

ESTE EJERCICIO ES DE TIPO MIXTO.

ES IRRELEVANTE SI CONTESTA A LA PREGUNTA DE TEST O NO. SIN EMBARGO, SE DEBE ESCANEAR DICHA HOJA JUNTO CON EL RESTO DE LA CONTESTACIÓN DEL EXAMEN. EL ALUMNO PUEDE QUEDARSE CON EL ENUNCIADO.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

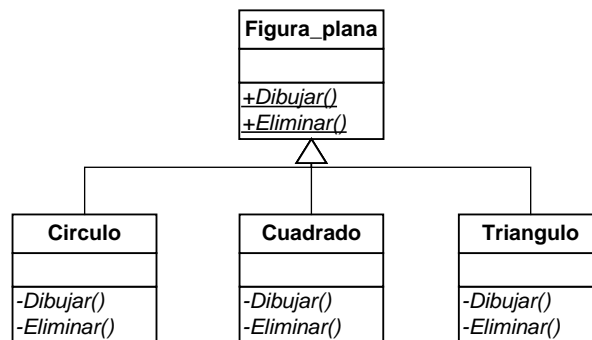
PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Defina el concepto de '*configuración software*'. Explique qué es y cómo se utiliza la '*línea base*' para gestionar los cambios en la configuración del software.

Solución

Epígrafe 1.9.5 del libro; páginas 30 a 32.

2. En el diagrama de objetos de la figura, observe que **Figura_plana** es una clase con polimorfismo. El método **Circulo.Dibujar()** sobrecarga el método de **Figura_plana**; y ocurre lo mismo con **Eliminar()** y con los respectivos métodos de **Cuadrado** y **Triangulo**.



El operador **new** se encarga de crear una instancia invocando a un método –denominado *constructor*– de una clase o un objeto determinado. Así, la sentencia:

Figura_plana Mi_figura = **new** **Circulo()**

crea la instancia **Mi_figura**, de tipo **Figura_plana**, mediante el constructor de **Circulo** –que es el método **Circulo()**, implícito en la clase **Circulo**–.

- a. ¿La sentencia anterior generaría correctamente el código o el compilador daría un mensaje de error por diferencia en los tipos? ¿Qué tipo se asigna, finalmente, a **Mi_figura**?
- b. Si se hace la llamada **Mi_figura.Dibujar()**, ¿qué método se ejecuta, el de **Figura_plana** o el de **Circulo**?

Solución

El hecho de *conectar* la invocación de un método con el código que le corresponde, se llama **acoplamiento** o **enlace**. Cuando el compilador conoce exactamente qué código le corresponde a la invocación de un método, la asignación se denomina enlazado temprano (*early binding*) y se produce durante la compilación, antes de la ejecución. Sin embargo, si durante la compilación no se sabe qué código le corresponde exactamente en la invocación de un método, la asignación se resuelve durante la ejecución del programa, cuando las referencias toman un valor concreto. A esto se le llama acoplamiento tardío (*late binding*) o **enlazado dinámico**.

- a. En el caso de la sentencia anterior, se crea un objeto de tipo **Circulo** e, inmediatamente, la referencia obtenida se asigna a un objeto, `Mi_figura`, de tipo **Figura_plana**. La sentencia es correcta y el compilador no daría error puesto que, por la herencia, un **Circulo** *es*, también, una **Figura_plana**. A esto se le llama **generalización** y es una transformación del tipo hacia arriba (*upcasting*).
- b. Se ha establecido que el tipo que el compilador ha asignado a `Mi_figura` es **Figura_plana** -enlazado temprano—. Sin embargo al invocar, **durante la ejecución**, al método `Mi_figura.Dibujar()`, el **polimorfismo** -o enlazado dinámico— garantiza que el método que se ejecuta en la invocación es la especialización correcta de **Circulo** -`Circulo.Dibujar()`— puesto que ese es el tipo de `Mi_figura`.

En las descripciones anteriores se pone de manifiesto de qué manera, la herencia y el polimorfismo, permiten diseñar con ocultación, ahorrar en el desarrollo de código -el cual resulta compacto, sencillo y fácilmente entendible— y lleva a diseños flexibles en los que es fácil añadir o eliminar elementos.

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Se desea diseñar un sistema informático para controlar la apertura y cierre de la compuerta trasera, articulada, de un camión. La compuerta está formada por dos plataformas cuyo movimiento se realiza con dos motores –M1 y M2— accionados mediante tres palancas –P1, P2 y P3—. Unos sensores, llamados ‘*final de carrera*’ detectan que cada plataforma ha llegado al límite de su movimiento: FC1_M1, FC2_M1, FC1_M2, FC2_M2 y FC3_M2. Una vez que se activa el sistema, las órdenes de las palancas se envían a un pequeño autómatas que alimenta a los motores. Dicho autómatas debe contener la lógica necesaria para que las maniobras de Apertura y Cierre sean seguras y correctas. Según la figura de más abajo:

- 1) La palanca P1 controla M1 y permite subir y bajar la compuerta en vertical. El sensor FC2_M1 indicará si la compuerta está a ras del suelo (0%, FC2_M1=1) y FC1_M1 si está arriba del todo (100%, FC1_M1=1).
- 2) La palanca P2 acciona el M2 para abrir y cerrar la compuerta –la segunda plataforma—. Los finales de carrera indicarán el ángulo correcto de la apertura de la compuerta. Se considera que la compuerta está abierta si el ángulo es de 0 grados (FC2_M2=1) y cerrada si el ángulo es de 90 grados (FC1_M2=1).
- 3) La palanca P3 acciona también el M2 e inclina más la plataforma para facilitar la carga y descarga del camión. El ángulo que puede girarse la compuerta con esta palanca va de 0 (FC2_M2=1) a -10 grados (FC3_M2=1).

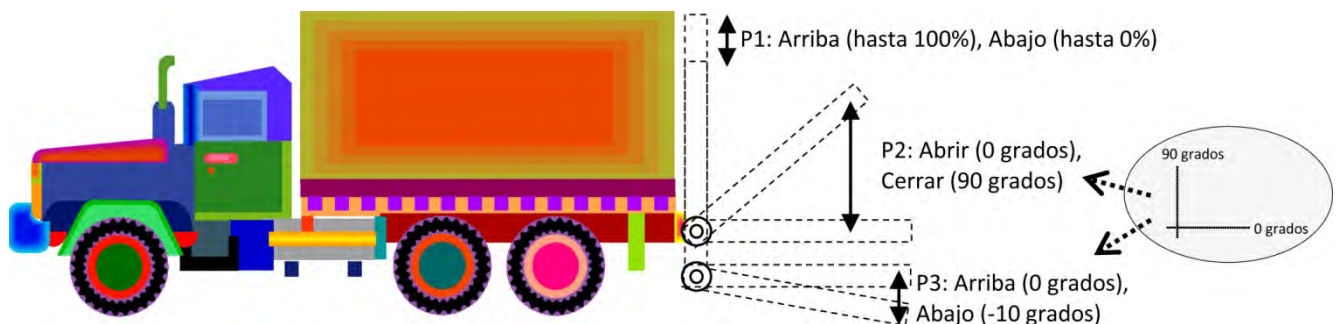
Las maniobras de Apertura y Cierre son las siguientes:

El proceso de apertura se consigue accionando las palancas: P2 (abrir) → P1 (bajar) → P3 (inclinar)

El proceso de cierre se consigue accionando las palancas: P3 (arriba) → P1 (subir) → P2 (cerrar)

Restricciones:

- No se pueden pulsar dos palancas a la vez.
- La plataforma **NO** se inclinará –en ningún sentido— si no está **totalmente abierta** (ángulo ≤ 0 grados) y, además, la **compuerta totalmente bajada** (0% ó FC2_M1=1)
- La plataforma **NO** se cerrará si está **inclinada** (ángulo ≤ 0 grados) o la **compuerta no está totalmente subida** (elevación $< 100\%$ ó FC1_M1 $\neq 1$)
- La compuerta **NO** se elevará ni bajará hasta que la **plataforma esté totalmente horizontal** (ángulo = 0 grados o FC2_M2=1)



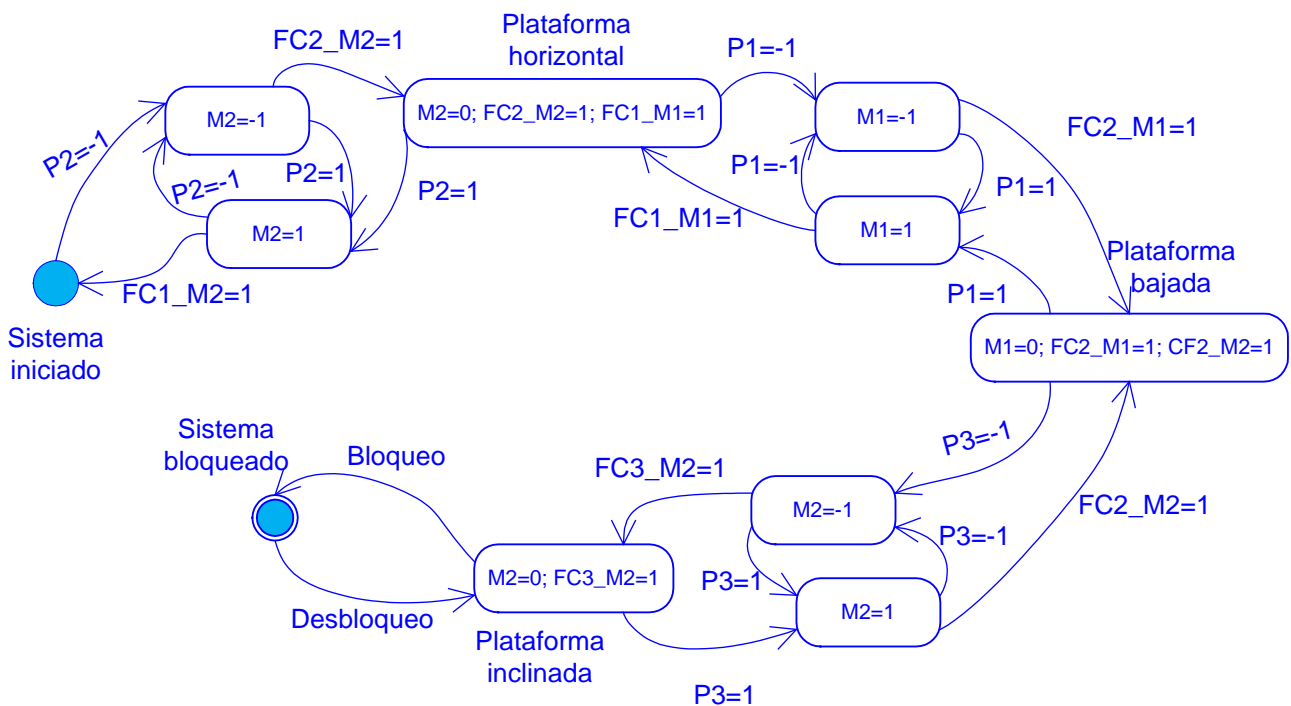
Se pide diseñar el funcionamiento descrito mediante un diagrama de transición de estados.

Solución

Supongamos que los valores que pueden tomar las variables del sistema son las siguientes:

Motores		Finales de carrera		Palancas	
M1	={-1, 0, 1} Bajar, parado, subir	FC1_M1	= {0, 1}	P1	={-1, 0, 1} Abajo, no pulsado, arriba
		FC2_M1	= {0, 1}		
M2	={-1, 0, 1} Abrir o bajar, parado, cerrar o subir	FC1_M2	= {0, 1}	P2	={-1, 0, 1} Abrir, no pulsado, cerrar
		FC2_M2	= {0, 1}		
		FC3_M2	= {0, 1}	P3	= {-1, 0, 1} Abajo, no pulsado, arriba

Las entradas al sistema serán las acciones de las palancas y el valor de los finales de carrera; las salidas, las señales de los motores. Los estados vendrán definidos por el movimiento de los motores y las posiciones de la compuerta. El diagrama de estados sería:



En el estado inicial, se verifican y calibran los accionadores, motores y finales de carrera. Los valores internos deben ser (estado 0):

ESTADO 0, REPOSO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =1
	FC2_M1 =0
M2 =0	FC1_M2 =1
	FC2_M2 =0
	FC3_M2 =0

La acción de la palanca P2 para abrir, permite la transición al movimiento de M2 hasta poner la plataforma horizontal o hasta que se pulse P2 en sentido contrario (1):

ESTADO 1, ABRIENDO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =1
	FC2_M1 =0
M2 =-1	FC1_M2 =0
	FC2_M2 =0
	FC3_M2 =0

ESTADO 2, ABIERTO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =1
	FC2_M1 =0
M2 =0	FC1_M2 =0
	FC2_M2 =1
	FC3_M2 =0

Al llegar a la posición horizontal el motor M2 se detiene y el sistema queda en el siguiente estado (2, de reposo).

A continuación, la sucesión del resto de estados:

ESTADO 3, BAJANDO	
MOTORES	FINALES DE CARRERA
M1 =-1	FC1_M1 =0
	FC2_M1 =0
M2 =0	FC1_M2 =0
	FC2_M2 =1
	FC3_M2 =0

ESTADO 6, INCLINADO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =0
	FC2_M1 =1
M2 =0	FC1_M2 =0
	FC2_M2 =0
	FC3_M2 =1

ESTADO 9, SUBIR	
MOTORES	FINALES DE CARRERA
M1 =1	FC1_M1 =0
	FC2_M1 =0
M2 =0	FC1_M2 =0
	FC2_M2 =1
	FC3_M2 =0

ESTADO 4, ABAJO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =0
	FC2_M1 =1
M2 =0	FC1_M2 =0
	FC2_M2 =1
	FC3_M2 =0

ESTADO 7, BLOQUEO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =0
	FC2_M1 =1
M2 =0	FC1_M2 =0
	FC2_M2 =0
	FC3_M2 =1

ESTADO 10, CERRANDO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =1
	FC2_M1 =0
M2 =1	FC1_M2 =0
	FC2_M2 =0
	FC3_M2 =0

ESTADO 5, INCLINANDO	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =0
	FC2_M1 =1
M2 =-1	FC1_M2 =0
	FC2_M2 =0
	FC3_M2 =0

ESTADO 8, ARRIBA	
MOTORES	FINALES DE CARRERA
M1 =0	FC1_M1 =0
	FC2_M1 =1
M2 =1	FC1_M2 =0
	FC2_M2 =0
	FC3_M2 =0

ESTE EJERCICIO ES DE TIPO MIXTO.

ES IRRELEVANTE SI CONTESTA A LA PREGUNTA DE TEST O NO. SIN EMBARGO, SE DEBE ESCANEAR DICHA HOJA JUNTO CON EL RESTO DE LA CONTESTACIÓN DEL EXAMEN. EL ALUMNO PUEDE QUEDARSE CON EL ENUNCIADO.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Supóngase que una organización decide afrontar, por primera vez, un proyecto software en el que no tiene experiencia y que se sitúa en un ámbito desconocido para ella. Desde el punto de vista del ciclo de vida, indique qué alternativas tiene esta organización para culminar el proyecto con éxito y explique cómo pueden, dichas alternativas, mitigar los problemas potenciales con los que la organización se puede encontrar durante el desarrollo.

Solución

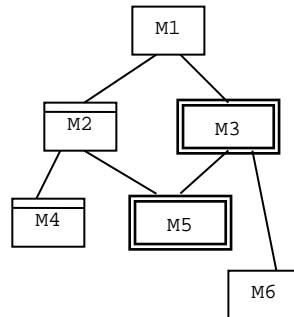
Los ciclos de vida clásicos definen un marco de trabajo controlado, de alta visibilidad sobre los procesos de desarrollo y un *estilo* de elaboración, en las diferentes fases, muy estable y repetitivo. Se denominan clásicos porque son los modelos de trabajo que se han venido empleando desde antes del origen de la Ingeniería de Software y porque están muy próximos a la manera *trivial* de elaborar cualquier producto. El ciclo de vida en cascada y en V es adecuado para la mayoría de los desarrollos cuya elaboración es suficientemente conocida y resulta especialmente cómodo para aquellos equipos de desarrollo u organizaciones que, por tener experiencia en la elaboración de un tipo de proyectos concreto, poseen procedimientos, plantillas o patrones de trabajo ajustados al desarrollo de dicho producto y que han demostrado sus ventajas en el éxito de anteriores proyectos de ese tipo. Sin embargo, estos modelos –ciclos de vida— se caracterizan, también, por su rigidez; ya que limitan el margen de maniobra ante circunstancias tales como los replanteamientos o la adecuación de las formas de trabajo descritas anteriormente. Las características de controlabilidad y visibilidad del desarrollo, de estos tipos de ciclo de vida, los hacen adecuados, también, para proyectos de gran envergadura en los que sea necesario tener un control y hacer un seguimiento cercano de sus partes. En las condiciones que se muestran en el enunciado, no parece aconsejable utilizar este modelo de trabajo.

El uso de modelos reducidos, maquetas y prototipos; su uso en simulación, análisis y pruebas para obtener experiencia o conclusiones que posibiliten paliar las incertidumbres que puedan existir en algún paso de la fabricación; ha sido una de las primeras y más notables contribuciones de la ingeniería al desarrollo de software. Actualmente existe un dinamismo frenético, no sólo en las herramientas, plataformas y tecnologías de desarrollo, sino en los entornos y tecnologías de ejecución y uso de los productos y servicios informáticos en sí. Por otro lado, el mercado demanda la necesidad de disminuir, drásticamente, los tiempos de desarrollo. El imperativo de la agilidad y la adaptabilidad, es la razón principal de que los ciclos de vida evolutivos y basados en prototipos, sean los más usados en los desarrollos actuales. Las ventajas del tratamiento eficaz de la incertidumbre y de la provisión de agilidad a los desarrollos, se contraponen a la disminución del control y de la visibilidad del proceso de fabricación, en relación con los ciclos de vida clásicos. En el escenario planteado en el enunciado, éste tipo de ciclo de vida parece el más adecuado.

El ciclo de vida en espiral se sitúa en un plano totalmente distinto respecto a los modelos anteriores; debido al hecho de que incorpora modelos de gestión al propio desarrollo. En este ciclo de vida, el modelo del

proceso de fabricación –que puede ser cualquiera de los ciclos de vida vistos anteriormente, en cada vuelta— está controlado a través de una gestión del riesgo, el cual acompaña todo el desarrollo del producto. Desde el punto de vista del enunciado, este modelo de trabajo no aporta nada nuevo respecto a lo argumentado anteriormente. La gestión del riesgo aumenta la controlabilidad entre etapas –sobre todo porque el análisis de riesgos genera una planificación que se coordina e integra en el Plan General del proyecto— aunque el incremento en la seguridad y estabilidad del desarrollo disminuya, quizás, su agilidad.

2. Dado el siguiente Diagrama de Abstracciones:

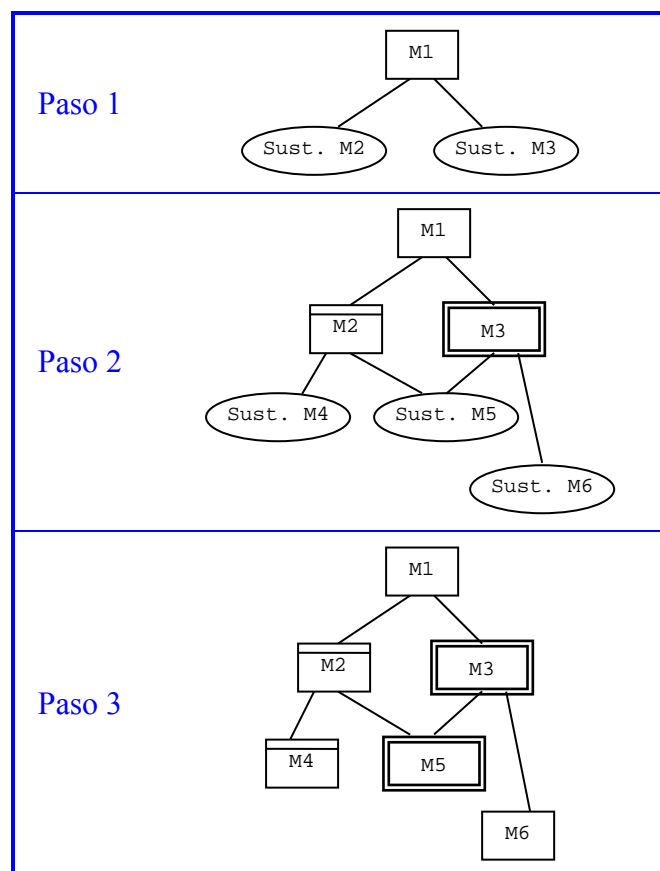


Establezca una comparativa entre las estrategias de integración ascendente y descendente, indicando el orden en el que se construirían los módulos del diagrama para cada una de ellas, así como sus ventajas e inconvenientes.

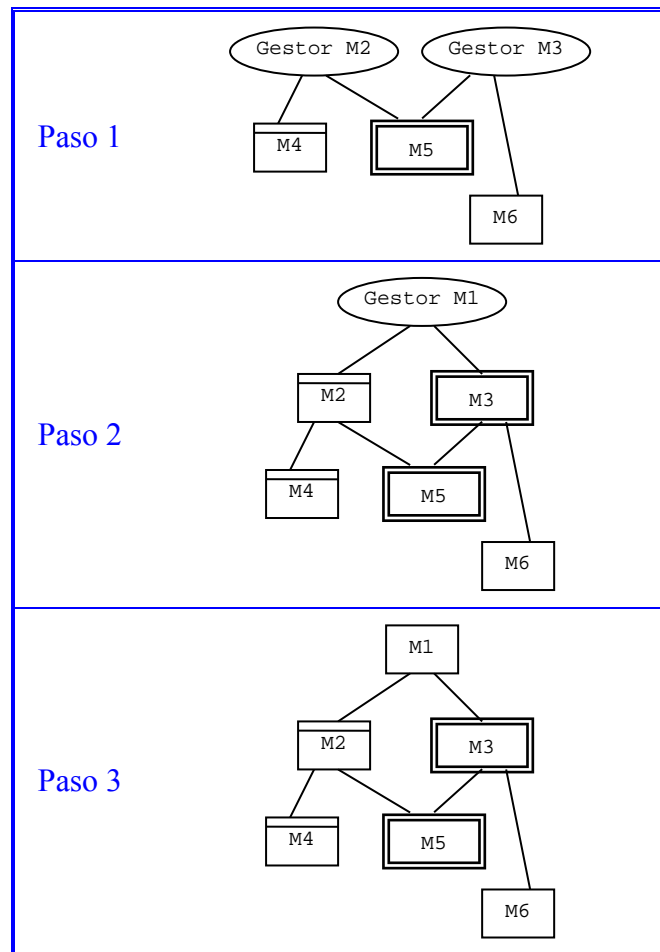
Solución

Examinemos el orden de realización de los módulos para cada estrategia de integración:

Integración Descendente



Integración Ascendente



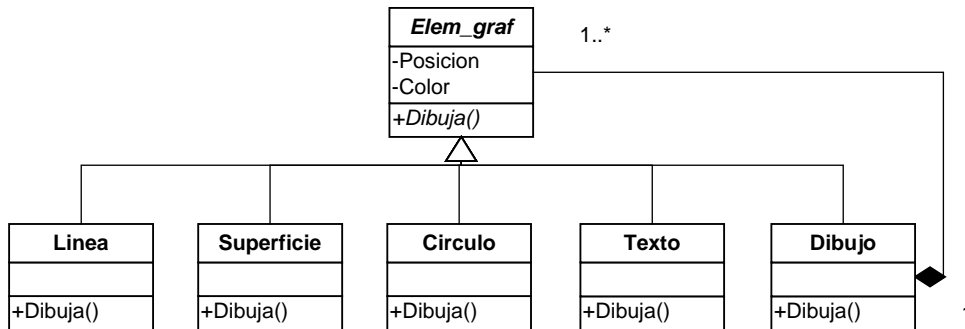
La siguiente tabla resume las ventajas e inconvenientes derivados del uso de las integraciones descendente y ascendente en el ejemplo que nos ocupa:

	I. Descendente	I. Ascendente
¿Facilita una visión general de la aplicación desde el principio?	Sí	No
Número de elementos de código desechables a construir	5 sustitutos o stubs	3 gestores o drivers
¿Facilita el ensayo de situaciones especiales para los módulos?	No	Sí
¿Facilita el trabajo en paralelo?*	Sí	Sí

(*): Aunque generalmente la integración ascendente propicia en mayor grado el trabajo en paralelo que la integración descendente, en este ejemplo, las dos estrategias de integración facilitan el trabajo en paralelo en el mismo grado.

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

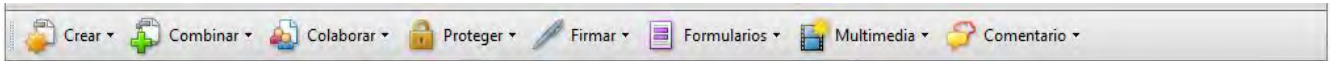
3. El diagrama de objetos de la figura es una solución que resuelve no sólo un problema específico de diseño, sino una buena cantidad de otros problemas similares. A eso se le denomina **patrón de diseño**.



El patrón anterior se denomina genéricamente **Contenedor** y corresponde, en este ejemplo, a la clase **Dibujo** de la figura.

- a. Supóngase que la clase **Decorador** es una especialización de **Dibujo** y la instancia del objeto `Mi_separador` es del tipo **Decorador**. La aplicación utilizaría el método `Mi_separador.Dibuja()`. **Explique qué ventajas tiene el uso de este patrón y de qué manera funciona como para justificar su uso extenso.**

Supóngase un entorno gráfico de ventanas como Windows donde los elementos que se manejan (**Objeto_entorno**) son una serie de elementos simples: Borde, Icono, Fondo, Etiqueta o Decorador; y otros **Elem_compuesto** son agrupaciones de los simples y de sí mismos, como Ventana, Cuadro, Marco, Barra, Menu o Boton. Así, una Barra de Herramientas (**Barra_herr**) es una especialización de Barra y está compuesto por un Borde y un Fondo, por Botones y Etiquetas, por Menus desplegables, por Separadores y un Asa (estos últimos, los consideramos especializaciones de Decorador). Por ejemplo:



- b. Haciendo que el patrón de diseño '**Contenedor**' sea la clase **Elem_compuesto**, construya el diagrama de objetos correspondiente al entorno y la clase **Barra_herr**. Si se añade un botón de '**Zoom**' (**Boton_Zoom**) ¿Cómo cambia el diagrama?

Solución

El hecho de *conectar* la invocación de un método con el código que le corresponde, se llama **acoplamiento** o **enlace**. Cuando el compilador conoce exactamente qué código le corresponde a la invocación de un método, la asignación se denomina *enlazado temprano* (*early binding*) y se produce durante la compilación, antes de la ejecución. Sin embargo, si durante la compilación no se sabe qué código le corresponde exactamente en la invocación de un método, la asignación se resuelve durante la ejecución del programa, cuando las referencias toman un valor concreto. A esto se le llama *acoplamiento tardío* (*late binding*) o **enlazado dinámico**.

- a. El patrón Contenedor, **Dibujo**, sobrecarga los métodos de **Elem_graf**. Si se crea una instancia de tipo **Dibujo**, por ejemplo así:

```
Elem_graf Mi_Dibujo = new Dibujo()
```

se crea un objeto de tipo **Dibujo** e, inmediatamente, la referencia obtenida se asigna a un objeto, **Mi_Dibujo**, de tipo **Elem_graf**. El compilador no daría error puesto que, por la herencia, un **Dibujo** es, también, un **Elem_graf**. A esto se le llama **generalización** y es una transformación del tipo hacia arriba (*upcasting*) que se hace durante la compilación (enlazado temprano).

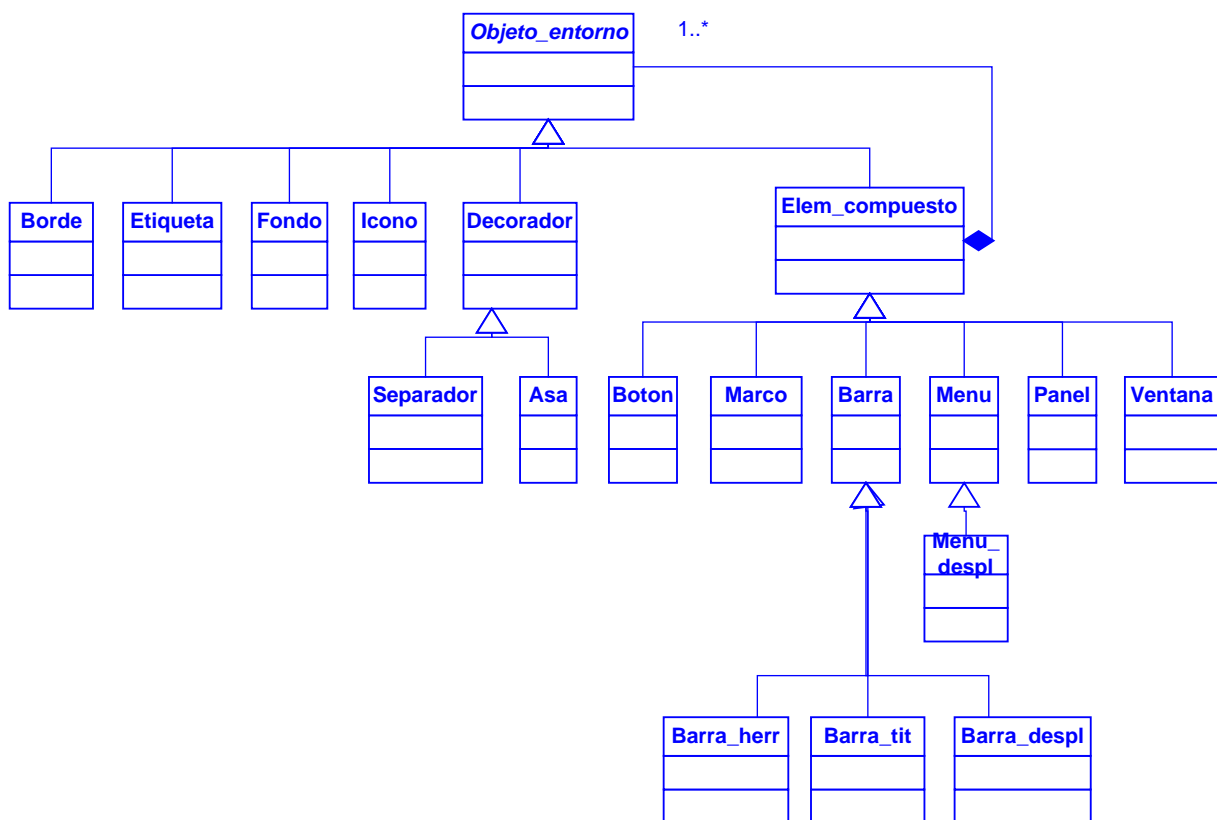
En el caso del patrón de composición, **Dibujo**, no sólo es un **Elem_graf** sino que, por la composición, puede estar compuesto por él mismo o cualquiera de sus hermanos y sus descendientes.

En el ejemplo, **Mi_Separador** es de tipo **Decorador** y, por la herencia, también es un **Dibujo** y un **Elem_graf**. Mediante la generalización, se le asignará —en tiempo de compilación— el tipo conocido del antecesor más cercano: **Dibujo** o **Elem_graf**, según haya sido la sentencia de creación.

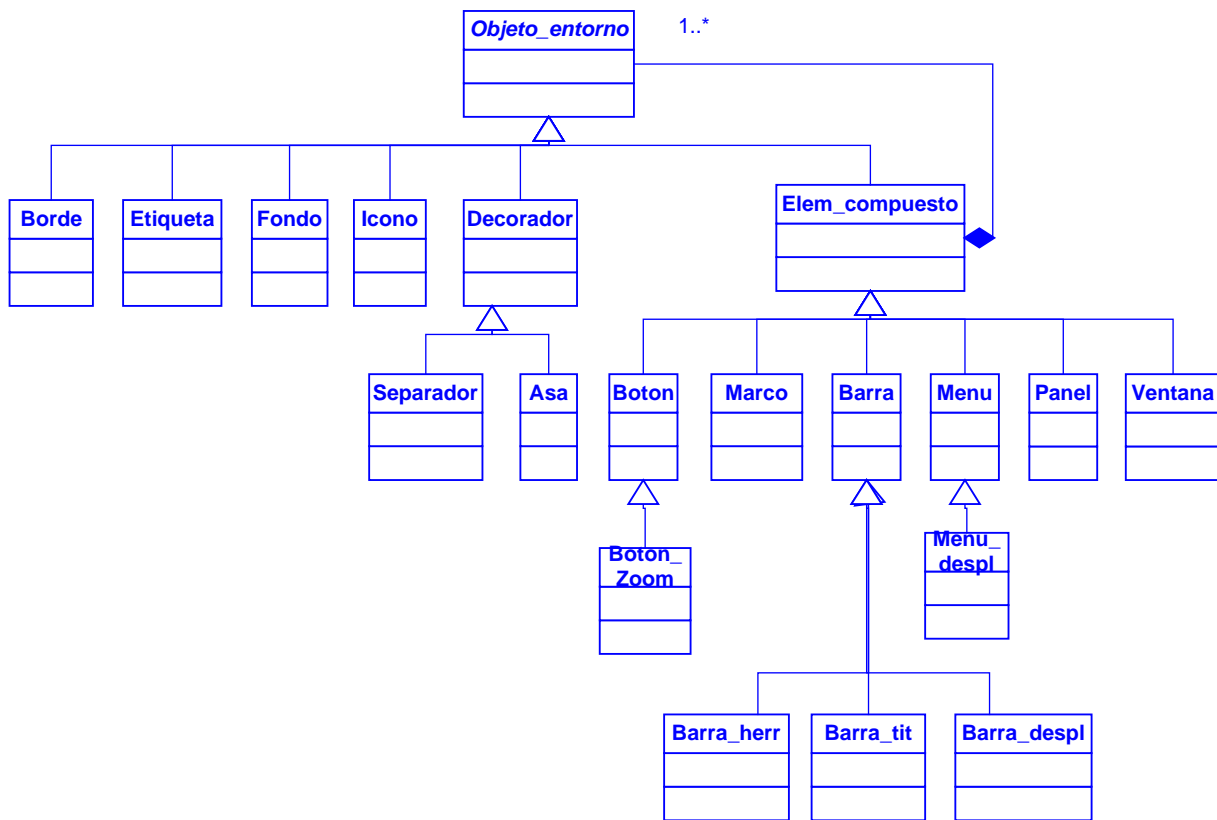
Al invocar, **durante la ejecución**, al método **Mi_Separador.Dibuja()**, el **polimorfismo** —o enlazado dinámico— garantiza que el método que se ejecuta en la invocación es la especialización correcta de **Decorador** —**Decorador.Dibuja()**— puesto que ese es el tipo de **Mi_Separador**.

Se pone de manifiesto de qué manera, la herencia y el polimorfismo, permiten diseñar con ocultación, ahorrar en el desarrollo de código —el cual resulta compacto, sencillo y fácilmente entendible— y lleva a diseños flexibles en los que es fácil añadir o eliminar elementos. El uso del patrón de composición tiene el valor añadido de que permite crear estructuras de cualquier profundidad, las cuales pueden ser perfectamente ignoradas para el código que las utilice —puesto que es suficiente con conocer la clase raíz, denominada *interfaz*— y, además, dicho código no se ve afectado porque se añadan elementos en la familia.

- b. El diagrama del entorno, con la clase **Barra_herr**, quedaría:



Si se añade la clase **Boton_Zoom**, quedaría:



ESTE EJERCICIO ES DE TIPO MIXTO.

ES IRRELEVANTE SI CONTESTA A LA PREGUNTA DE TEST O NO. SIN EMBARGO, SE DEBE ESCANEAR DICHA HOJA JUNTO CON EL RESTO DE LA CONTESTACIÓN DEL EXAMEN. EL ALUMNO PUEDE QUEDARSE CON EL ENUNCIADO.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Supóngase que una organización decide afrontar, por primera vez, un proyecto software en el que no tiene experiencia y que se sitúa en un ámbito desconocido para ella. Desde el punto de vista del ciclo de vida, indique qué alternativas tiene esta organización para culminar el proyecto con éxito y explique cómo pueden, dichas alternativas, mitigar los problemas potenciales con los que la organización se puede encontrar durante el desarrollo.

Solución

Los ciclos de vida clásicos definen un marco de trabajo controlado, de alta visibilidad sobre los procesos de desarrollo y un *estilo* de elaboración, en las diferentes fases, muy estable y repetitivo. Se denominan clásicos porque son los modelos de trabajo que se han venido empleando desde antes del origen de la Ingeniería de Software y porque están muy próximos a la manera *trivial* de elaborar cualquier producto. El ciclo de vida en cascada y en V es adecuado para la mayoría de los desarrollos cuya elaboración es suficientemente conocida y resulta especialmente cómodo para aquellos equipos de desarrollo u organizaciones que, por tener experiencia en la elaboración de un tipo de proyectos concreto, poseen procedimientos, plantillas o patrones de trabajo ajustados al desarrollo de dicho producto y que han demostrado sus ventajas en el éxito de anteriores proyectos de ese tipo. Sin embargo, estos modelos –ciclos de vida– se caracterizan, también, por su rigidez; ya que limitan el margen de maniobra ante circunstancias tales como los replanteamientos o la adecuación de las formas de trabajo descritas anteriormente. Las características de controlabilidad y visibilidad del desarrollo, de estos tipos de ciclo de vida, los hacen adecuados, también, para proyectos de gran envergadura en los que sea necesario tener un control y hacer un seguimiento cercano de sus partes. En las condiciones que se muestran en el enunciado, no parece aconsejable utilizar este modelo de trabajo.

El uso de modelos reducidos, maquetas y prototipos; su uso en simulación, análisis y pruebas para obtener experiencia o conclusiones que posibiliten paliar las incertidumbres que puedan existir en algún paso de la fabricación; ha sido una de las primeras y más notables contribuciones de la ingeniería al desarrollo de software. Actualmente existe un dinamismo frenético, no sólo en las herramientas, plataformas y tecnologías de desarrollo, sino en los entornos y tecnologías de ejecución y uso de los productos y servicios informáticos en sí. Por otro lado, el mercado demanda la necesidad de disminuir, drásticamente, los tiempos de desarrollo. El imperativo de la agilidad y la adaptabilidad, es la razón principal de que los ciclos de vida evolutivos y basados en prototipos, sean los más usados en los desarrollos actuales. Las ventajas del tratamiento eficaz de la incertidumbre y de la provisión de agilidad a los desarrollos, se contraponen a la disminución del control y de la visibilidad del proceso de fabricación, en relación con los ciclos de vida clásicos. En el escenario planteado en el enunciado, éste tipo de ciclo de vida parece el más adecuado.

El ciclo de vida en espiral se sitúa en un plano totalmente distinto respecto a los modelos anteriores; debido al hecho de que incorpora modelos de gestión al propio desarrollo. En este ciclo de vida, el modelo del proceso de fabricación –que puede ser cualquiera de los ciclos de vida vistos anteriormente, en cada

vuelta— está controlado a través de una gestión del riesgo, el cual acompaña todo el desarrollo del producto. Desde el punto de vista del enunciado, este modelo de trabajo no aporta nada nuevo respecto a lo argumentado anteriormente. La gestión del riesgo aumenta la controlabilidad entre etapas –sobre todo porque el análisis de riesgos genera una planificación que se coordina e integra en el Plan General del proyecto— aunque el incremento en la seguridad y estabilidad del desarrollo disminuya, quizás, su agilidad.

2. ¿Qué tres objetivos fundamentales o cualidades mínimas es deseable alcanzar al hacer la descomposición modular de un sistema? Explique cada uno de ellos y, en cada caso, cómo se pueden medir o qué factores intervienen.

Solución

Independencia funcional, comprensibilidad y adaptabilidad.

Síntesis y resumen esquemático del epígrafe 4.1. Páginas 148 a 160 del libro de texto.

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Se desea construir un sistema que reciba como entrada una lista ordenada de líneas, cada una de las cuales está formada por una lista ordenada de palabras, que a su vez será una lista ordenada de caracteres. Sobre cada línea se pueden realizar rotaciones, que consisten en eliminar la primera palabra y concatenarla al final de la línea. El sistema devolverá como resultado una lista con las posibles rotaciones de todas las líneas ordenadas alfabéticamente (incluyendo las rotaciones nulas).

Por ejemplo,

<u>voy a cenar</u> <u>al restaurante</u>	→	<u>a cenar voy</u> <u>al restaurante</u> <u>cenar voy a</u> <u>restaurante al</u> <u>voy a cenar</u>
---	---	--

- a. Diseñe el sistema utilizando herencia, polimorfismo y Diagramas de Objetos.
- b. ¿Cómo cambiaría su diseño si el sistema recibe una línea –lista ordenada de palabras—, realiza rotaciones sobre cada palabra –va rotando los caracteres circularmente— y devuelve como resultado una lista con las posibles rotaciones de todas las palabras ordenadas alfabéticamente (incluyendo las rotaciones nulas)?

Por ejemplo,

<u>voy a cenar</u>	→	<u>a</u> <u>arcen</u> <u>cenar</u> <u>enarc</u> <u>narce</u> <u>oyv</u> <u>rcena</u> <u>voy</u> <u>yvo</u>
--------------------	---	--

Solución

En una primera aproximación, aplicaremos el método de Abbott para determinar las clases u objetos que componen el diseño. Para ello, marcaremos en rojo los sustantivos (candidatos a ser clases o atributos) y en azul los verbos (candidatos a ser métodos).

Se desea construir un sistema que reciba como entrada una *lista ordenada de líneas*, cada una de las cuales está formada por una *lista ordenada de palabras*, que a su vez será una *lista ordenada de caracteres*. Sobre cada línea se pueden *realizar rotaciones*, que consisten en *eliminar la primera palabra* y *concatenarla al final de la línea*. El sistema devolverá como resultado una *lista con las posibles rotaciones de todas las líneas ordenadas alfabéticamente* (incluyendo las rotaciones nulas).

A partir de este marcado elaboramos una doble lista con los elementos correspondientes a clases y a métodos.

CLASES O ATRIBUTOS	MÉTODOS
<ul style="list-style-type: none">• lista ordenada de líneas• lista ordenada de palabras• lista ordenada de caracteres• lista con las posibles rotaciones de	<ul style="list-style-type: none">• realizar rotaciones• eliminar la primera palabra• concatenar la primera palabra al final de la línea

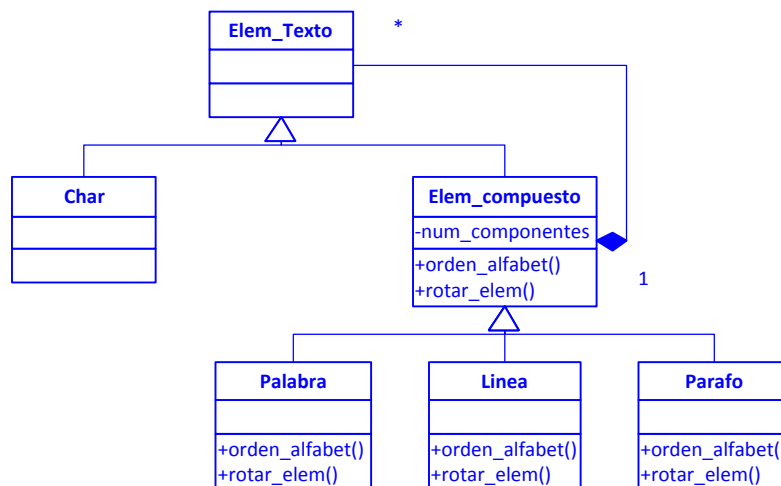
todas las líneas ordenadas
alfabéticamente



Cuando piense en cómo asignar una clase a una estructura de datos y en cómo distribuir la herencia o el polimorfismo, considere que una clase **es una definición del tipo de un dato**. Decir ‘Mirlo’ hereda de ‘Ave’ significa **que son del mismo tipo** y la relación se enuncia como: un ‘Mirlo’ **‘es un’** ‘Ave’. La especialización —y algunas formas de polimorfismo— también se deben a la herencia. Un ‘Avestruz’ **‘es como un’** ‘Mirlo’... Es pues un ‘Ave’ —tiene el mismo tipo— pero ha anulado el método de volar y tiene sobrecargadas otras funciones de ‘Ave’.

Del análisis del cuadro anterior, obtenido con el método de Abbott, parece claro que se van a manejar listas ordenadas cuyos componentes son elementos de texto. El objetivo de la aplicación es realizar determinadas operaciones con lo que hemos denominado ‘listas ordenadas’. Ahora bien, cada elemento que se maneja en una lista es, a su vez, otra lista; a no ser que el elemento sea un carácter. Además, para ahorrar código y simplificar el diseño, sería conveniente que el código escrito para una operación en una ‘lista ordenada’ —por ejemplo, ‘ORDENAR ALFABÉTICAMENTE LAS LÍNEAS DE ENTRADA’— sirva igualmente para otras listas. Al fin y al cabo, ¡todos los elementos de una lista son elementos de texto! Éste es el elemento crucial: encontrar el *tipo común*.

El diseño podría ser así:



ESTE EJERCICIO ES DE TIPO MIXTO.

ES IRRELEVANTE SI CONTESTA EN LA CASILLA DE TEST O NO (lo que ponga, NO CUENTA para la calificación). SIN EMBARGO, SE DEBE ESCANEAR DICHA HOJA JUNTO CON EL RESTO DE LA CONTESTACIÓN DEL EXAMEN.

EL ALUMNO PUEDE QUEDARSE CON EL ENUNCIADO.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Defina y distinga la 'validación' y la 'verificación'. ¿En qué fase del ciclo de vida en cascada se realiza cada una?

Solución

Libro de texto, epígrafe 1.4.1.2, páginas 15 y 16.

Validación: la comprobación de que un elemento (o todo el sistema) satisface las necesidades del usuario identificadas durante el análisis.

Verificación: la comprobación de que una parte (módulo o unidad) del sistema cumple con sus especificaciones particulares (de funcionamiento).

Evidentemente, estas comprobaciones se deberían hacer siempre; independientemente del ciclo de vida que se esté utilizando. La validación se hará después de la integración y pruebas de una parte funcionalmente significativa del sistema o del sistema completo, antes de su entrega y aceptación por parte del cliente. Por el contrario, en la verificación se comprueba que una unidad funcional se ejecuta correcta y coherentemente con las especificaciones de funcionamiento que se han establecido en el diseño para esa parte de la aplicación. Por tanto, la verificación se confunde con las '*pruebas de unidades*' y se hace tras la codificación o, si se está comprobando un módulo, tras la integración de sus componentes.

2. Defina los siguientes tipos de pruebas de software: *caja negra*, *caja transparente*, *alfa* y *beta*. ¿Las pruebas *alfa* y *beta* son de caja negra o transparente? Y las pruebas de unidades ¿de qué tipo son y cuándo se hacen?

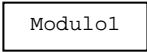

Solución

Epígrafes 5.7.2, 5.7.3 y 5.9.2 del libro de texto.

Las pruebas *alfa* y *beta* son pruebas del sistema completo, integrado. Pueden formar parte de las comprobaciones establecidas para la validación. Son de caja negra. Para las pruebas de unidades se puede seguir cualquier técnica de las reflejadas en los epígrafes de 5.7: de caja negra o de caja transparente. Estas pruebas constituyen la *verificación* y se hacen tras la codificación o, si se está comprobando un módulo, tras la integración de sus componentes.

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Dentro de un sistema informático se emplea el módulo `Modulo1`. Para su desarrollo se plantean dos diseños alternativos:

Diseño 1	Diseño 2
	
<div>Código aproximado en Modula-2</div> <pre>1MODULE Modulo1; 2FROM InOut 3 IMPORT WriteCard, WriteString; 4... 5VAR 6 fecha1, fecha2: 7 ARRAY [1..6] OF TipoDigito; 8 ... 9BEGIN 10 ... 11 (* fecha1 := 30-04-1974 *) 12 fecha1[1] := 3; 13 fecha1[2] := 0; 14 fecha1[3] := 0; 15 fecha1[4] := 4; 16 fecha1[5] := 7; 17 fecha1[6] := 4; 18 (* Imprimir anno *) 19 WriteCard(fecha1[5], 1); 20 WriteCard(fecha1[6], 1); 21 ... 22 (* fecha2 := 10-01-1998 *) 23 fecha2[1] := 1; 24 fecha2[2] := 0; 25 fecha2[3] := 0; 26 fecha2[4] := 1; 27 fecha2[5] := 9; 28 fecha2[6] := 8; 29 ... 30END Modulo1.</pre>	<pre>1MODULE Modulo1; 2IMPORT Fecha; 3... 4VAR 5 fecha1, fecha2: Fecha.Tipo; 6 ... 7BEGIN 8 ... 9 Fecha.Crear(fecha1, 30, 4, 1974); 10 Fecha.ImprimirAnno(fecha1); 11 ... 12 Fecha.Crear(fecha2, 10, 1, 1998); 13 ... 14END Modulo1. 1DEFINITION MODULE Fecha; 2TYPE Tipo; 3PROCEDURE Crear(VAR fecha: Tipo); 4PROCEDURE ImprimirAnno(fecha: Tipo); 5... 6END Fecha.</pre>

Compare los dos diseños analizando cómo aplican los conceptos de Abstracción, Modularidad y Ocultación.

Inicialmente, cuando se planteó el sistema informático, se consideró que para el almacenamiento de los años bastaba con dos dígitos. Sin embargo, con la llegada el nuevo milenio se descubrió que eran necesarios cuatro dígitos. **Razone cómo afectaría este cambio a cada uno de los diseños.**

Solución

1. Abstracción

- El primer diseño no utiliza abstracciones de ningún tipo.
- El diseño 2 utiliza el Tipo Abstracto de Datos¹ `Fecha`.

Como resultado, se puede observar que el código asociado al diseño 1 será muy redundante (líneas 12-17 ≈ líneas 23-28). La redundancia induce a errores e inconsistencias.

¹Concretamente y según la nomenclatura estudiada en la asignatura Programación 1, el *Tipo Opaco de Datos* `Fecha`.

2. Modularidad

El diseño 1 es monolítico, mientras que el diseño 2 es modular. Por ello, el diseño 2 dispone de las siguientes ventajas sobre el diseño 1:

- Permite dividir la implementación entre varias personas (un programador puede codificar `Modulo1` y otro, `Fecha`)
- La implementación asociada al diseño 2 es clara y concisa.
- Los costes asociados al desarrollo, la depuración, la documentación y el mantenimiento del diseño 2 son menores que los del diseño 1.
- El diseño 2 permite reutilizar el concepto de fecha en otros proyectos.

3. Ocultación

En el diseño 1 las “interioridades de las fechas están al descubierto”, es decir, no existe ocultación del concepto de fecha. Este hecho, conlleva dos grandes problemas:

- Si se produce un error en el uso de una fecha, su detección será ardua, ya que a este concepto se accede directamente desde muchos puntos del programa.
- La modificación de la representación del concepto fecha se propagará a muchas partes del código del `Modulo1`. Si tal y como se propone en el enunciado, se decide extender la representación de los años de 2 dígitos a 4, será necesario modificar las líneas 7, 12-28.

En el diseño 2, el concepto de fecha se encapsula y oculta a través de un Tipo Abstracto de Datos. Las consecuencias benignas de esta estrategia son:

- Si se produce un error asociado a una fecha, la búsqueda se limitará al módulo de implementación de `Fecha`.
- Gracias a la ocultación del concepto fecha, la ampliación del número de dígitos de los años implica cambios exclusivamente en el código del módulo de implementación de `Fecha`.

ESTE EJERCICIO ES DE TIPO MIXTO.

ES IRRELEVANTE SI CONTESTA EN LA CASILLA DE TEST O NO (lo que ponga, NO CUENTA para la calificación). SIN EMBARGO, SE DEBE ESCANEAR DICHA HOJA JUNTO CON EL RESTO DE LA CONTESTACIÓN DEL EXAMEN.

EL ALUMNO PUEDE QUEDARSE CON EL ENUNCIADO.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Suponga que se quiere modelar un sistema en el que lo más importante es representar las tareas y transformaciones que se hacen con la información y los demás aspectos son prácticamente irrelevantes. ¿Sería imprescindible el uso de Diagramas Entidad-Relación? ¿Qué aspecto del modelo prioriza esta notación?

Solución

Epígrafes 2.3.2, 2.3.3 y 2.3.6 del libro. Páginas 52 a 60 y 65 a 68.

Los tres tipos diagramas para representar de modelos del comportamiento del sistema, que se ven en la asignatura, son complementarios y cuasi-ortogonales. Cada uno de ellos hace hincapié en un grupo de aspectos del modelo, diferenciado de los otros. De esta forma, para que un sistema esté completa o suficientemente descrito, necesitaría de todas las vistas. Ahora bien, el objetivo principal de un modelo es que facilite la comprensión del comportamiento del sistema que se va a construir; y lo haga hasta tal punto que se pueda diseñar el funcionamiento con una incertidumbre mínima y, después, la implementación del diseño lleve al comportamiento deseado. Por tanto, el grado de desarrollo del modelo, dependerá de la complejidad del sistema y de los aspectos que aún no se comprenden o no están definidos con claridad. En este caso, para representar las tareas y transformaciones que se hacen con la información, lo más apropiado es utilizar Diagramas de Flujo de Datos. Si, con esta representación, se alcanza la comprensión del comportamiento que se espera del producto y los demás aspectos son, realmente, irrelevantes, se podría prescindir de DTE y DER.

Los Diagramas Entidad-Relación son modelos de datos que se centran en los aspectos relacionados con los '*objetos*', '*artefactos*' o elementos de información –las entidades— y en las vinculaciones relevantes que existen entre ellos –las relaciones—; pero no en cómo se manejan ni cómo se controla el funcionamiento del sistema ni cuál es su evolución dentro de él.

2. ¿Qué tres objetivos fundamentales o cualidades mínimas es deseable alcanzar al hacer la descomposición modular de un sistema? Explique cada uno de ellos y, en cada caso, cómo se pueden medir o qué factores intervienen.

Solución

Introducción de la sección “4.1 Descomposición modular” y epígrafes 4.1.1 a 4.1.3. Páginas 149 a 160 del libro de la asignatura.

Realizar la descripción del funcionamiento del sistema descomponiéndolo en módulos, tiene como objetivo fundamental facilitar la implementación y que, tras la codificación, disminuyan los problemas de funcionamiento. El diseño *‘hace funcionar’* al producto porque define sus mecanismos de funcionamiento, que se codificarán a continuación. Pero las decisiones relativas a **qué** diseño se utiliza, qué descomposición modular se elige, están orientadas a facilitar el mantenimiento, inmediato o futuro. Que el producto sea más o menos mantenible está estrechamente ligado a los costes de producción y a la rentabilidad que se espera al comercializarlo.

Para incrementar la mantenibilidad del software, la experiencia acumulada en desarrollos anteriores aconseja que la descomposición modular elegida cumpla con unos niveles mínimos de:

1. Independencia funcional. (Qué es y cómo se mide. Síntesis de las páginas 150 a 157 del libro)
2. Comprensibilidad.
3. Adaptabilidad

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Con el fin de promocionar el uso del transporte público y el ocio al aire libre, RENFE ha decidido encargar la construcción de un sistema informático que asesore a sus clientes acerca de *‘rutas verdes’* para hacer a pie a partir de sus estaciones de tren.

El sistema recibirá periódicamente la siguiente información:

- Un informe meteorológico del Instituto Nacional de Meteorología que contendrá las previsiones climáticas para los próximos días.
- Datos referentes a las estaciones de tren, horarios y precios de billetes. Esta información será suministrada por RENFE.
- Se ha encargado a la empresa “Viajes Najarra” la elaboración e introducción en el sistema de las rutas verdes. Para ello, la empresa podrá solicitar del sistema un informe de las estaciones de RENFE existentes.

Los clientes introducirán en el sistema sus preferencias. A partir de estas y los datos antes descritos, se construirá un informe con las rutas aconsejadas.

Analice el sistema mediante DFDs (Diagramas de Flujo de Datos), desarrollando exclusivamente los DFDs de nivel 0 (contexto) y nivel 1.

Solución

