected Perro(String nombre) {
        super(nombre, new Extremidad[] {new Pata(), new Pata(), new
Pata(), new Pata()});
    }
    public String expresate() {return "Guau";}
}
```

```
public class Extremidad {
    public Extremidad clona() {return new Extremidad();}
}
```

```
public class Brazo extends Extremidad {
    public Extremidad clona() {return new Brazo();}

    // 25°. Se añade a Respuesta
```

stack

el método clona()

```
// 26°. Se reserva en Respuesta
```

heap

la memoria para la nueva instancia de Brazo

```
// El tipo en tiempo de Respuesta
```

ejecución

del valor devuelto por el método clona() es Brazo

```
// El tipo en tiempo de Respuesta compilación

del valor devuelto por el método clona() es Extremidad
// Se invoca al constructor por defecto de Brazo, luego al de
Extremidad

// 27º. Se elimina de Respuesta stack

el método clona()
}
```

```
public class Pierna extends Extremidad {
    public Extremidad clona() {return new Pierna();}

    // 28º. Se añade a Respuesta stack
```

```
el método clona()

// 29º. Se reserva en Respuesta heap
```

```
la memoria para la nueva instancia de Pierna

// El tipo en tiempo de Respuesta ejecución
```

```
del valor devuelto por el método clona() es Pierna

// El tipo en tiempo de Respuesta compilación
```

```
del valor devuelto por el método clona() es Extremidad
// Se invoca al constructor por defecto de Pierna, luego al de
Extremidad

// 30º. Se elimina de Respuesta stack
```

```
el método clona()
}
```

```
public class Pata extends Extremidad {
    public Extremidad clona() {return new Pata();}
}
```

```
public class Cuento {
    protected List personajes;

    public Cuento() {

        // 3º. Se añade a Respuesta stack
```

el constructor

```
        creaPersonajes();  
    }
```

```
    public void creaPersonajes() {
```

```
        // 4º. Se añade a Respuesta
```

stack

el método creaPersonajes()

```
        personajes = new ArrayList<>();
```

```
        // 5º. Se reserva en Respuesta
```

heap

la memoria para un objeto de tipo ArrayList

```
        // 6º. Se enlaza en el Respuesta
```

heap

la referencia 'personajes' de esta instancia

```
        List<String> frasesLucky = new ArrayList<>();
```

```
        // 7º. Se reserva en Respuesta
```

heap

la memoria para un objeto de tipo ArrayList<String>

```
        // 8º. Se reserva en Respuesta
```

stack

la memoria para la referencia 'frasesLucky'

```
        frasesLucky.add("Soy un pobre vaquero solitario");
```

```
        frasesLucky.add("Estoy lejos de mi hogar");
```

```
        Humano lucky = new Humano("Lucky", frasesLucky);
```

```
        // 9º. Se reserva en Respuesta
```

heap

la memoria de esta instancia de tipo Humano

```
        personajes.add(lucky);
```

```
        // 39º. Se añade a Respuesta
```

stack

el método add de ArrayList,

```
        // donde el tipo de su argumento proporcionado en tiempo de
```

Respuesta

compilación


es Personaje

```
        // y en tiempo de Respuesta
```


ejecución

es Humano

```
        // El ejercicio acaba en este paso 39º - se invita al alumno  
a continuar con el resto de traza
```

```
        Perro rantamplan = new Perro("Rantamplán");  
        lucky.addMascota(rantamplan);  
        personajes.add(rantamplan);  
        personajes.add(new Arbol("Árbol del desierto"));  
    }  
  
    public void personajesSeExpresan() {  
        for (Personaje personaje: personajes) {  
            System.out.println(personaje.getNombre());  
            System.out.println("\t" + personaje.expresate());  
        }  
    }  
  
    public static final void main(String [] args) {  
        // 1º. Se añade a Respuesta 
```

el método estático main

```
        Cuento cuento = new Cuento();  
        // 2º. Se reserva en Respuesta 
```

la memoria para un nuevo objeto de tipo Cuento

```
        cuento.creaPersonajes();  
        cuento.personajesSeExpresan();  
    }  
}
```

---

## Enunciado de la pregunta

Dado este código:

```
public class Animal {  
    public void come() {  
        System.out.println("Animal, ¡come!");  
    }  
}
```

```
public class Serpiente extends Animal {  
    public void come() {  
        System.out.println("Serpiente, ¡come!");  
    }  
    public void reptar() {  
        System.out.println("¡Repta!");  
    }  
}
```

```
public class Ave extends Animal {  
    public void mudaPlumas() {  
        System.out.println("Ave, ¡muda las plumas!");  
    }  
}
```

```
    }  
    public void come() {  
        System.out.println("Ave, ¡come!");  
    }  
}
```

```
public class AveVoladora extends Ave {  
    public void vuela() {  
        System.out.println("¡Vuela!");  
    }  
}
```

```
public class Pinguino extends Ave {  
    public void come() {  
        System.out.println("Pingüino, ¡come!");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Cliente m = new Cliente();  
        Animal a = m.F(args[0].charAt(0));  
  
        a.come();  
        if (a instanceof Serpiente) {  
            // invoca a repta para la instancia a  
            Respuesta   
  
            .repta();  
  
        } else if (a instanceof Ave) {  
            Ave ave = Respuesta   
  
            a;  
  
            // invoca a mudaPlumas para la instancia ave  
            Respuesta   
  
            ave.mudaPlumas();  
  
            Respuesta   
  
            ave.come();  
  
            if (ave instanceof AveVoladora) {  
                // invoca a vuela para la instancia ave  
                Respuesta 
```

```
.vuela();  
  
    }  
}  
}
```

## **TEMA 6**

Selecciona la opción correcta

En C++, el tipo de enlace por defecto para el polimorfismo en herencia para métodos virtuales es

Respuesta 1

dinámico

En Java, donde conviven las distintas redefiniciones que encontramos en la jerarquía de herencia, se usa la estrategia de redefinición denominada

Respuesta 2

mezcla

El método toString() en Java es un ejemplo de sobrecarga de métodos basada en ...

Respuesta 3

ámbito

Las variables que pueden referenciar más de un tipo de objeto se denominan

Respuesta 4

polimórficas

La sobrecarga de operadores es un ejemplo de sobrecarga basada en ...

Respuesta 5

signatura de tipo

En C++, para que una clase sea polimórfica debe tener al menos un método

Respuesta 6

virtual

En C++, donde las redefiniciones en las subclases ocultan a las definiciones de las superclases, se usa la estrategia de redefinición denominada

Respuesta 7

shadowing

Cuando el método de la subclase oculta el acceso al método de la superclase, estamos hablando de

Respuesta 8

shadowing

En Java, para que una clase no sea polimórfica debemos usar la palabra clave

Respuesta 9

final

La sobrecarga de métodos de clase se resuelve en tiempo de

Respuesta 10

compilación

El downcasting en Java siempre se especifica de forma

Respuesta 11

explícita

En Java, si el método de la subclase se denomina igual que el de la superclase y tiene la misma signatura, usamos la anotación @Override, y por tanto es una

Respuesta 12

sobrescritura

Como alternativa a la sobrecarga, en C -p.ej. con la función printf-, o en Java -p.ej con un método en void F(int ... p)-, tenemos las funciones

Respuesta 13

poliádicas

En C++, el tipo de enlace por defecto para el polimorfismo en herencia para métodos es

Respuesta 14

estático



Las clases que tienen algún método con enlace dinámico denominan

En Java, el tipo de enlace para el polimorfismo en herencia para métodos de instancia públicos y protegidos es

En Java, esta llamada usa:

Object o = new String();

El mecanismo que permite conocer los tipos en tiempo de ejecución se denomina

En Java, el tipo de enlace para el polimorfismo en herencia para métodos privados, finales, de clase y atributos es

En Java, por defecto todas las clases son

Cuando el método en la superclase y en la subclase tienen distinta signatura de tipo estamos hablando de

Cuando un método sobrescrito en la subclase devuelve un tipo diferente pero compatible con el devuelto por la superclase estamos hablando de

Las llamadas a métodos sobrecargados se resuelven en tiempo de ...

Respuesta 15

polimórficas

Respuesta 16

dinámico

Respuesta 17

upcasting

Respuesta 18

RTTI

Respuesta 19

estático

Respuesta 20

polimórficas

Respuesta 21

redefinición

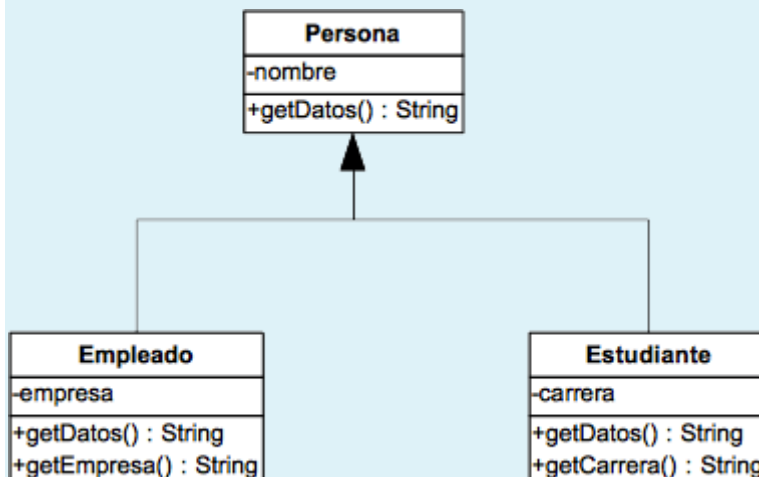
Respuesta 22

covarianza

Respuesta 23

compilación

Dado el siguiente modelo



y parte de la implementación a continuación:

```
class Persona {
    public Persona(String n)    {nombre=n;}
    public String getDatos() {return nombre;}
    ...
    private String nombre;
```

```

}

class Empleado extends Persona {
    public Empleado(String n,String e)
    { super(n); empresa=e; }
    public String getDatos()
    { return super.getDatos()+" trabaja en " + empresa; }
    ...
    private String empresa;
}

class Estudiante extends Persona {
    public Estudiante(String n,String c)
    { super(n); carrera=c; }
    public String getDatos()
    { return super.getDatos() + " estudia " + carrera; }
    ...
    private String carrera;
}

```

¿Cuál será el resultado de tratar de compilar y ejecutar los siguientes fragmentos de código? Indica si

- se produce un error de compilación
- se produce un error de ejecución
- compila y ejecuta correctamente

Si piensas que se produce algún error de compilación o de ejecución, indica en qué línea se produce el primer error (1 para la primera, 2 para la segunda, etc.). Si crees que se compila y ejecuta sin errores, indica el resultado que aparecerá en la salida estándar. Indica VACIO donde no proceda indicar respuesta.

### Enunciado de la pregunta

```

1 Empleado empleado = new Empleado("Carlos","Lavandería");
2 Persona pers = new Persona("Juan");
3 empleado = pers;
4 System.out.println( empleado.getDatos() );

```



Error de compilación



Error de ejecución



Compila y ejecuta correctamente

Línea de error: Respuesta

3

Resultado de la ejecución correcta: Respuesta

VACIO

### Enunciado de la pregunta

```
1 Estudiante estudiante = null;  
2 Persona pers = new Persona("José");  
3 estudiante = pers;  
5 System.out.println( estudiante.getDatos() );
```

- ☒ Error de compilación
- ☐ Error de ejecución
- ☐ Compila y ejecuta correctamente

Línea de error: Respuesta

Resultado de la ejecución correcta: Respuesta

### Enunciado de la pregunta

```
1 Empleado emp = new Empleado("Carlos", "lavanderia");  
2 Estudiante est = new Estudiante("Juan", "Derecho");  
3 Persona pers;  
4 pers = emp;  
5 System.out.println( pers.getDatos() );
```

- ☐ Error de compilación
- ☐ Error de ejecución
- ☒ Compila y ejecuta correctamente

Línea de error: Respuesta

Resultado de la ejecución correcta: Respuesta

### Enunciado de la pregunta

```
1 Empleado uno= new Empleado("Carlos", "lavanderia");  
2 Persona desc = uno;  
3 System.out.println( desc.getEmpresa() );
```

- ☒ Error de compilación

- ☐ Error de ejecución
- ☐ Compila y ejecuta correctamente

Línea de error: Respuesta

Resultado de la ejecución correcta: Respuesta

#### Enunciado de la pregunta

```
1  Persona desc = new Persona("Carlos");  
2  Empleado emp = (Empleado)desc;  
3  System.out.println( emp.getEmpresa() );
```

- ☐ Error de compilación
- ☒ Error de ejecución
- ☐ Compila y ejecuta correctamente

Línea de error: Respuesta

Resultado de la ejecución correcta: Respuesta

#### Enunciado de la pregunta

```
1  Persona desc = new Persona("Carlos");  
2  if (desc instanceof Empleado) {  
3      Empleado emp = (Empleado)desc;  
4      System.out.println( emp.getEmpresa() );  
5  } else  
4      System.out.println( desc.getDatos() );
```

- ☐ Error de compilación
- ☐ Error de ejecución
- ☒ Compila y ejecuta correctamente

Línea de error: Respuesta

Resultado de la ejecución correcta: Respuesta

Carlos

### Enunciado de la pregunta

Supongamos que el método `getDatos()` en la clase `Persona` ha sido definido de la siguiente forma:

```
public final String getDatos() { return (nombre); }
```

y eliminada su implementación de las subclases. Dado el siguiente código:

```
1 Empleado emp = new Empleado("Carlos", "lavanderia");
2 Persona pers;
3 pers = emp;
4 System.out.println( emp.getDatos() );
```



Error de compilación



Error de ejecución



Compila y ejecuta correctamente

Línea de error: Respuesta

VACIO

Resultado de la ejecución correcta: Respuesta

Carlos

### Enunciado de la pregunta

Supongamos que el método `getDatos()` en la clase `Persona` ha sido definido de la siguiente forma:

```
public static String getDatos() { return (nombre); }
```

y eliminada su implementación de las subclases. Dado el siguiente código:

```
1 Empleado emp = new Empleado("Carlos", "lavanderia");
2 Persona pers;
3 pers = (Persona) emp;
4 System.out.println( pers.getDatos() );
```



Error de compilación



Error de ejecución



Compila y ejecuta correctamente

Línea de error: Respuesta

VACIO

Resultado de la ejecución correcta: Respuesta

Carlos

## **TEMA 7**

En Java no existe ningún tipo de herencia múltiple, en C++ sí

Seleccione una:



Verdadero



Falso

### Retroalimentación

En Java se pueden implementar distintas interfaces, por lo que podemos hablar de herencia múltiple de interfaz. Además, podemos tanto implementar una interfaz como heredar (extends) de una clase

### Enunciado de la pregunta

En Java, un interfaz no puede incluir atributos de instancia y sólo puede contener declaraciones de métodos públicos sin implementación. La excepción a esto último son los métodos **default** de las últimas versiones de Java.

Seleccione una:



Verdadero



Falso

### Enunciado de la pregunta

En Java, una clase abstracta tiene que tener al menos un método abstracto

Seleccione una:



Verdadero



Falso

### Retroalimentación

La única restricción es que la clase se defina con la palabra clave **abstract**. Además, podemos poner uno o más métodos abstractos.

Cuando establecemos una clase abstracta, estamos evitando que se puedan crear instancias de esa clase.

### Enunciado de la pregunta

Para cumplir el principio de sustitución de Liskov, siendo 'B' una subclase de 'A', en el siguiente caso

```
A ref = new B();
```

para que no se invoque un método de A que B no implementa, debemos sustituir la variable por un downcasting:

```
B ref2 = (B) ref;
```

y usar 'ref2' en vez de 'ref'.

Seleccione una:



Verdadero



Falso

### Retroalimentación

Justamente es lo contrario, se debe poder invocar a cualquier método de la superclase independientemente de la subclase con que se haya instanciado la variable ('ref' en este caso).

### Enunciado de la pregunta

Emplearemos los interfaces para asegurarnos que una clase sabe responder, como mínimo, a unos mensajes concretos. En el caso de que una clase implemente o realice una interfaz y no sobrescriba los métodos definidos en ésta, la clase deberá ser obligatoriamente abstracta.

Seleccione una:



Verdadero



Falso

### Retroalimentación

Enviar un mensaje es equivalente a invocar a un método. Cuando una clase implementa o realiza una interfaz ésta debe implementar los métodos declarados en esa interfaz, y si alguno no se implementa la clase no podrá ser instancia. En ese caso, para indicar que la clase no se puede instanciar la debemos declarar como abstracta.


### Enunciado de la pregunta

En Java, los métodos de las interfaces son públicos por defecto.

Seleccione una:



Verdadero

 Falso

## Retroalimentación

Se puede poner public, pero no es necesario

## **TEMA 8**

Rellena con las palabras clave correctas de forma que compile, escribe la cadena VACIO si no es necesario que aparezca ninguna palabra clave

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

abstract class Figura {
    abstract public void pintar();
}

class Circulo extends Figura {
    @Override
    public void pintar() {
    }
}

class Rectangulo extends Figura {
    @Override
    public void pintar() {
    }
}

class Lienzo {
    private List figuras;

    public void Lienzo() {
        figuras = new LinkedList();
    }

    public void add(List<Respuesta ? extends Figura

> listaFiguras) {
        for (Figura figura : listaFiguras) {
            this.figuras.add(figura);
        }
    }
}

public class Genericidad {
    void F() {
        Lienzo lienzo = new Lienzo();

        List<Respuesta Figura
```



```

> fs = new ArrayList<>();
    fs.add(new Circulo());
    fs.add(new Rectangulo());

    lienzo.add(fs);

    List<Circulo> circulos = new ArrayList<>();
    circulos.add(new Circulo());
    lienzo.add(circulos);
}

```

## Retroalimentación

¿Por qué hemos usado genericidad restringida en el método:

```
public void add(List<? extends Figura> listaFiguras)
```

Si el prototipo fuera:

```
public void add(List<Figura> listaFiguras)
```

la llamada `lienzo.add(fs);` sería correcta porque envía un `ArrayList<Figura>` para `List<Figura>`, que son compatibles.

Sin embargo, la llamada **`lienzo.add(circulos)`** envía `ArrayList<Circulo>` que no es compatible con `List<Figura>`. Esto se resuelve indicando que `Lienzo.add` puede recibir tanto una lista de `Figura` como de cualquier tipo que herede de `Figura` con genericidad restringida: `<? extends Figura>`, con lo que queda:

```
public void add(List<? extends Figura> listaFiguras)
```

## Enunciado de la pregunta

Rellena con las palabras clave correctas de forma que compile, escribe la cadena VACIO si no es necesario que aparezca ninguna palabra clave

Suponemos que las clases a instanciar están todas en el paquete "figuras"

```
public static Figura crearFigura(String figura) throws
InstantiationException, IllegalAccessException, ClassNotFoundException {
```

```
    Class<?> c = Class.forName(
```

```
(Respuesta
```

```
figura);
```

```
    Figura objeto = Respuesta
```

```
c.Respuesta
```

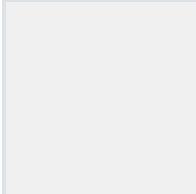
```
());
```

```
        return objeto;
    }
```

## Retroalimentación

Pregunta **3**  
Correcta

Puntúa 1,00 sobre 1,00



Marcar pregunta

## Enunciado de la pregunta

En Java, los tipos genéricos sólo se pueden usar para parametrizar clases, como por ejemplo

```
class ClaseParametrizada<T> {
    T unCampo;
    void calcula(T a) {
        ....
    }
}
```

Seleccione una:



Verdadero



Falso

## Retroalimentación

No es cierto, también pueden valer para parametrizar métodos. P.ej. este método de clase:

```
static <T> boolean comparar(T a, T b) {
    .....
}
```

## Enunciado de la pregunta

En Java, podemos incluir varios tipos parametrizados dentro del operador diamante, e incluso anidar varios operadores diamante.

Seleccione una:



Verdadero

☐ Falso

#### Retroalimentación

Sí, dentro del operador diamante, es decir, <>, se pueden incluir un número arbitrario de tipos parametrizados. Un tipo parametrizado puede ser a su vez también parametrizado. Por ejemplo:

```
clase A<Tipo1, Tipo2<Tipo1>, Tipo3> {  
    Tipo1 a;  
    Tipo2<Tipo1> b;  
    List<Tipo3> c;  
}
```

### **TEMA 9**

El principio de "inversión de control" implica que al usar un *framework* cierta funcionalidad será implementada por métodos de usuario que serán invocados por el código del *framework* mediante enlace dinámico.

Seleccione una:

☒ Verdadero

☐ Falso

#### Retroalimentación

En efecto, un framework ofrece interfaces y/o clases abstractas cuyos métodos pueden ser sobrescritos por el usuario, con el objetivo de ser invocados por el código del propio framework.

#### Enunciado de la pregunta

Una opción para la inversión de control en los frameworks consiste en implementar métodos de la interfaz del framework y que sea éste quien decida cuándo se invocan.

Seleccione una:

☒ Verdadero

☐ Falso

#### Enunciado de la pregunta

La inversión de control en los frameworks es posible gracias al enlace estático de métodos.

Seleccione una:

☐ Verdadero

☒ Falso

Enunciado de la pregunta

Un *framework* es una infraestructura para fines específicos que debemos completar y personalizar mediante la implementación de interfaces, herencia de clases abstractas, anotaciones o ficheros de configuración.

Seleccione una:

☒ Verdadero

☐ Falso

Enunciado de la pregunta

Sea un conjunto de clases X que nos descargamos en Java en formato .jar, indica la correcta denominación de ese conjunto de clases según la descripción:

Para usar las clases se requiere que implementemos una serie de interfaces que serán invocadas por X

Respuesta 1

Framework

Para usar X, simplemente debemos instanciar sus clases e invocar a métodos en éstas

Respuesta 2

Librería

Para usar X, instanciamos sus clases e invocamos a métodos en éstas. Además, podemos usar ciertas funcionalidades adicionales si heredamos unas clases abstractas que serán invocadas por X

Respuesta 3

Framework

Retroalimentación

Respuesta correcta

Enunciado de la pregunta

Una de las formas de utilizar un framework es mediante la implementación de interfaces definidas por éste.

Seleccione una:

☒ Verdadero

☐ Falso

Enunciado de la pregunta

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

Seleccione una:

☒ Verdadero

☒ Falso

Enunciado de la pregunta

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

Seleccione una:

☐ Verdadero

☒ Falso

Retroalimentación

No es necesario implementar todas las interfaces de un framework. Piensa, por ejemplo, en lo que sucede cuando usas el Java Collection Framework.

Enunciado de la pregunta

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

Seleccione una:

☒ Verdadero

☐ Falso

Enunciado de la pregunta

La inversión de control es una técnica de programación utilizada en las librerías.

Seleccione una:

☐ Verdadero

☒ Falso

Enunciado de la pregunta

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

Seleccione una:

☐ Verdadero

☒ Falso

## Asocia los conceptos a términos

Sentencias 'switch'

Respuesta 1

Código maloliente / sospechoso

Clase con demasiados métodos y/o atributos

Respuesta 2

Código maloliente / sospechoso

Antes de realizar ninguna refactorización debemos tener un conjunto completo de ...

Respuesta 3

Tests unitarios

Atributos, métodos, variables no usadas

Respuesta 4

Código maloliente / sospechoso

Para minimizar los errores y simplificar su corrección cuando hacemos cambios en mantenimiento, o bien lo refactorizamos, o bien añadimos una nueva funcionalidad, pero no ambas cosas al mismo tiempo.

Respuesta 5

Metáfora de los dos sombreros

Métodos muy largos

Respuesta 6

Código maloliente / sospechoso

Teníamos un código probado y funcionando correctamente. Tras unos cambios en el código derivados por una petición de nuestro cliente hay cosas que han dejado de funcionar.

Respuesta 7

Cambios estructurales

Hemos cometido un error en el diseño de la jerarquía de clases por entender mal el problema. Para corregirlo añadimos nuevas clases.

Respuesta 8

Error de regresión

Código duplicado

Indica cuándo es conveniente refactorizar el código.

Seleccione una o más de una:

- ☐ una vez implementado y probado el código, poco antes de que se cumplan los plazos de entrega.
- ☒ cuando hay código maloliente.
- ☒ antes de añadir una nueva funcionalidad.
- ☐ cuando el código original es tan 'malo' (por diseño, múltiples errores de implementación, etc.) que merece la pena reescribirlo desde cero.
- ☒ al encontrar un error (bug) en el código

Retroalimentación

Respuesta correcta

Enunciado de la pregunta

Indica en qué se ocupa la mayor parte del tiempo de desarrollo de una aplicación:  
Seleccione una:

- ☐ a. en arreglar los errores del código de la aplicación
- ☐ b. en programar el código de la aplicación
- ☒ c. en encontrar los errores en el código de la aplicación

#### Retroalimentación

#### Respuesta correcta

#### Enunciado de la pregunta

Indica cuáles de las siguientes son propiedades que deberían cumplir las pruebas unitarias.

Seleccione una o más de una:

- ☒ deben ser repetibles siempre en las mismas condiciones (por ejemplo, siempre con el mismo estado de persistencia)
- ☒ deben ser independientes entre sí
- ☐ es recomendable que las diferentes pruebas unitarias dependan unas de otras para poder capturar posibles errores al integrar diferentes partes de una aplicación.
- ☐ deben poder repetirse en diferentes condiciones (por ejemplo, con diferentes estados de persistencia)

#### Retroalimentación

#### Respuesta correcta

#### Enunciado de la pregunta

Indica cuáles de las siguientes son propiedades que deberían cumplir las pruebas unitarias.

Seleccione una o más de una:

- ☒ deben ser automatizables
- ☒ deben tener una cobertura del 100% del código que se prueba
- ☐ deben poder ejecutarse manualmente
- ☐ no es necesario que sean completas

#### Retroalimentación

### Respuesta correcta

#### Enunciado de la pregunta

Indica cuáles de las siguientes son propiedades que deberían cumplir las pruebas unitarias.

Seleccione una o más de una:

- ☐ las pruebas son una parte opcional del desarrollo de aplicaciones, lo importante es centrarse en el desarrollo del código que finalmente formará parte de la aplicación.
- ☒ cada prueba unitaria debería diseñarse para probar un único caso de uso de un único método.
- ☐ resultan muy útiles para capturar posibles errores de integración entre diferentes módulos de una aplicación.
- ☒ Las pruebas deben considerarse como parte esencial del desarrollo de software, equiparándolas en importancia al propio código para el cual se diseñan.

### Retroalimentación

### Respuesta correcta

## **TEMA 11**

#### Relaciona los conceptos con términos

```
interface IA {  
    void F();  
}  
interface IB {  
    void G();  
}  
interface IC {  
    void H();  
}  
class A implements IA {  
    IC c;  
    public A(IC c) {  
        this.c = c;  
    }  
    @Override  
    public void F() {  
    }  
    public void F2() {  
        c.J();  
    }  
}  
  
class B implements IB {
```

#### Respuesta 1

Las clases están acopladas



```

        IA a;
        public B(IA a) {
            this.a = a;
        }
        @Override
        public void G() {
            a.F();
        }
    }

    class C implements IB {
        IB b;
        public C(IB b) {
            this.b = b;
        }
        @Override
        public void H() {
            b.G();
        }
        public void J() {
        }
    }
}

```

```

class A {
    C c = new C();
    public void F() {
    }
    public void F2() {
        c.J();
    }
}

```

```

class B {
    A a = new A();
    public void G() {
        a.F();
    }
}

```

```

class C {
    B b = new B();
    public void H() {
        b.G();
    }
    public void J() {
    }
}

```

- Clases 'Singleton'
- Alto acoplamiento
- Intestabilidad
- Optimización prematura
- Nombrado no descriptivo
- Duplicación

## Respuesta 2

Las clases están acopladas

## Respuesta 3

Principios STUPID que debemos evitar

- Principio de segregación de interfaz
- Principio de inversión de dependencias
- Principio abierto/cerrado
- Principio de responsabilidad única
- Principio de sustitución de Liskov

#### Respuesta 4

Principios SOLID que debemos seguir

### Enunciado de la pregunta

Indica, para cada nombre, si se trata de un principio (buena práctica) o un anti-principio (mala práctica).

Código duplicado	Respuesta 1	Antiprincipio (Mala práctica)
Nombres no descriptivos	Respuesta 2	Antiprincipio (Mala práctica)
Intesteable	Respuesta 3	Antiprincipio (Mala práctica)
Alto acoplamiento	Respuesta 4	Antiprincipio (Mala práctica)
Optimización prematura	Respuesta 5	Antiprincipio (Mala práctica)
Inversión de dependencias	Respuesta 6	Principio (buena práctica)
Segregación de interfaz	Respuesta 7	Principio (buena práctica)
Sustitución (Liskov)	Respuesta 8	Principio (buena práctica)
Singleton	Respuesta 9	Antiprincipio (Mala práctica)
Responsabilidad única	Respuesta 10	Principio (buena práctica)
Abierto/Cerrado	Respuesta 11	Principio (buena práctica)