



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN  
NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

PRIMERA SEMANA

PREGUNTA 1. Tiempo estimado 50 minutos PUNTUACIÓN 4 puntos

Una empresa de alquiler de automóviles tiene a su disposición un conjunto de vehículos indicados en la siguiente tabla.

Se quiere diseñar e implementar un programa que almacene y gestione la información relacionada con estos vehículos.

Se pide:

Tipo de vehículo	Características
Motos	marca, matrícula, número de identificación, número de kilómetros, estado actual de depósito de gasolina.
Coches (turismos)	marca, matrícula, número de identificación, tipo (normal / familiar), número de puertas, número de kilómetros, tipo de motor (gasolina / gasoil), estado actual del depósito.
Coches (deportivos)	marca, matrícula, número de identificación, capacidad de motor, número de kilómetros, turbo o no, número de puertas, número de asientos, estado actual del depósito de gasolina.
Coches (4x4)	marca, matrícula, número de identificación, número de kilómetros, tipo de motor (gasolina / gasoil), número de asientos, estado actual de depósito.
Monovolumenes	marca, matrícula, número de identificación, número de kilómetros, número de puertas, puertas laterales, número de asientos, tipo de motor (gasolina / gasoil), capacidad de carga, estado actual del depósito.
Furgonetas	marca, matrícula, número de identificación, número de kilómetros, capacidad de carga, altura, estado actual del depósito de gasoil.

- 1) Identificar la estructura y las relaciones de herencia y de uso de las clases necesarias para almacenar y gestionar esta información. Dibujar un esquema de la organización de estas clases en el diseño global. (1,5 punto)
- 2) Implementar la especificación de las clases. (1 punto)
- 3) Se quiere declarar un array de objetos para almacenar todos los vehículos, independientemente del tipo.
  - ¿Cómo declararías el array? (0,5 puntos)
  - ¿Qué métodos se necesitan para acceder a un vehículo concreto? (0,5 puntos)
  - ¿Cómo se almacenan las diferencias entre los distintos tipos de vehículos? (0,5 puntos)

**SOLUCIÓN:**

1. Dados las características de los vehículos, se pueden agrupar los datos para ver los siguientes elementos comunes:

	Motos	Coches (turismos)	Coches (deportivos)	Coches (4x4)	Monovolumenes	Furgonetas
Datos comunes	marca, matrícula, número de identificación, número de kilómetros	marca, matrícula, número de identificación, número de kilómetros	marca, matrícula, número de identificación, número de kilómetros	marca, matrícula, número de identificación, número de kilómetros	marca, matrícula, número de identificación, número de kilómetros	marca, matrícula, número de identificación, número de kilómetros
		estado actual del depósito		estado actual del depósito	estado actual del depósito	
		tipo de motor (gasolina / gasoil)		tipo de motor (gasolina / gasoil)	tipo de motor (gasolina / gasoil)	
			número de puertas		número de puertas	
			número de asientos		número de asientos	
Datos específicos	estado actual del depósito de gasolina	número de puertas tipo (normal / familiar)	estado actual del depósito de gasolina, capacidad de motor, turbo o no	número de asientos	puertas laterales, capacidad de carga	estado actual del depósito de gasoil, capacidad de carga, altura

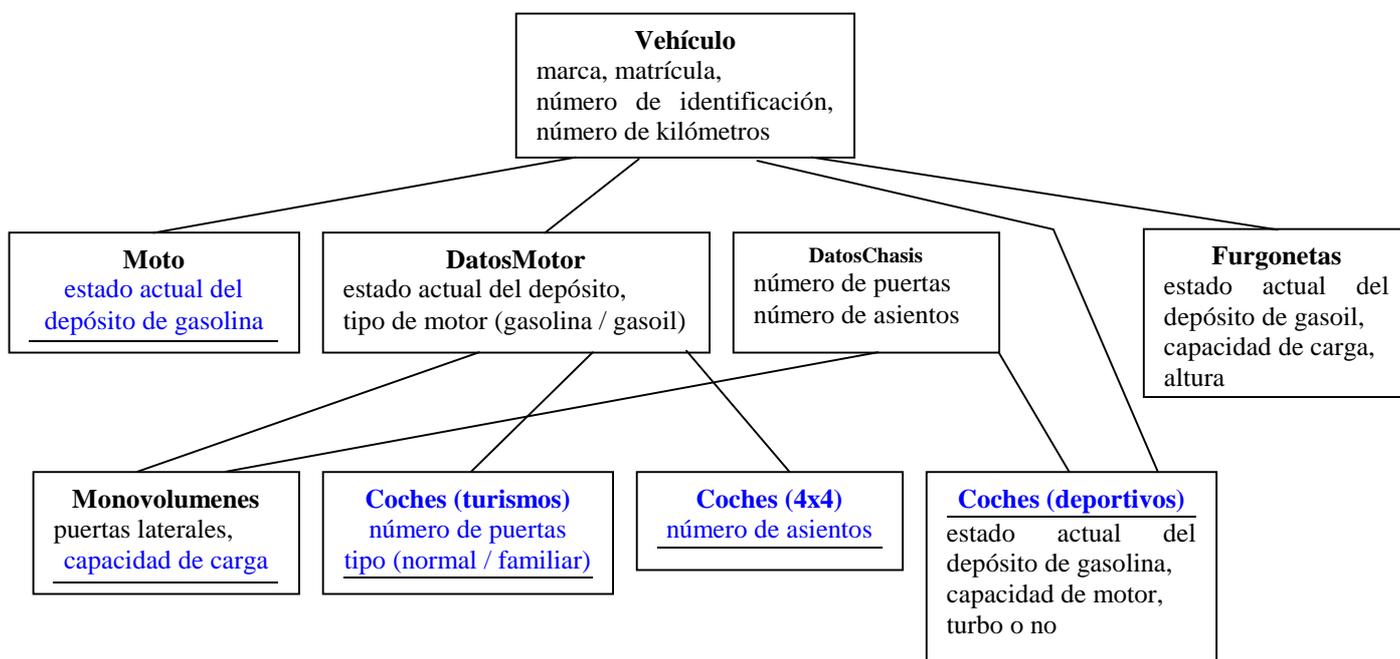
Con herencia múltiple, se tienen las siguientes relaciones de herencia:



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA UNIVERSITARIA DE INFORMÁTICA

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL



2. Se puede ver la especificación de estas clases en Java a continuación:

```
class Vehiculo {
    private String marca;
    private String matrícula;
    private int id;
    private int km;
}

class Moto extends Vehiculo {
    private int depGasolina;
}

class Furgonetas extends Vehiculo {
    private int depGasoil;
    private int capCarga;
    private int altura;
}

class DatosMotor extends Vehiculo {
    private int deposito;
    private boolean gasolina;
}

class CochesTurismos extends DatosMotor {
    private int puertas;
    private boolean normal;
}

class Coches4x4 extends DatosMotor {
    private int asientos;
}

class DatosChasis {
    private int puertas;
    private int asientos;
}

class CochesDeportivos extends DatosChasis {
    private int depGasolina;
}
```



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA UNIVERSITARIA DE INFORMÁTICA

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**  
**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

---

```
private int capMotor;
private boolean turbo;
}

//en este caso, como no se puede tener la herencia múltiple en Java
//hay que usar una versión de la clase DatosChasis
class Monovolumenes extends Vehiculo {
    DatosChasis dc;
    private int puertasLaterales;
    private int capCarga;
}
```

3.

a. Se declara el array como la clase base, `Vehiculo`, así, como todas los vehículos están basados en esta clase se puede guardar cualquier tipo de vehículo allí:

```
Vehiculo vehiculos[] = new Vehiculo[5];
vehiculos[0] = new Moto("Yamaha", "QHW1278", 123, 50, 100);
vehiculos[1] = new CocheTurismo("Renault", "nht1980", 987, 20050, 100, true, 4, true);
vehiculos[2] = new Coche4x4("Jeep", "mfr3645", 454, 1809, 58, false, 2);
vehiculos[3] = new CocheDeportivo("Jaguar", "bhy4567", 480, 23, 3, 2, 98, 4500, true);
vehiculos[4] = new Monovolumen("Citroën", "mnb0987", 788, 34500, 1, 5, 4, 8);
```

b. Se puede acceder a cada objeto dentro del array a través del número de su posición en el array. Pero para poder acceder a los datos dentro de cada objeto hay que tener o un método virtual general para devolver todos los datos (el mismo método en cada clase) o un método virtual para cada tipo de datos, como por ejemplo, para indicar el contenido del depósito de gasolina:

```
class Vehiculo {
    //código no incluido
    public int getDepGas(){}
}

class Moto extends Vehiculo {
    //código no incluido
    public int getDepGas(){
        return depGasolina;
    }
}
```

c. Se puede almacenar las diferencias entre los tipos de dos maneras:

1. Usando el constructor de cada tipo de vehículo como se ha hecho arriba.
2. Incorporando métodos virtuales para cada tipo de datos, como por ejemplo:

```
class Vehiculo {
    //código no incluido
    public void setDepGas(){}
}

class Moto extends Vehiculo {
    //código no incluido
    public void setDegGas(int g){
        depGasolina = g;
    }
}
```



## PREGUNTA 2. Tiempo estimado 20 minutos. PUNTUACIÓN 2,5 puntos

Queremos representar cadenas en un lenguaje de programación en el que no existe este tipo definido, pero sí existe el tipo CHARACTER. Se pide:

1. Proponer dos representaciones posibles para definir el tipo CADENA-CARACTERES. Indicar las ventajas e inconvenientes que ofrecen cada una de las representaciones elegidas. (1 punto)
2. Para cada una de las representaciones elegidas en el apartado anterior:
  - a. Escribir la función recursiva CONCATENA que permita encadenar dos cadenas de caracteres. Justificar el tipo de paso de parámetros utilizado en la función. (1 punto)
  - b. ¿Redefinir CONCATENA para que en vez de ser una función sea un procedimiento. Justificar el paso de parámetros utilizado y los cambios que requiera la implementación. (0,5 puntos)

### SOLUCIÓN:

1. Dos posibles representaciones, sencillas, es un ARRAY de caracteres, y la otra una lista encadenada de caracteres. Las ventajas de la primera son la derivadas del uso de arrays: la facilidad de manejo que ofrecen los arrays, como problemas están en lo que si hay posiciones no ocupadas se está desperdiciando espacio, la dificultad para añadir o quitar elementos. En la lista encadenada se desperdicia menos espacio y es más fácil añadir y borrar elementos, como desventajas están la dificultad del manejo de la estructura de datos que obliga a hacer recorridos por ellas hasta que se llega a la posición deseada.
2. Para la lista-encadenada, no hace falta recursividad basta con enlazar el final de una cadena con el principio de la otra

```
FUNCTION CONCATENA1 (C1, C2: cadena_punteros): cadena_punteros
  C1.SIGUIENTE:= C2;
  Return C1
end;
```

Para la representación en Array, la solución recursiva en un lenguaje procedural es mucho menos eficiente que la iterativa, ya que llevamos en C1 parte de la copia de C2. Inicialmente posición vale 1, que indica la primera posición de C2. Supongo que el carácter # indica fin de la cadena.

```
FUNCTION CONCATENA2 (C1, C2:cadena_tabla, ultimaPosC1, posición: integer): cadena_tabla
  If C2[posición] == '#'then return C1
  else return CONCATENA (C1[ultimaPosC1 + 1] := C2[posición, C2, ultimaPosC1 +1, posición +1);
end;
```

En ambos casos se usa paso de parámetros por valor ya que es la propia recursividad la que mantiene los valores intermedios.

La redefinición a procedimiento sería en CONCATENA1

```
FUNCTION CONCATENA1 (IN OUT C1: cadena_punteros; C2: cadena_punteros);
```

Donde IN OUT indica paso de parámetros por referencia, y C2 es paso de parámetros por valor. Para CONCATENA2, si se usa paso de parámetros por valor en la llamada hay que poner C1 y, por lo tanto, se hace la asignación antes de la llamada.

```
FUNCTION CONCATENA2 (IN OUT C1:cadena_tabla, , C2:cadena_tabla, ultimaPosC1, posición: integer):
cadena_tabla
  If C2[posición] == '#'then exit
  else
    C1[ultimaPosC1 + 1] := C2[posición];
    CONCATENA (C1, C2, ultimaPosC1 +1, posición +1);
end;
```



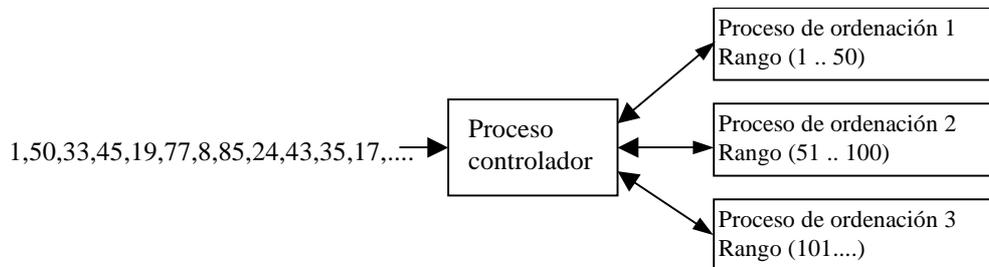
### PREGUNTA 3. Tiempo estimado 50 minutos. PUNTUACIÓN 3,5 puntos

Se desea implementar un programa que permita ordenar una secuencia de números de forma descendente. Para ello se dispone de tres tareas ORDEN que ordenan una secuencia de números. Otra tarea T divide la secuencia de números en tres subsecuencias, luego las reparte en las tres tareas ORDEN (una subsecuencia a cada una de ellas) que una vez tienen su subsecuencia ordenada pasan el resultado a la tarea T. Ésta une los tres resultados y saca la secuencia ordenada por pantalla.

La manera en que T divide la secuencia inicial en las tres subsecuencias es la siguiente:

- En la primera subsecuencia mete aquellos números que van en el rango de 1 a 50.
- En la segunda subsecuencia mete aquellos números que van en el rango de 51 a 100.
- En la tercera subsecuencia mete aquellos números que son mayores que 100

En la figura adjunta se muestra gráficamente este proceso.



- 1) Identificar las tareas necesarias para realizar este programa. Escribir la especificación de cada una de ellas justificando los tipos de paso de parámetros utilizados. (0,2 puntos)
- 2) Proponer y justificar la estructuras de datos adecuadas para poder realizar este programa (0,8 puntos)
- 3) Escribir el cuerpo de la tarea T (0,5 puntos)
- 4) Sin entrar en el algoritmo de ordenación utilizado (poner simplemente una llamada a DEVUELVE\_SECUENCIA\_ORDENADA) que realiza este trabajo escribir el cuerpo de las otras tareas. (1 punto)
- 5) Se quiere rediseñar el problema para que cada proceso de ordenación, a su vez, divida su secuencia en otras subsecuencias y utilice a su vez otros procesos de ordenación. Este proceso se repetiría hasta que la secuencia sólo tuviese dos números. Indicar los cambios que se producirían en el programa escrito en los apartados anterior para considerar esta nueva solución. (1 punto).

#### SOLUCIÓN:

1. En este programa hace falta cuatro tareas (como queda especificado en el enunciado del problema) más el programa principal.

Una tarea T para:

- dividir la secuencia de números en tres sub-secuencias y enviar los números a la tarea de ordenación correspondiente.
- recuperar y adjuntar los resultados después de ser organizadas.
- para poder realizar esto tendría que: recibir una secuencia de números, enviar cada número a la tarea de ORDENACION

Tres tareas ORDENACION idénticas para:

- ordenar un conjunto de números.

Las especificaciones para las dos tareas serían:

task T is



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

```
-- no tiene puntos de entrada
end;

task type ORDENACION is
  -- hay un punto de entrada para recibir y enviar la secuencia de números
  entry LISTA (La_lista : IN OUT Lista_Numeros);
end;
```

2. Suponemos que sabemos de antemano cuántos números tiene la secuencia y por lo tanto la guardamos en un array. Las tres subsecuencias serán de tamaño menor que la secuencia y para ellas, como sabemos que son tres, declaramos un array de tres elementos cada uno de los cuales es – a su vez – otro array. Por último, para las tareas de ordenación declaramos un array de tres tareas, ya que las tres se comportan de forma similar pero trabajando con diferentes subsecuencias de números.

Array de N números, para la secuencia de números de entrada

```
Type Lista is array (1..N) OF INTEGER;
-- el tamaño se indica en la declaración
```

Array de Arrays: para guardar las subsecuencias de números

```
Type Subsecuencia is array (1.. 3) of Lista(N);
```

Array de 3 tareas (ya no declaramos un tipo, ponemos directamente el array)

```
PROCESO_ORDENACION: ARRAY (1..3) of ORDENACION;
```

3. Suponiendo que en el programa principal están declarados los tipos que se indican en el Segundo apartando, y que tenemos un procedimiento que divide la secuencia en las tres subsecuencias, El cuerpo de la tarea T tendría la siguiente estructura:

```
task body T is
  Lentrada, Lsalida : Lista (100); ---suponemos que tenemos 100 números
  Lintermedia : Subsecuencia (100);
begin
  Formar_Lista (Lentrada, Lintermedia, 3);
  -- divide Lentrada en tres subsecuencias que guarda en Lintermedia
  For I in 1..3 loop
    PROCESO_ORDENACION[I].LISTA (Lintermedia[I]); --llamada a la tarea de ordenación
  end loop;
  UNIR (Lintermedia, Lsalida); -- Une las subsecuencias en el array de números Lsalida
  Mostrar_Resultado (Lsalida);
End T;
```

4. Las otras tareas de ORDENACION tienen la siguiente estructura:

```
task body ORDENACION is
begin loop
  select
    accept Lista (La_lista: IN OUT LISTA) do -- es una lista de números
      DEVUELVE_SECUENCIA_ORDENADA (La_Lista);
    end;
  end select;
end ORDENACION;
```

5. Para poder producir un cambio de este tipo hay que cambiar cada proceso de ordenación según los siguientes criterios. Ahora todas las tareas son iguales (excepto el programa principal) ya que reciben una secuencia y si su tamaño es mayor que 2, la dividen en tres partes y continúan el proceso. Ahora, además de recibir la secuencia, necesitan dos valores: el rango inferior y el rango superior de la secuencia.

Debido a que cada tarea llama a su vez a otras tareas, dentro del cuerpo de cada tarea se declara un array de tres tareas que se invocará después de que se divida la secuencia de entrada en las tres subsecuencias, y así sucesivamente hasta que la subsecuencia tenga tamaño 2. En este último caso, no se estarán utilizando las tres tareas locales, ya que no se va a llamar a sus puntos de entrada. La ventaja es que se podrá utilizar el mismo código que en las otras tareas.

El cambio sería el siguiente: Una tarea recibe la secuencia, el valor inferior y el valor superior, Calcula los rangos de las tres subsecuencias, Divide la lista, Llama a sus tareas locales y espera el resultado, Une el resultado en un único array, Devuelve el resultado.



SEGUNDA SEMANA

PREGUNTA 1. Tiempo estimado 55 minutos

PUNTUACIÓN 4 puntos

Un banco necesita un sistema para controlar las acciones en bolsa de sus clientes. Hace falta guardar y realizar lo siguiente:

Información sobre las empresas:

- Cada empresa está dentro de un sector. Ejemplos de sectores son Bancos, Eléctricas, Alimentación, Construcción, etc.
- Dentro de cada sector, para cada empresa se guarda su índice (valor medio de todos los valores de las empresas de ese sector), el valor mínimo, el valor máximo y la diferencia (este se calcula entre el valor máximo y mínimo)
- Para cada empresa, información sobre: su categoría, el nombre del valor, el precio anterior y los valores mínimos, máximos y la diferencia.

Métodos para usar con información sobre las empresas:

- La diferencia (entre los valores máximo y mínimo) para una empresa, un sector y todos los sectores.
- El valor (mínimo, actual y máximo) de n acciones en una empresa.

Por ejemplo, para el sector Banca si solo tuviésemos tres empresas B.PASTOR, B.POPULAR, B.VALENCIA:

SECTOR : Bancos y Financieras					
		Último	Difer.	Máximo	Mínimo
		27,76	0,52	27,92	27,4
Empresa	Anterior	Último	Difer.	Máximo	Mínimo
B.PASTOR	41,95	42,20	0,53	42,23	41,70
B.POPULAR	31,48	31,80	0,89	32,19	31,30
B.VALENCIA	9,34	9,28	0,13	9,34	9,21

Información sobre un cliente:

- Cuántas acciones tiene en cada empresa.
- A qué precio compró las acciones de cada empresa.
- Los órdenes que ha establecido para compra/venta (es decir: el nombre de la empresa, que quiere hacer [comprar o vender], el precio que está esperando para realizar la acción, y cuanto quiere gastar si se trata de una compra).

Métodos para los clientes:

- Poner órdenes para la compra/venta de acciones, especificando precio y valor.
- Calcular el valor actual de todas sus acciones.
- Calcular la diferencia entre el valor actual de todas sus acciones y su valor de compra.

Se pide:

1. Identificar las clases necesarias para este sistema y dibujar un esquema de la organización de estas clases en el diseño global. (1,5 puntos)
2. Implementar la especificación de las clases en un lenguaje orientado a objetos. (1,5 puntos)
3. Implementar los métodos para listar el valor de todas las acciones que tiene un cliente. (1 punto)

**SOLUCIÓN:**

1. Se hará falta clases para representar los siguientes objetos en el sistema:

- **Una acción:** nombre de la empresa, nombre del valor, precios (anterior, mínimo, máximo, y último).
- **Acciones de cada cliente:** (datos del objeto acción) número de acciones en esta empresa, el precio de compra, ordenes de compra / venta.
- **Un sector:** nombre del sector, una lista de acciones, precios (mínimo, máximo, y último).
- **Sectores:** una lista de todos los sectores en el mercado, precios (mínimo, máximo, y último).
- **El cliente:** datos personales, una lista de objetos 'Acciones de cada cliente' para cada empresa en que el cliente tienen acciones.



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

- **ControlarAcciones:** una lista de objetos 'El cliente' y un objeto 'Sectores' para controlar los datos del mercado y del cliente.

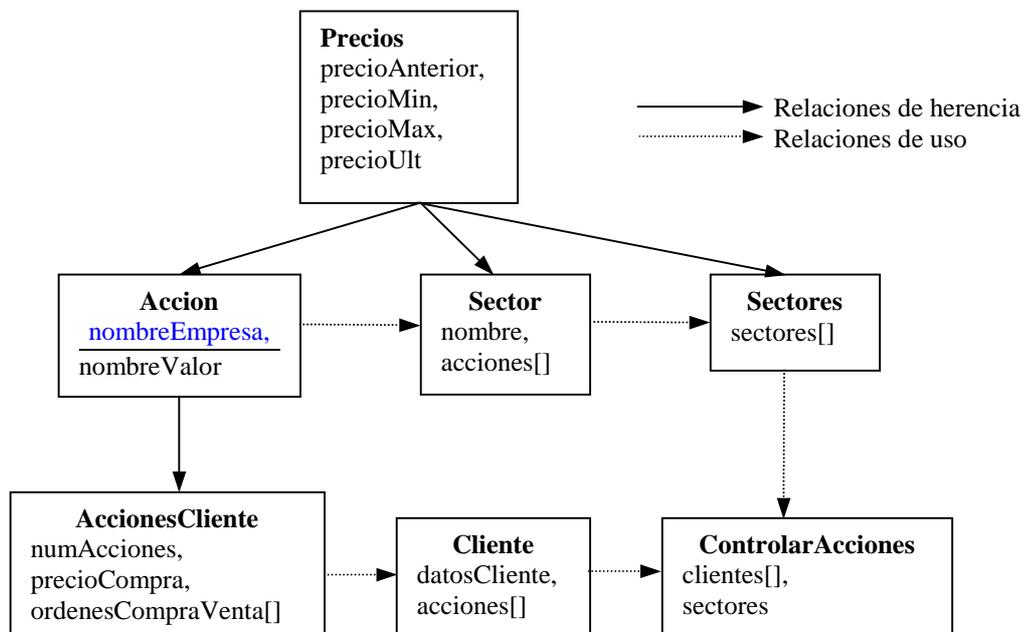
Notas:

- a. Como las clases **Una accion**, **Acciones de cada cliente**, **Un sector** y **Sectores** tienen datos sobre los precios se podría tener otra clase que tiene solamente estos datos:
- **Precios:** anterior, mínimo, máximo, y último.

En el caso de los sectores donde no hay ningún precio anterior, se podría dejar este valor a cero.

- b. No hay ninguna representación para la diferencia entre los precios máximo y mínimo porque se puede calcular este valor sobre la marcha.

Se puede ver la relación de estas clases a continuación:



2. En Java la especificación de las clases anteriores sería:

```
class Precios {
    //datos de los precios
    private double precioAnterior, precioMin, precioMax, precioUlt;
}

class Accion extends Precios {
    private String empresa; //nombre de la empresa
    private String nombreValor; //nombre que aparece para el valor
}

class AccionesCliente extends Accion {
    private int noAcciones; //el número de acciones
    private int precioCompra; //lo que pago el cliente para cada acción
    private String ordenesCompraVenta[]; //todas las ordenes relacionadas a esta empresa
}

class Cliente {
    private String datosClientes;
```



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

```
private AccionesCliente acciones[];
}

class Sector extends Precios {
    private String nombre; //nombre del sector
    private Accion accion[]; //un array de todos las acciones que componen este sector
}

class Sectores extends Precios {
    private Sector sectores[]; //la lista de todos los sectores
}

class ControlarAcciones {
    private Cliente clientes[]; //un array de todas las clientes y sus acciones
    private Sectores sectores; //una referencia a la lista principal de acciones
}
```

3. Se puede ver los métodos necesarios para listar el valor de todas las acciones que tiene un cliente en el siguiente program basado en las clases anteriores:

```
class Precios {
    //datos de los precios
    private double precioAnterior, precioMin, precioMax, precioUlt;

    public Precios(double pA, double pMin, double pMax, double pUlt) {
        precioAnterior = pA;
        precioMin = pMin;
        precioMax = pMax;
        precioUlt = pUlt;
    }

    public double getValorActual(){
        return precioUlt;
    }
}

class Accion extends Precios {
    private String empresa; //nombre de la empresa
    private String nombreValor; //nombre que aparece para el valor

    public Accion(String e, String n, double pA, double pMin, double pMax, double pUlt) {
        super(pA,pMin,pMax,pUlt);
        empresa = e;
        nombreValor = n;
    }
}

class AccionesCliente extends Accion {
    private int noAcciones; //el número de acciones
    private int precioCompra; //lo que pago el cliente para cada acción
    private String ordenesCompraVenta[]; //todas las ordenes relacionadas a esta empresa

    public AccionesCliente(String e, String n, double pA, double pMin, double pMax,
        double pUlt, int noA, int pCompra, String oCV[]) {
        super(e,n,pA,pMin,pMax,pUlt);
        noAcciones = noA;
        precioCompra = pCompra;
        ordenesCompraVenta = oCV;
    }

    public double getValor(){
        return(noAcciones * getValorActual());
    }
}

class Cliente {
    private String datosClientes;
    private AccionesCliente acciones[];

    public Cliente(String dC, AccionesCliente acs[]) {
```



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
 ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

```

datosClientes = dC;
AccionesCliente acciones;
}

public AccionesCliente[] getAcciones(){
    return acciones;
}
}

class Sector extends Precios {
    private String nombre; //nombre del sector
    private Accion accion[]; //un array de todos las acciones que componen este sector

    public Sector(String n, Accion a[], double pMin, double pMax, double pUlt) {
        super(0.0,pMin,pMax,pUlt);
    }
}

class Sectores extends Precios {
    private Sector sectores[]; //la lista de todos los sectores

    public Sectores(Sector s[], double pMin, double pMax, double pUlt) {
        super(0.0,pMin,pMax,pUlt);
    }
}

class ControlarAcciones {
    private Cliente clientes[]; //un array de todas las clientes y sus acciones
    private Sectores sectores; //una referencia a la lista principal de acciones
}

class ControlarMain {
    public static void main(String argv[]) {
        AccionesCliente ac[] = new AccionesCliente[2];
        ac[0] = new AccionesCliente("Banco Pastor","B.Pastor",41.95,42.23,41.7,42.2,100,40,null);
        ac[1] = new AccionesCliente("Banco Popular","B.Popular",31.48,31.3,32.19,31.8,45,20,null);
        Cliente c = new Cliente("D. Diego FooBar, 91-111-2222",ac);

        //calcular valores
        double v = 0.0;
        for (int i=0;i<2;i++)
            v += ac[i].getValor();
        System.out.println("Valor total es: " + v);
    }
}

```

Que produce el siguiente resultado:

Valor total es: 5651.0

**PREGUNTA 2. Tiempo estimado 15 minutos**

**PUNTUACIÓN 1,5 puntos**

Dados los siguientes tres ejemplos escritos en un lenguaje:

<pre> for J in S'range loop     Name (J) := S (J);     exit when J &gt;= Name'Last; end loop;  for Count in W.Current.Column .. W.Last.Column loop     Text_IO.Put (' '); end loop; </pre> <p style="text-align: center;"><b>Ejemplo 1</b></p>	<pre> while L1 /= Nil loop     A := Element (L1);     Display_Value (A.all);     L1 := Tail (L1); end loop; </pre> <p style="text-align: center;"><b>Ejemplo 2</b></p>	<pre> if A=B then     A := B + 1; elsif A&gt;C then     A := C + 1; elsif A&lt;D then     A := D + 1; end if; </pre> <p style="text-align: center;"><b>Ejemplo 3</b></p>
--	--	--

1. Escribir una gramática BNF para los ejemplos 1, 2 y 3. (1 punto)
2. Producir un esquema de sintaxis para la gramática 1. (0,5 punto)



**SOLUCIÓN:**

Considerando sólo las sentencias de cada ejemplo, una posibles gramáticas para el código de los ejemplos, podrían ser las siguientes

Ejemplo 1

```

<ejemplo1> ::= for <identificador>in <rango> <bucle>;
<identificador> ::= <letra><letra><identificador>
<letra> ::= a|A|b|B|c...|Z
<bucle> ::= loop <sentencias> end loop
<sentencias> ::= <sentencia><sentencias> |<sentencia>
<sentencia> ::= <sentencia_asignación> |<sentencia_salida>
                |<sentencia_escritura>
<rango> ::= <identificador> .. <identificador> | <identificador> ' range
<sentencia_asignación> ::= <variable> := <variable>
<variable> := <identificador> | <identificador> ( <identificador> )
<sentencia_salida> ::= exit when <condicion> (*<condicion> en Ejemplo 3 )
<sentencia_escritura> ::= text_io.Put ( ' <valor> ' )
<valor> ::= <variable> | <numero> | <numero><valor> | <variable><valor>|<cadena_vacia>

```

Ejemplo 2

```

<ejemplo2> ::= while <condicion> <bucle> (bucle tiene la definición del ejemplo 1 y expresión del ejemplo3)

```

Ejemplo 3

```

<ejemplo3> ::= if <condicion> then <sentencia_suma1> <resto>;
<resto> ::= elsif <condicion> then <sentencia_suma1 > <resto> |
end if;
<condicion> ::= <identificador> <comparación> <identificador>
                (*<identificador> en Ejemplo 1 )
<comparación> ::= = | > | < | /=
<sentencia_suma1> ::= <identificador> + 1;

```

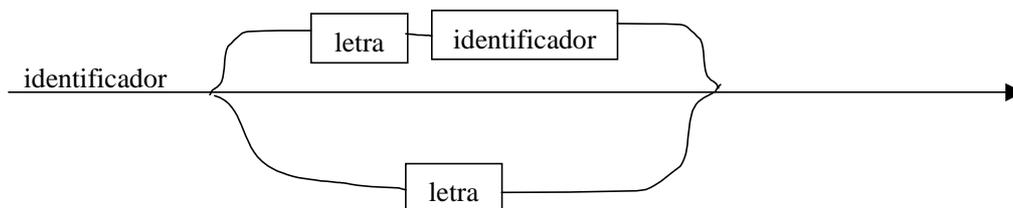
2) Esquema de sintaxis para la gramática 1. (Vale un esquema de cualquiera de las producciones, Ponemos aquí un par de ejemplos).

Por ejemplo, producimos un esquema de sintaxis para

<ejemplo1> ::= for <identificador>in <rango> <bucle>;



<identificador> ::= <letra><letra><identificador>





UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

PREGUNTA 3. Tiempo estimado 50 minutos

PUNTUACIÓN 4,5 puntos

$$K = \sum_{i=1}^m n^i$$

Se quiere implementar un programa que realice el sumatorio de un número elevado a n, es decir

Por ejemplo, para n=2 y m=3 deberá realizar  $2^1+2^2+2^3$  y devolverá 14  
Para n=3 y m=5 deberá realizar  $3^1+3^2+3^3+3^4+3^5$

Si se quiere realizar el programa utilizando las ventajas que ofrece el paradigma concurrente

- 1) Identificar y justificar las tareas necesarias para este programa para m=3 y para cualquier número n. Para cada tarea indicar su especificación y los puntos de entrada necesario (justificando las estructuras de datos utilizadas y el tipo de paso de parámetros elegido). (1 punto)
- 2) Explicar cómo se soluciona el problema si queremos generalizar para un número m genérico que introduce el usuario por teclado (0,5 puntos)
- 3) Implementar el cuerpo de las tareas identificadas en el apartado 1. (1 punto)

Si nos planteamos resolver el problema recursivamente

- 4) Identifica la(s) estructura(s) de datos necesarias y la definición de la función recursiva. Escribe el pseudocódigo de la función que realiza el sumatorio de este problema. (1 punto)
- 5) Compara las soluciones paralela y recursiva para este problema. ¿Qué ventajas e inconvenientes le ves a cada solución? ¿Encuentras otra solución mejor en otro paradigma de programación? (1 punto)

#### SOLUCIÓN:

1. Para m, se necesitan 5 tareas más el programa principal

Tarea A: Una tarea que inicie el proceso

Tres tareas B: van pasándose el número, multiplicado por sí mismo. Como son iguales hacemos un array de tareas

Tarea C: Un tarea que vaya sumando y que devuelva el resultado cuando se haya hecho el proceso m veces.

El programa en Ada, correspondiente el es el siguiente<sup>1</sup>. Apartados 1 y 3 :

```

Procedure HACER is
  ---- Especificación de las tareas
  Task A;
  Task type B is
    entry E (X: in INTEGER, SIG: in INTEGER);
  end B;
  Task C is
    Entry R (X: in INTEGER);
    Entry S (X: in INTEGER);
    Entry T (X: in INTEGER);
  end C;

  miArray : array (1..3) of B;
  maximo: INTEGER;

  --- Cuerpo de las tareas
  task body A is
    miA: INTEGER;
  BEGIN

```

<sup>1</sup> (A modo informativo, la solución a este problema es en enunciado del problema 1 (segunda semana) de junio de 1999, en el que se mostraba el código para m=3 y se preguntaba qué hacía el programa)



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

```
PUT_LINE ("Inicio tarea A"); --saca mensaje por pantalla
Delay(10);
GET(miA);
PUT_LINE ("Soy la tarea A. He leído"); PUT(miA); PUT_LINE; --saca mensaje por
pantalla
C.R(miA);
miArray[1].E(miA, 1);
END A;
-----
task body B is
BEGIN
PutLine ("inicio tarea B");
accept E (in X: INTEGER; in SIG: INTEGER) do
PUT("Soy la tarea"); PUT(SIG); PUT("He leído"); PUT(X); PUT_LINE; -- por
pantalla
if (SIG /= maximo) then
C.S (X*X);
miArray[SIG+1].E(X*X, SIG+1);
else
C.T(X*X);
end E; -- del accept
END B;
-----
task body C is
total: INTEGER:= 0;
BEGIN
PutLine ("inicio tarea C"); --saca mensaje por pantalla
select
when (total = 0) =>
accept R(X: in INTEGER) do
total := X;
PUT("Soy la tarea C: Total vale:"); PUT(TOTAL); PUT_LINE; --saca
mensaje por pantalla
or when (total>0) =>
accept S(X: in INTEGER) do
total:= total + X;
PUT("Soy la tarea C: Total vale:"); PUT(TOTAL); PUT_LINE; --saca
mensaje por pantalla
end;
or accept T(X: in INTEGER) do
total:= total + X;
PUT("Soy la tarea C: Total vale:"); PUT(TOTAL); PUT_LINE; --saca
mensaje por pantalla
total:= 0;
or terminate; -- para que salga del select
end select;
END C;
BEGIN -- Programa principal
PUT_LINE ("Inicio del programa principal"); --saca mensaje por pantalla
maximo := 3;
END HACER;
```

2) Sería que el usuario leyese la variable máximo por teclado y que en la declaración del array `miArray : array (1..3) of B;` se declarase `miArray : array (1..maximo) of B;`

3) Se incluye con el punto 1.

4) Una función recursiva para este problema tendría la siguiente estructura:

```
function Sumatorio(N: INTEGER; Elev: INTEGER) return INTEGER
K: INTEGER;
begin
if Elev > 1 then
K := N;
for I in 1 to Elev loop
K = K * N;
end loop;
return(K + Sumatorio(N,Elev-1));
```



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA UNIVERSITARIA DE INFORMÁTICA

EXAMEN DE LENGUAJES DE PROGRAMACIÓN

---

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

---

```
else
    return(N);
endif;
end;
end Sumatorio;
```

5.) Solución Paralela:

+ Más rápida, menos calculaciones, - Más código

Solución Recursiva:

+ Menos código, - Más calculaciones, más lento..

PRIMERA SEMANA

**PREGUNTA 1. Tiempo estimado 20 minutos**

**3 puntos**

En la definición de un Lenguaje de Programación

- 1) ¿Qué diferencia hay entre una definición o modelo **conceptual** de un lenguaje de y un modelo **formal**? (0,5 puntos)
- 2) ¿Cómo se hace la definición formal de la sintaxis de un lenguaje de programación?. Nombrar el modo y describir sus elementos. Ilustrar la explicación con un ejemplo sencillo (1, 5 puntos)
- 3) ¿Para qué sirve el modelado semántico de un lenguaje de programación? Explicarlo y nombrar al menos tres de los métodos existentes para realizarlo. (1 punto)

Solución.

1)

- Un modelo que definir un lenguaje puede ser conceptual (es decir, cualitativo) y describir el lenguaje en términos de un conjunto subyacente de conceptos básicos (por ejemplo: variables, bucles, etc), sin tener que dar una descripción matemática formal de los conceptos.
- Por otro lado, si es forma (es decir, cuantitativo) describe el lenguaje en términos de un modelo matemático preciso que se puede estudiar, analizar y manipular utilizando herramientas matemáticas (Página 404 del libro)

- 2) La definición formal de la sintaxis de un lenguaje de programación se hace con una **gramática** que se compone de un conjunto de **reglas** o **producciones** que especifican las series de caracteres (o **elementos léxicos**) que forman programas permisibles en el lenguaje que se está definiendo. (página 79 del libro)

Ejemplo: Una gramática para expresiones con sumas y restas

Expresión  $\rightarrow$  Expresión + Termino | Expresión - Termino | Término

Termino  $\rightarrow$  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Esta gramática tiene trece producciones, tres para Expresión y diez para Terminos. Y reconoce un lenguaje de expresiones matemáticas con sumas y restas como 2+3-5. Los elementos léxicos son el -, + y 0..9

- 3) Permite dar significado y definir el efecto de la ejecución sobre cualquier enunciado o sentencia de un programa escrito en un lenguaje de programación. Lo utiliza el creador del lenguaje para por construir un compilador correcto. (página 98 del libro)

Métodos: (tres de los siguientes):

- Modelos gramaticales
- Modelos imperativos y operativos
- Modelos aplicativos
- Modelos axiomáticos
- Modelos de especificación

**PREGUNTA 2. Tiempo estimado 50 minutos**

**3,5 puntos**

Los lenguajes de programación tiene diferentes métodos para realizar el **paso de parámetros** de un subprograma o función a otro:

1. Nombrar, explicar y poner un ejemplo de los métodos de paso de parámetros existentes en los lenguajes de programación (1,5 puntos)
2. Poner un ejemplo de un procedimiento con un tipo de paso de parámetros elegido y explicar cómo sería el funcionamiento (memoria, forma de guardar la llamada, forma de devolver los datos, etc.) par una llamada concreta (1 punto)

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

3. Algunos lenguajes de programación permiten que existan múltiples nombres para un mismo objeto de datos en un mismo programa. Indicar las dos situaciones en las que se puede dar esta situación. Poner un ejemplo de cada caso (1 puntos)

Solución (página 312-):

1. A la hora de llamar a un subprograma o función (teniendo en cuenta la distinción entre un parámetro real - el objeto de datos que se comparte con el subprograma, y el parámetro formal - una variable local del mismo tipo de datos que el parámetro real), existen 6 métodos de paso de parámetros:

- Llamada por nombre** - cada parámetro formal representa la evaluación concreta del parámetro real correspondiente, y la transmisión de parámetros trata una llamada de subprograma como una sustitución del cuerpo completo del subprograma. Un ejemplo también podría ser un acceso a una variable dentro de un procedimiento o clase, como P1.x.
- Llamada por referencia** - se transmite un apuntador a la posición del elemento de datos; el objeto de datos no cambia de posición en memoria. Por ejemplo P2(&q);
- Llamada por valor** - se transmite el valor del parámetro y, por lo tanto, la función / subprograma no tiene acceso a la variable para cambiar su valor. Por ejemplo, P3(x + 10);
- Llamada por valor-resultado** - se parece a la llamada por valor excepto que al final del funcionamiento del subprograma / función, el valor original de la variable se sustituye por el valor nuevo.
- Llamada por valor constante** - se parece a la llamada por valor. No se puede asignar un nuevo valor ni modificar el valor del parámetro formal.
- Llamada por resultado** - se usa sólo para transmitir un resultado de retorno desde un subprograma.

2.

```
Q(int i, int *j) {
    i = i + 10;
    *j = *j + 10;
    printf("i=%d; j=%d\n", i, *j);
}
```

Subprograma llamado

```
P() {
    int a, b;
    a = 2;
    b = 3;
    Q(a, &b);
    printf("%d %d\n", a, b);
}
```

Subprograma de llamada

Se puede ver un ejemplo de llamada a un subprograma con dos parámetros formales, *i* transmitido por valor y *j*, transmitido por referencia. En *P* se evalúan las expresiones de parámetro real *a* y *&b* para transmitir el valor de *a* y la localización de *b*. Por lo tanto, *i* se representa como una variable local en *Q* (y los cambios de *i* no modifican *a*) y *j* es un apuntador a un entero. Cuando *Q* inicia su ejecución, las modificaciones de *i* son locales a *Q* y las de *j* cambian el valor de la variable *b*. Así que, cuando termina *Q*, la variable *a* sigue con el valor 2 y *b* tiene un valor nuevo de 13.

3. Principalmente se puede destacar la relación entre variables globales y locales con nombres distintos para el mismo elemento de datos, como por ejemplo:

```
void imprime (int foo) { ... }
```

```
main () {
    int N = 5;
    imprime(N);
}
```

y la relación entre variables globales y punteros:

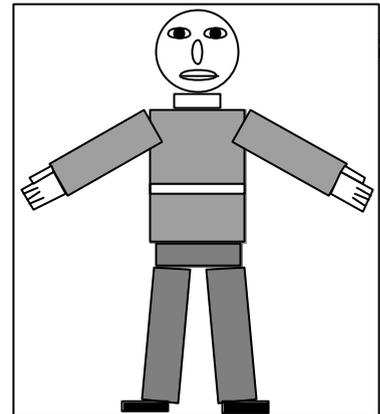
```
void cambio (int *px, int *py) {
    int t;
    t = *px;
    *px = *py;
    *py = t;
}

main(){
    int i = 5;
    int j = 10;
    swap(&i,&j);
    ...
}
```

### **Pregunta 3. Tiempo estimado 40 minutos**

La imagen mostrada aquí estaba realizada por un programa de dibujo por ordenador para niños. Como puede verse, la figura está compuesta por un conjunto de formas geométricas de varios tipos, tamaños, colores, texturas, etc.

1. Indica la utilidad de la programación orientada a objetos en un programa de este tipo que utiliza diferentes figuras geométricas. ¿Qué papel tendría el polimorfismo en un programa así? (1 punto)
2. Realizar un diseño orientado a objetos para las estructuras de datos y de control necesarios para un programa que permita hacer dibujos compuestos a partir de figuras más sencillas. Identificar las clases necesarias y mostrar **claramente** los distintos tipos de relaciones en el diseño. (2,5 puntos)



Solución:

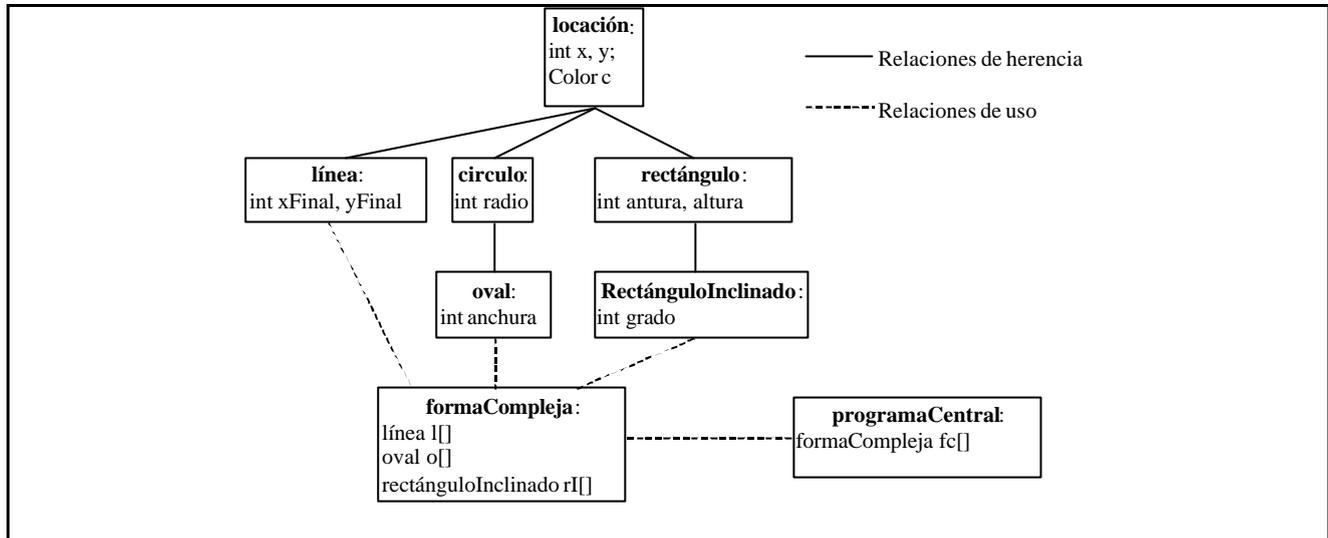
1. La programación orientada a objetos ofrece ventajas para la resolución de problemas como este. Por ejemplo, la herencia ayuda a evitar errores por que no hay que repetir variables y métodos innecesariamente. El encapsulamiento de datos permite aislar las variables de cada figura y controlar los tipos de acceso permisibles. La estructura jerárquica de los programas (basada en los puntos anteriores y el polimorfismo) también ayuda con la incorporación de nuevas figuras con los mínimos cambios necesarios.

El polimorfismo permite que un solo operador o nombre de subprograma se referira a varias definiciones de función, de acuerdo con los tipos de datos de los argumentos y resultados. En este ejemplo, el polimorfismo ayuda porque la imagen está compuesta por formas geométricas muy parecidas y, por lo tanto, tiene sentido que haya métodos parecidos para tratarlos dentro de cada clase (como se puede ver en la siguiente parte de esta respuesta). Así que se pueden tener en cuenta los pequeños cambios de cada forma sin modificar la forma de tratarlos.

2. Se puede ver un diseño orientado a objetos para este problema a continuación:

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

Y la estructura de clases (sin acentos) en Java sería:

```

private class locacion {
    private int x, y;
    private Color c;

    //muestra de métodos
    public locacion(int xx, int yy, Color cc){}
    public int getX() {}
    public int getY() {}
}

public class linea extends locacion {
    private int xFinal, yFinal;

    //muestra de métodos
    public linea (int x1, x2, y1, y2){}
    public int getxFinal(){ }
    public int getyFinal(){ }
}

private class circulo extends locacion {
    private int radio;

    //muestra de métodos
    private circulo(int xx, int yy, Color cc, int r){}
    private dibujar();
}

private class rectangulo extends locacion {
    private int anchura, altura;

    //muestra de métodos
    private rectángulo(int xx, int yy, int ant, int alt, Color c);
    private dibujar();
}

public class oval extends circulo {
    private int anchura;
  
```

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

---

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

---

```
//muestra de métodos
public oval(int xx, int yy, Color cc, int ant, int alt);
public dibujar();
}

public class rectanguloInclinado extends rectangulo{
    private int grado;

    //muestra de métodos
    private rectánguloInclinado(int xx, int yy, int ant, int alt, int g, Color c);
    private dibujar();
}

public class formaCompleja {
    linea l[];
    oval o[];
    rectanguloInclinado rI[];

    public formaCompleja(linea ll[], oval oo[], rII[]);
    private dibujar();
}

public class programaCentral {
    private formaCompleja formas[] = new formaCompleja[1000];

    //código para crear área gráfico y los botones para cada forma

    //código para captar la acción del ratón y formar una nueva forma básica

    //código para añadir las formas básicas existentes a una forma compleja nueva

    // etc.
}
```

**SEGUNDA SEMANA**

**Pregunta 1. Tiempo estimado 40 minutos**

**3 puntos**

Queremos modelar un sistema que registre y gestione las operaciones llevadas a cabo en una entidad bancaria, que pueden ser de o bien retirada de dinero o bien ingreso en efectivo. Para cada operación se apunta la fecha y hora en que tiene lugar, así como el cajero en el que se realiza y el número de cuenta.

Para cada cuenta se dispone, entre otros, de los siguientes datos: número de cuenta, el DNI del titular y el saldo. Vamos a considerar tres tipos de cuentas:

- **cuentas corrientes**, en las que la cantidad restada se actualiza inmediatamente, tras autorizarse la operación
- **cuentas de crédito**, en las que el importe de las operaciones de retirada de efectivo se carga en cuenta a final de mes y la operación se autoriza siempre que no se alcanza un límite de crédito, acordado en el momento de solicitar la cuenta. Al final de cada mes, un proceso de la entidad se encarga de actualizar los saldos de estas cuentas considerando las operaciones realizadas a lo largo del mes que se procesa.
- **cuenta bonificada**: ofrece las características de las otras cuentas de crédito más otras bonificaciones a sus titulares, a razón de diez puntos de promoción por cada movimiento efectuado. Cada punto de promoción se traduce en premios (coches, vajillas, viajes....) para el cliente.

En esta entidad bancaria, un cliente NO puede poseer más de una cuenta a su nombre.

1. Identificar las estructuras de datos para implementar un programa orientado a objetos que gestione las operaciones bancarias de esta entidad (1 punto)
2. Proponer una jerarquía de clases adecuada para el dominio del problema. Señala los métodos que son virtuales y dónde se utiliza polimorfismo. Destacar claramente las relaciones y los tipos de relaciones entre las clases (2 puntos)

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**Apartado 1

Para resolver el problema vamos a utilizar el paradigma Orientado a Objetos, tal y como se nos dice en el enunciado. Vamos a considerar las siguientes clases, explicando su estructura:

Entidad: que guarda el nombre y código de la entidad bancaria y el número de sucursal.

Guarda también una lista de cajeros y una lista de cuentas.

Cajero: guarda el número de cajero.

Cliente: es el titular de la cuenta. DNI, nombre, apellidos.

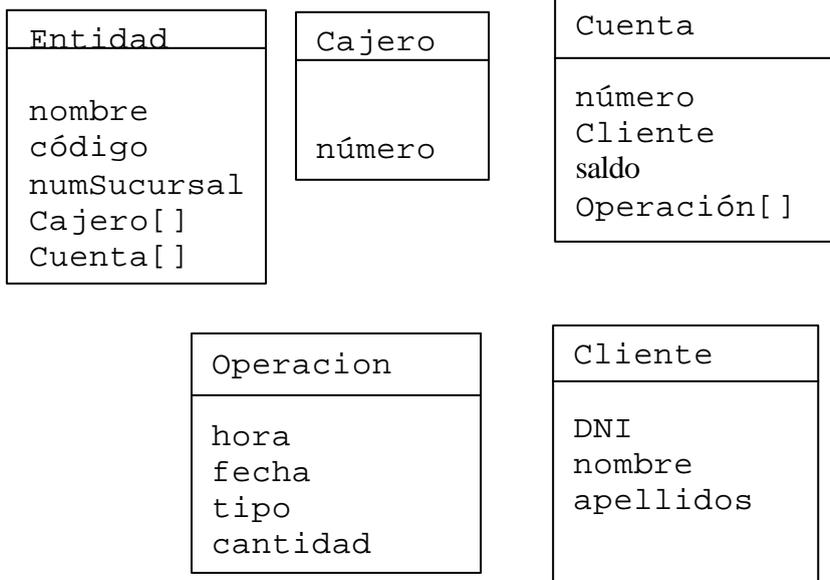
Operación: guarda la hora, fecha en que tiene lugar, el tipo de operación (retirada o ingreso), la cantidad y el cajero.

Cuenta: tiene un cliente asociado, un número de cuenta y el saldo. También guarda una lista con todas las operaciones efectuadas.

CuentaCredito tiene los atributos de Cuenta.

CuentaCorriente tiene los de Cuenta y además un *límiteCrédito* y un *créditoAcumulado*.

CuentaBonificada tiene los de CuentaCorriente y además *puntos*, que el cliente va a acumulando.

Apartado 2

Se va a definir los principales métodos de cada clase, para luego explicar las relaciones de uso y de herencia.

La clase base Cuenta va a definir los métodos *ingresar* y *retirarEfectivo* para efectuar las operaciones. Puede considerarse un solo método *efectuarOperación*, de forma que el signo del importe determine si se trata de un ingreso de un reintegro. *ingresar* y *retirarEfectivo* son métodos virtuales invocados desde la clase Cajero –haciendo uso del polimorfismo, por tanto- y que anotan las operaciones en una lista. Se va a definir

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

actualizarSaldo() virtual y abstracto, el cuál se definirá en las subclases de Cuenta.

```
ingresar(Cajero cajero, Fecha fecha, int cantidad) :  
    Añadir una nueva operación de tipo ingreso a la lista.
```

```
retirarEfectivo(Cajero cajero, Fecha fecha, int cantidad) :  
    Añadir una nueva operación de tipo reintegro a la lista.
```

actualizarSaldo() es abstracta.

De Cuenta derivan CuentaCorriente y CuentaCredito.

CuentaCorriente redefine ingresar y retirarEfectivo, invocando también los correspondientes de la superclase. En el caso de ingresar, se actualiza directamente el saldo. RetirarEfectivo comprueba si la operación es autorizada  $-\text{saldo-cantidad} > 0$  y actualiza el saldo en caso afirmativo. ActualizarSaldo() aquí es redefinido sin ninguna sentencia, ya que el saldo se actualiza en cada operación.

```
ingresar(Cajero cajero, Fecha fecha, int cantidad) :  
    super.ingresar(cajero, fecha, cantidad);  
    saldo+=cantidad,
```

```
retirarEfectivo(Cajero cajero, Fecha fecha, int cantidad) :  
    super.retirarEfectivo(cajero, fecha, cantidad);  
    si (saldo-cantidad>=0)  
        entonces  
            saldo-=cantidad;
```

```
actualizarSaldo():
```

La clase CuentaCorriente redefine ingreso y retirarEfectivo, invocando antes los correspondientes métodos de la superclase. Para ingreso se actualiza el miembro creditoAcumulado. Para retirarEfectivo se comprueba que  $\text{creditoAcumulado} < \text{limiteCredito}$  para poder autorizar la operación. ActualizarSaldo() aquí hace  $\text{saldo} += \text{creditoAcumulado}$  y posteriormente puede realizarse algún cargo (intereses a favor, etc.)

```
ingresar(Cajero cajero, Fecha fecha, int cantidad) :  
    super.ingresar(cajero, fecha, cantidad);  
    creditoAcumulado-=cantidad;
```

Aquí se ha optado por hacer que los ingresos sean también diferidos, de forma que afectan al crédito acumulado.

```
retirarEfectivo(Cajero cajero, Fecha fecha, int cantidad) :  
    super.retirarEfectivo(cajero, fecha, cantidad);
```

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

---

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

---

```
si (creditoAcumulado+cantidad<=limiteCredito)
    entonces
        creditoAcumulado +=cantidad;
```

creditoAcumulado se resta.

```
actualizarSaldo():
    saldo-=creditoAcumulado;
    creditoAcumulado=0;
```

CuentaBonificado deriva de CuentaCorriente. Por ello, hereda de ella todos sus miembros, por lo que actualizarSaldo() es exactamente igual. Ingresar y retirarEfectivo son redefinidas invocando antes las de la superclase y añadiendo la bonificación correspondiente.

```
ingresar(Cajero cajero, Fecha fecha, int cantidad) :
    super.ingresar(cajero, fecha, cantidad);
    puntos+=10;
```

```
retirarEfectivo(Cajero cajero, Fecha fecha, int cantidad) :
    super.retirarEfectivo(cajero, fecha, cantidad);
    puntos+=10;
```

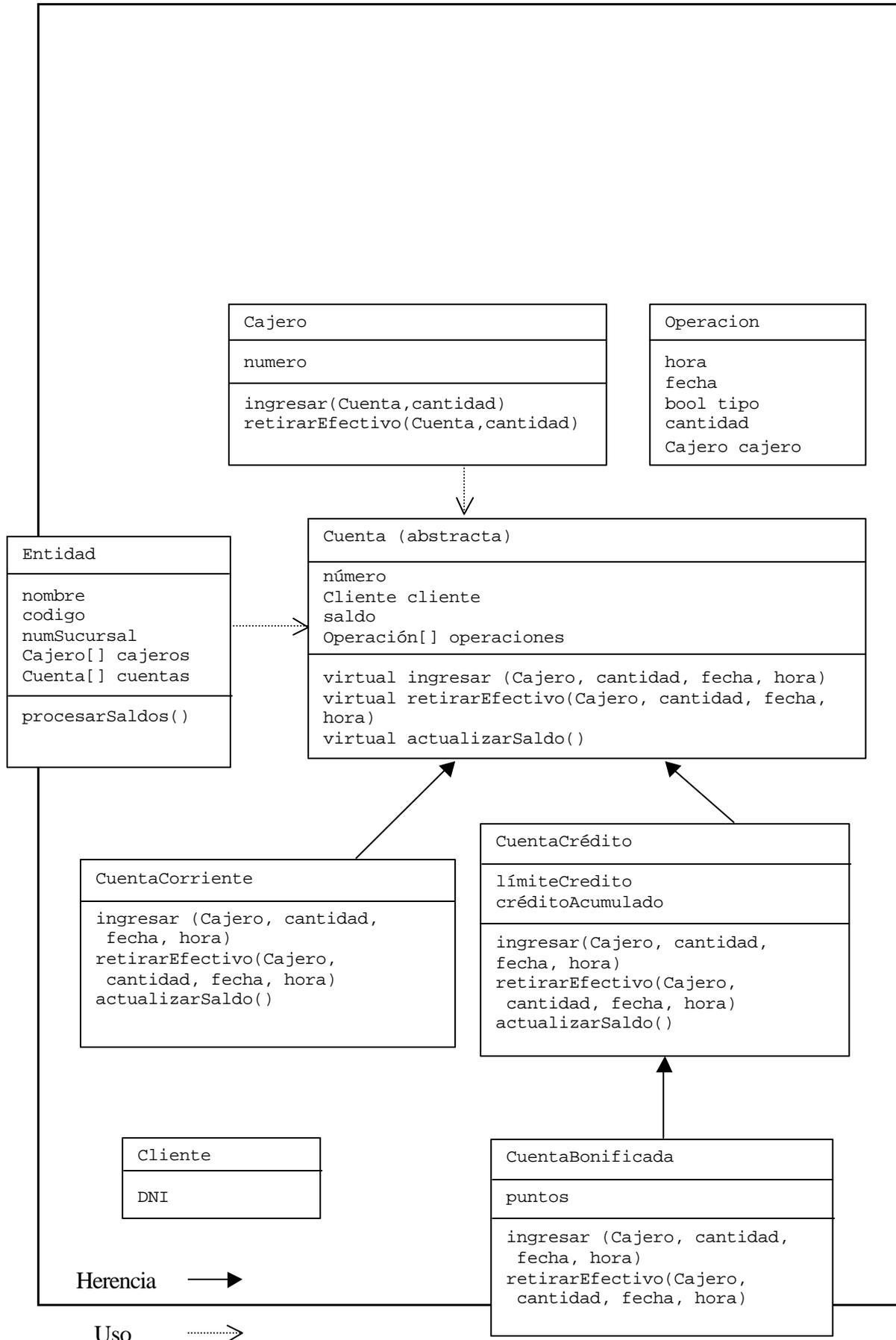
La clase Entidad define el método procesarSaldos(), que hace uso del método actualizarSaldo() de las cuentas de la Entidad, haciendo uso del polimorfismo.

```
procesarSaldos():
    Parar todas las cuentas de cuentas hacer
        cuenta[i].actualizarSaldo();
```

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**



---

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

---

**Pregunta 2. Tiempo estimado 40 minutos**

**3,5 puntos**

- 1) ¿Qué es la semántica de un lenguaje de programación? Para qué sirve y que diferencia hay con la sintaxis (0,5 puntos)
- 2) Nombrar las funciones más comunes de un analizador semántico de un lenguaje. (0,5 puntos)
- 3) Si consideramos un lenguaje para expresiones aritméticas (sólo sumas y productos con números enteros y donde puede hacer paréntesis) (1,5 puntos, 0,5 cada apartado)
  - Definir la sintaxis de este lenguaje
  - Indicar una forma sencilla de reconocer los elementos sintácticos de un lenguaje y aplicarlo al de expresiones aritméticas definida
  - Indicar una forma de definir un modelo semántico para este lenguaje y aplicarlo
- 4) Nombra y explica al menos otras dos formas de realizar la semántica para este lenguaje. Pon un ejemplo de cada tipo. (1 punto)

- 1) La semántica de un lenguaje de programación es el significado que se da a las diversas construcciones sintácticas. (pág 46)

La sintaxis proporciona reglas para escribir enunciados en un lenguaje de programación, mientras que la semántica aporta significado a esos enunciados, que es utilizado en la etapa de traducción.

- 2) Algunas de las funciones más comunes de un analizador semántico son:  
Mantenimiento de la tabla de símbolos,  
Inserción de información implícita,  
Detección de errores,  
Procesamiento de macros y operaciones en tiempo de compilación,  
(pág. 75, 76)

3.1) la sintaxis vendría dada por la siguiente gramática BNF:

expresión  $\rightarrow$  termino | expresión + termino

termino  $\rightarrow$  factor | termino \* factor

factor  $\rightarrow$  entero | (expresión )

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

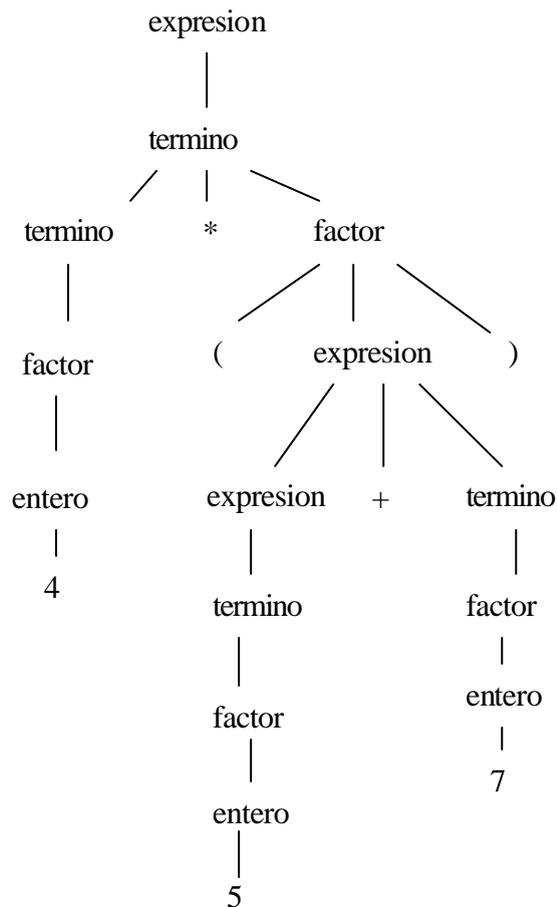
JUNIO DE 2002

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**entero  $\rightarrow$  numero | -numeronumero  $\rightarrow$  digito | numero digitodigito  $\rightarrow$  0 | 1|2|3|4|5|6|7|8|9

3.2) Para reconocer los elementos de un lenguaje, dada su sintaxis, se emplean los *árboles sintácticos*, los cuales se construido mediante *derivaciones*.(p. 84) De esta forma un elemento es reconocido por el lenguaje si se puede derivar a partir del símbolo inicial. Por ejemplo, el elemento  $4 * ( 5 + 7 )$  va a ser derivado a partir de la producción que define el símbolo no terminal expresión.

expresión  $\Rightarrow$  termino  $\Rightarrow$  termino \* factor  $\Rightarrow$  factor \* factor  $\Rightarrow$  entero \* factor  $\Rightarrow$  4 \*  
 ( expresión )  $\Rightarrow$  4 \* (expresión + termino)  $\Rightarrow$  4 \* ( termino + termino)  $\Rightarrow$  4 \* ( factor + factor)  $\Rightarrow$  4 \* ( entero + entero)  $\Rightarrow$  4 \* ( 5 + 7)

el árbol quedaría:



**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

3.3) Existen varios modelos para definir la semántica de un lenguaje. Uno de ellos es la gramática de atributos. Los atributos pueden ser sintetizados o heredados, dependiendo si son función de símbolos no terminales de más abajo del árbol o de más arriba, respectivamente.

Para cada producción de la gramática definida en el apartado 1 se va a definir el atributo valor:

Producción	Atributo
$\text{expresión} \rightarrow \text{expresión} + \text{termino}$	$\text{valor}(\text{expresión}) = \text{valor}(\text{expresión}) + \text{valor}(\text{termino})$
$\text{expresión} \rightarrow \text{termino}$	$\text{valor}(\text{expresión}) = \text{valor}(\text{termino})$
$\text{termino} \rightarrow \text{termino} * \text{factor}$	$\text{valor}(\text{termino}) = \text{valor}(\text{termino}) * \text{valor}(\text{factor})$
$\text{termino} \rightarrow \text{factor}$	$\text{valor}(\text{termino}) = \text{valor}(\text{factor})$
$\text{factor} \rightarrow \text{entero}$	$\text{valor}(\text{factor}) = - \text{valor}(\text{entero})$
$\text{factor} \rightarrow ( \text{expresión} )$	$\text{valor}(\text{factor}) = \text{valor}(\text{expresión})$

4) (pág 99,..)

Como en el caso de la sintaxis, se requiere de algún método formal para proporcionar una definición legible, precisa, y concisa de la semántica de un lenguaje completo. Además de la gramática de atributos, se han desarrollado (sólo dos):

- Modelos gramaticales: Las gramáticas de atributos es un tipo de modelo gramatical, luego esta respuesta no es válida
- Modelos imperativos u operativos.  
Están basados en la idea de máquina virtual. Estos modelos utilizan un autómata que implementa una máquina virtual con sus registros, código ejecutable, flags

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

de condición. Se usa un conjunto de operaciones formalmente definidas que especifica cómo cambia el estado interno. También especifica cómo se traduce el código a un estado inicial de la máquina, del cuál, aplicando las operaciones formales, se llega a un estado final. Una definición operativa puede constituir la base para elaborar un intérprete del lenguaje. Un ejemplo de este modelo es el Lenguaje de la Definición de Viena (VDL; Viena Definition Language). Amplía el árbol de sintaxis para incluir también el estado de la máquina, de forma que un estado de cómputo es el árbol de programa, pero también el árbol de la máquina.

- Modelos aplicativos.

Un modelo aplicativo construye una función que calcula cada enunciado del programa. Esta función se construye de forma jerárquica a través de la definición de la función que calcula cada construcción. Así, cada operación primitiva y cada función definida por el programador es una función. Las estructuras de control integran estas funciones en otra mayor: las series de enunciados y las estructuras condicionales se representan como funciones de sus componentes individuales. La iteración se define como una función recursiva construida a partir del cuerpo de la iteración. Al final se obtiene un modelo funcional de todo el programa.

El método de semántica denotativa de Scott y Strachey y el método de semántica funcional de Mills son ejemplos de este enfoque.

- Modelos axiomáticos:

Define la semántica mediante axiomas y reglas de inferencia que se pueden usar para deducir el efecto de la ejecución de esa construcción.

A partir del supuesto inicial de las variables de entrada satisfacen ciertas restricciones, se usan los axiomas y reglas de inferencia para deducir las restricciones que satisfacen las variables para cada enunciado del programa. Al final, el método verifica que los resultados del programa satisfacen las restricciones.

El método de semántica axiomática desarrollado por Hoare es un ejemplo de este método.

- Modelos de especificación:

En el modelo de especificación se describe la relación entre las diversas funciones que implementan un programa. Cuando se demuestra que la implementación obedece esta relación, la implementación es correcta respecto a la especificación.

El tipo algebraico de datos es una forma de especificación formal. Por ejemplo al construir un programa que implementa pilas se establece el siguiente axioma:  $\text{desempilar}(\text{empilar}(S,x))=S$

**PREGUNTA 3. Tiempo estimado 50 minutos****3,5 puntos**

En un lenguaje de programación se quiere definir un tipo de datos estructurado que permita almacenar y gestionar un número variable de elementos, todos ellos con la misma estructura, que vamos a llamar VARIOS. Por ejemplo, el número de empleados de una empresa, todos los niños de un aula, los coches que están aparcados en un aparcamiento, etc.. Para ello, independientemente del paradigma de programación utilizado (es decir, para cualquier paradigma de los conocidos)

Decir cuáles serían los principales atributos necesarios que definen un tipo estructurado de datos. Aplicar estos atributos al tipo de datos estructurado VARIOS. (1 punto)

Indicar y clasificar el conjunto de operaciones necesarias para definir un tipo de datos estructurado. Definir esas operaciones (es suficiente con dos para cada grupo) para el tipo de datos estructurado VARIOS. (1 punto)

- 1) En la implementación de un tipo de datos estructurado hay que tomar varias operaciones para que sea una estructura eficiente y adecuada al uso que se le va a dar. Nombrar y explicar dos modos de almacenamiento que se pueden utilizar para representar el tipo de datos estructurado VARIOS, explicando tres ventajas que ofrece cada una de ellas así como tres problemas que presenta para el programador de la estructura de datos así como para el usuario que luego la utilizará en sus programas. (1,5 puntos)

**Solución.**

- 1) Atributos para especificar una estructura de datos: (página 143 del libro)
  1. Número de componentes: tamaño fijo o tamaño variable
  2. Tipo de cada componente: estructura homogénea (todos los elementos son del mismo tipo) o estructura heterogénea (los componentes son de tipos distintos)
  3. Nombres que se usan para seleccionar los componentes: Mecanismo de selección para identificar componentes individuales de la estructura de datos
  4. Número máximo de componentes. Depende de 1
  5. Organización de los componentes. Lineal, multidimensional.

Para la estructura de datos VARIOS

1. Variable
2. Estructura homogenera  
El tipo se da por otra estructura de datos, que dependiendo del lenguaje será un registro, otro tipo de datos o una clase  
Elegimos para juntarlos una lista encadenada
3. Un procedimiento, función o método (depende del lenguaje utilizado) llamado GET\_ITEM que dada una clave nos devuelva elemento buscada  
Además, si queremos que la lista sea una pila podemos definir un GET que devuelve el primero de la lista
4. Es variable.
5. Es lineal.

- 2) Las operaciones se pueden clasificar en cuatro grupos (página 144 del libro)

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

---

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

---

1. Operaciones de selección de componentes
2. Operaciones con estructuras de datos complejas
3. Inserción/eliminación de componentes
4. Creación/destrucción de estructuras de datos

Para la estructura de datos VARIOS

1. GET\_ITEM -> busca elemento dato. Habrá que definir qué criterio, por ejemplo por posición, por una palabra clave (cada opción podría dar lugar a una función diferente)

GET → devuelve el primero

2. Para producir estructuras de datos más complejas o manejarlas

CONVERT → convierte VARIOS a un ARRAY, por ejemplo

CONSTRUCT → dada una lista de elementos de tipo elegido construye la lista

3. INSERT (ELEM, POS) -> insertar un elemento del tipo en la posición indicada, lo mismo para DEL(ELEM, POS)

4. CREATE → crea la estructura de datos vacía

DEL (POS) → borra el elemento de la posición indicada

3) Las representaciones básicas de almacenamiento para una estructura de datos son

- Representación secuencial, la estructura de datos se guarda en un solo bloque contiguo de almacenamiento que incluye tanto descriptor como componentes. Ejemplo: tablas o arrays
- Representación enlazada (o vinculada), se guarda en bloques contiguos de almacenamiento, enlazados con punteros. Ejemplo, listas de punteros

Para la estructura de datos VARIOS

a. Array,

Ventajas:

- i. fácil de implementar,
- ii. fácil de borrar, insertar
- iii. fácil de acceder

Inconvenientes:

- iv. Algunos LP permiten definir arrays de tamaño variable otros no
- v. Cuando se elimina un elemento queda el hueco libre.
- vi. Se reserva todo el espacio.

b. Lista enlazada

Ventajas:

- i. Flexibilidad en el manejo de la memoria,
- ii. no depende del lenguaje para tener una estructura de datos variable
- iii. Ahorro de espacio.

Inconvenientes:

- iv. debe ser cuidadoso con el manejo de enlaces; problemas con basura y referencias desactivadas.
- v. para insertar o buscar un elemento tiene que disponer de

**EXAMEN DE LENGUAJES DE PROGRAMACIÓN**

JUNIO DE 2002

---

**NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL**

---

- mecanismos eficientes para recorrer las listas,  
vi. debe liberar memoria cuando se elimina un elemento.

Al usuario de la estructura de datos, si está bien implementada, le da igual como esté porque él sólo se limita utilizarla.



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN  
SEPTIEMBRE 2002

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

PREGUNTA 1. Tiempo estimado 40 minutos

4 puntos

Dadas las siguientes clases escritas en el lenguaje Java:

```
abstract class Abeja {
    protected int id;
    public Abeja(int n){
        id = n;
    }
    public abstract Abeja reproducir(Abeja otraAbeja);
}

class Obrera extends Abeja {
    public Obrera(int n) {
        super(n);
    }
    public Abeja reproducir(Abeja otraAbeja){
        return (Abeja) null;
    }
}

class Reina extends Abeja {
    int numAbejasEnjambre=0;
    public Reina(){
        super(0);
        numAbejasEnjambre++;
    }
    public Abeja reproducir(Abeja otraAbeja) {
        return new Obrera(numAbejasEnjambre + otraAbeja.id);
    }
    public static void main(String args[]){

        Abeja reina = new Reina();
        Abeja otra = reina.reproducir(new Obrera(54));
        System.out.println(otra.id);
    }
}
```

Se pide:

- Explica en qué consiste la herencia de clases y qué beneficios aporta. Explica su semántica en las sentencias del enunciado. (1 punto).
- ¿Qué tipo de métodos son `Reina()` y `Obrera(int n)`? Explica cómo están codificados en el ejemplo de arriba. (1 punto)
- ¿Qué tipo de método es `reproducir`? Explica la semántica de este tipo de funciones. (0,5 puntos).
- Si ejecutamos el programa, ¿qué imprime?. Explica por qué y enumera qué propiedades de la Programación Orientada a Objetos que están involucradas. (1.5 punto)

Solución:

- (1 punto) La herencia de clases consiste en definir nuevos tipos de datos en base a otros tipos de datos, heredando aquellos las propiedades de éstos. La clase que hereda se denomina subclase o clase extendida. La clase padre se llama superclase o clase base. La herencia facilita la reutilización/reusabilidad del código.



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN  
SEPTIEMBRE 2002

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

En el enunciado la superclase es Abeja. Las clases Obrera y Reina heredan los métodos y miembros de la sección pública y protected, en este caso reproducir y el miembro estado. Los constructores no se heredan.

- b) (1 punto) Son constructores. Un constructor sirve para asignar un nueva zona de memoria al objeto que instancia. El constructor Reina() llama al constructor de la clase Abeja con el parámetro 0, mediante la palabra clave super. El de la clase Obrera está definido de una manera similar.
- c) (0.5 punto) Se trata de un método virtual (en Java todos los métodos son virtuales). Este tipo de funciones no se enlazan hasta el momento en el que se determina la clase a la que pertenece y/o la clase de sus parámetros (polimorfismo). (Circunstancialmente, reproducir es definido en la clase Abeja como método abstracto, lo que transfiere a Abeja la propiedad de clase abstracta).
- d) (1.5 punto) A pesar de que reina es declarado como Abeja, se instancia con un constructor de la clase Reina (**herencia** y **polimorfismo**). La invocación del constructor de Obrera devuelve un objeto Obrera con id igual a 54, que es un miembro heredado (**herencia**) de la clase base Abeja. Finalmente, la invocación de reproducir (**polimorfismo** de clase y de método) concluye devolviendo un objeto Obrera, que también es de la clase Abeja –nuevamente **polimorfismo**- con id igual a 55.

## PREGUNTA 2. Tiempo estimado 40 minutos

Los lenguajes de programación generalmente proporcionan mecanismos de definición de tipos de datos estructurados de longitud fija, como son los registros y los vectores.

- a) Explica cada uno y enumera sus diferencias . (1 punto).
- b) Se quiere definir en un lenguaje de programación un tipo de datos estructurado de número un número fijo de elementos, con la estructura siguiente para cada elemento:
- tipo de vehículo: coche, barco, avión.
  - Peso, propietario, y precio del vehículo,
  - Para el avión: Peso, extensión de las alas, y número de motores.
  - Para el automóvil: Número de ruedas y número de puertas.
  - Para el barco, la longitud y la velocidad en nudos que es capaz de desarrollar. Define este tipo de datos con las estructuras de datos adecuada, justificando por qué has elegido cada una. (1 punto)
- c) Explica qué es un descriptor . Representa el aspecto que tendría el descriptor correspondiente al vector del apartado b). (1 punto)

Solución:

a) (1 punto) Vector:

- gestiona elementos del mismo tipo;
- accesibles a través de un índice;



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN  
SEPTIEMBRE 2002

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

- la dirección del elemento seleccionado puede que no se conozca en tiempo de compilación, ya que puede ser una expresión que depende de una variable. Para poder realizar comprobaciones en tiempo de ejecución se requiere guardar algún tipo de información, como límite inferior y superior y tamaño de los elementos.

Registro:

- gestiona elementos de distinto tipo;
- accesible a través de identificadores;
- no es necesario almacenar ninguna información adicional.

- b) (1 punto) Dado que no se especifica ninguna restricción que obligue a utilizar un paradigma u otro vamos a adoptar la solución más simple que aporta la solución al problema. La estructura más conveniente para este caso sería una vector de registros de campos variantes. Debido a que el número de elementos es conocido y fijo, debemos emplear un vector. El tipo de cada componente sería un registro variante:

```
Const Integer MaxVehiculos=20;
Type TipoVehiculo={coche,barco,avion};
Type Vehiculos =array[0.. MaxVehiculos] of record
    Tipo:TipoVehiculo;
    peso: real;
    propietario: array [1..20] of char;
    Case tipo:TipoVehiculo of
        avion:(ext, num: integer);
        coche:(numru, numpu:integer);
        barco:(long, velocidad:real);
    end;
```

- c) (1 punto) Un descriptor es una estructura que proporciona información al compilador sobre el tipo de datos que representa en tiempo de compilación o incluso en tiempo de ejecución.

En el ejemplo del apartado b), un descriptor del tipoVehiculos tendría el aspecto:

estructura	descriptor
Limite Inferior	0
Limite Superior	20
Tipo	registro
Tamaño tipo	Tamaño de registro en bytes
componentes	Registro1
	...
	Registron



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN  
SEPTIEMBRE 2002

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

### PREGUNTA 3. Tiempo estimado 40 minutos

- a) (0.5) Verificar un programa es comprobar que efectúa la función para la que se había diseñado.
- b) (1) La corrección de un programa se puede abordar desde varias aproximaciones:
- 1 Extrayendo su especificación semántica: modelos gramaticales, axiomáticos.
  - 2 Desarrollando el programa a partir de su especificación (ingeniería del software)
  - 3 Empleando modelos que comprueben que la función de su especificación coincide con la del programa (cálculo de predicados).
- c) (1.5) Un programa correcto efectúa la función para la que había sido diseñado, lo cual se ha podido comprobar mediante cualquiera de las formas descritas arriba.

```
Función Multiplicar (A:entero, B:entero)
    acum:entero;
    acum=0;
    Mientras que B >0 hacer
        acum=acum + A;
        B=B-1;

    FMQ
    Devolver acum.
Fin
```

Un programa depurado ha llegado a ser correcto empleando técnicas de ingeniería del software.

```
Función Multiplicar (A:entero, B:entero)
    acum:entero;
    acum=0;
    Mientras que B >= 0 hacer // Incorrecto, debería ser B>0
        acum=acum + A;
        B=B-1;

    FMQ
    Devolver acum.
Fin
```

Un programa optimizado presenta mejores tiempos de ejecución que uno no optimizado, ya que utiliza técnicas de eficiencia (desenrollado de bucles, almacenamiento de cálculos intermedios, ...) o particularidades del hardware (unidades aritméticas específicas, registros vectoriales, ...).

```
Función Multiplicar (A:entero, B:entero)
    A*B
Fin
```

Aquí se hace uso de la operación de multiplicación disponible en el hardware.

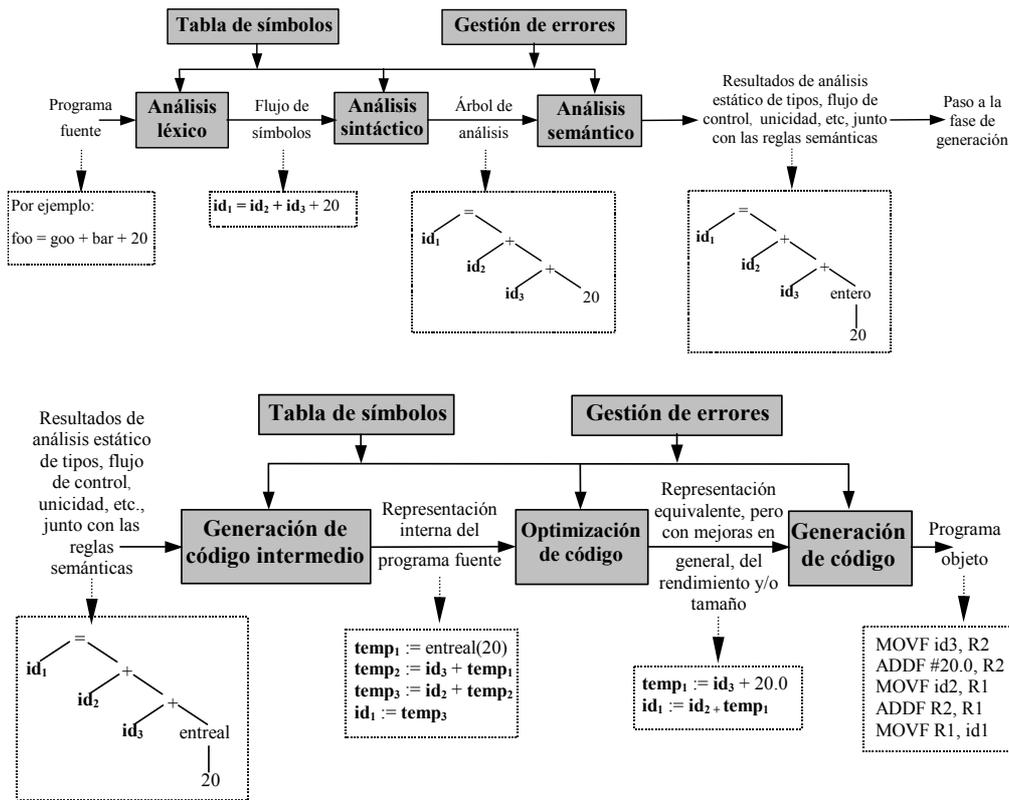
Segunda semana de junio de 2003

PREGUNTA 1. Tiempo estimado 30 minutos

PUNTUACIÓN 3 puntos

1. Dibuja la estructura de un compilador. Explica brevemente en qué consiste el análisis léxico. ¿En qué se diferencia el análisis léxico del análisis sintáctico? (0,5 puntos)

Se puede ver la estructura de un compilador a continuación:



**Análisis Léxico.** En la fase de análisis léxico se leen los caracteres del programa fuente y se agrupan en cadenas que representan los componentes léxicos. Cada componente léxico es una secuencia lógicamente coherente de caracteres relativa a un identificador, una palabra reservada, un operador o un carácter de puntuación. A la secuencia de caracteres que representa un componente léxico se le llama lexema (o con su nombre en inglés, token). En el caso de los identificadores creados por el programador, no sólo se genera un componente léxico, sino que se genera otro lexema en la tabla de símbolos.

**Análisis Sintáctico.** En esta fase, los componentes léxicos se agrupan en frases gramaticales que el compilador utiliza para sintetizar la salida.

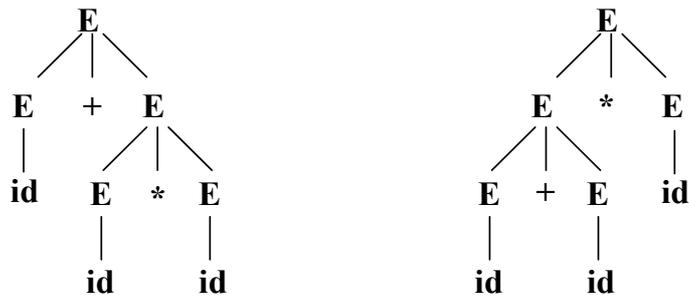
2. ¿Cómo se añadiría un nuevo tipo a un lenguaje de programación? ¿A qué partes del compilador afectaría? (1 punto)

Para añadir un nuevo tipo a un lenguaje de programación cualquiera, hay que modificar la gramática que lo define. No es suficiente contemplar la incorporación de dicho tipo a un lenguaje en términos de manipulación de los tipos abstractos que puede contener un lenguaje dado (como por ejemplo, el "struct" de C), porque no todos los lenguajes disponen de estos mecanismos. Dicha gramática es la definición formal de la sintaxis del lenguaje y está compuesta por un conjunto de reglas que especifican las series de caracteres permisibles. Por lo tanto, un cambio en la gramática va a requerir cambios en el analizador léxico, sintáctico y semántico y la generación de código intermedio, y también, en la tabla de símbolos y gestión de errores.

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

3. ¿A qué se debe la ambigüedad en las construcciones de un lenguaje? Pon un ejemplo. ¿Puede haber ambigüedad siendo una construcción sintácticamente correcta? (1 punto)

La ambigüedad de las construcciones de un lenguaje se debe a la una gramática ambigua, es decir, una en la que existe más de una interpretación sintáctica correcta para una entrada dada, por ejemplo, dada la entrada **id + id \* id**:

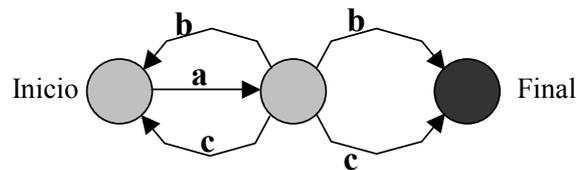


Interpretación sintáctica #1

Interpretación sintáctica #2

4. Dibuja el autómata de estados finitos que reconozca las cadenas representadas por la expresión:  
 $a((b \vee c)a)^*(b \vee c)$  (0,5 puntos)

Utilizando un autómata de estados finitos no determinístico:



## PREGUNTA 2. Tiempo estimado 40 minutos

PUNTUACIÓN 3 puntos

1. ¿Cuáles son las dos maneras básicas en que un elemento de datos está disponible para operaciones en un programa? Ilustre la explicación con fragmentos de código (1 punto).

Las dos maneras básicas de acceder a un elemento de datos son por nombre y por referencia. Se accede al valor de un elemento de datos a través de su nombre como evaluación real del elemento. Y cada acceso requiere una re-evaluación de dicho elemento. El paso de un elemento de datos por su nombre a una función pasaría una copia que existiría localmente dentro de la función; los cambios a la copia no cambiarían el elemento original. Por ejemplo:

```
main() {
    int i = 0;
    printf("Empezando main : i = %d", i);
    printf("Llamando goo ahora\n");
    goo(i);
    printf("Después de goo\n");
    printf("Termando main : i = %d", i);
}

goo(int x) {
    printf("Empezando goo: x = %d ", x);
    x ++;
    printf("Terminando goo: x = %d\n", x);
}
```

Y se accede a un elemento de datos a través de una referencia a la localización de memoria donde está almacenado su valor. El paso de un elemento de datos por su nombre a una función pasaría un acceso a la posición de memoria; los cambios a la copia cambiarían el elemento original. Por ejemplo:

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

```
main() {
    int j = 0;
    printf("Empezando main : j = %d\n", j);
    printf("Llamando hoo ahora\n");
    hoo(&j);
    printf("Después de hoo\n");
    printf("Termando main : j = %d\n", j);
}

hoo(int *y) {
    printf("Empezando hoo: y = %d\n", *y);
    (*y)++;
    printf("Terminando hoo: y = %d\n", *y);
}
```

2. Dado el siguiente programa:

```
main() {
    int i = 0;
    int j = 0;
    printf("Empezando main : i = %d, j = %d\n", i, j);
    printf("Llamando foo ahora\n");
    foo(i, &j);
    printf("Después de foo\n");
    printf("Termando main : i = %d, j = %d\n", i, j);
}

foo(int x, int *y) {
    printf("Empezando foo: x = %d, y = %d\n", x, *y);
    x++;
    (*y)++;
    printf("Terminando foo: x = %d, y = %d\n", x, *y);
}
```

a. ¿Para qué sirven los nombres seudónimos para objetos de datos y cuáles son los problemas relacionados con su uso? (0,5 punto).

Los nombres seudónimos sirven para acceder al mismo objeto de datos con nombres distintos en el mismo ambiente de referencia. Hay varios problemas con su utilización: dificultan la comprensión del código, puede existir interdependencia entre los seudónimos, no es posible recortar o eliminar partes del código y es más difícil para el compilador optimizar el código.

b. Demuestra la utilización de los nombres seudónimos y la utilización de memoria en el código anterior (1 punto).

En la llamada a la función foo, tenemos dos nombres de variable que se refieren a la misma posición de memoria: j e (\*y) son dos identificadores diferentes para la misma posición de memoria &j. Como ambas afectan a la misma posición de memoria las modificaciones que se produzcan a través de un identificador se ven recogidas igualmente a través del otro. Dentro de foo se incrementa (\*y) y, por tanto, a la salida de foo tendremos que también se ha incrementado el valor de j.

Si j fuera una variable accesible desde foo entonces tendríamos que tanto j como (\*y) accederían al mismo objeto de datos con nombres distintos y en el mismo ambiente de referencia ( la función foo), es decir, serían pseudónimos.

La estructura de memoria del código sería:

Segmento de código de main
Segmento de código de foo
Registro de activación de main
Registro de activación de foo
Pila
Heap

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

Donde los segmentos de código para main y foo serían respectivamente:

Prólogo para crear registro de activación de main	Prólogo para crear registro de activación de foo
Código ejecutable de main	Código ejecutable de foo
Epílogo para borrar registro de activación de main	Epílogo para borrar registro de activación de foo

Los registros de activación para main y foo serían respectivamente:

Punto de retorno y otros datos del sistema	Punto de retorno y otros datos del sistema
&i i: objeto local de datos	&x x: parámetro
&j j: objeto local de datos	&y y: parámetro

Valores l y r de los elementos de datos involucrados:

Identificador de variable	Posición de memoria (ubicación, valor l)	Valores que toma (valores r)
i	&i	0
j	&j	0,1
x,	&x	0,1
y, es un puntero a &j	&y	&j
(*y), contenido de &j	&j	0,1

c. Extiende el código anterior para ilustrar los problemas relacionados con los nombres seudónimos (0,5 puntos).

Si dentro de foo se hubiera utilizado también j el programa sería difícil de entender. Por ejemplo:

```
int j = 0;
main() {
    int i = 0;
    printf("Empezando main : i = %d, j = %d\n", i, j);
    printf("Llamando foo ahora\n");
    foo(i, &j);
    printf("Después de foo\n");
    printf("Termando main : i = %d, j = %d\n", i, j);
}

foo(int x, int *y) {
    printf ("Empezando foo: x = %d, y = %d\n", x, *y);
    j = 1;
    x ++;
    (*y) ++;
    printf ("Terminando foo: x = %d, y = %d\n", x, *y);
}
```

Parece que la función foo tiene como resultado incrementar en uno el valor de (\*y). Pero esto no siempre ocurre así: la llamada foo(i, &j) tiene como efecto asignar a j el valor 2.

### PREGUNTA 3. Tiempo estimado 50 minutos

**PUNTUACIÓN 4 puntos**

Se desea implementar un programa que reciba una URL, una fecha y una palabra, y devuelva la URL de un documento con fecha posterior a la de entrada y que contenga la palabra dada. El proceso es el siguiente:

- Introducir la URL inicial en una cola de candidatos.

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

- Mientras la cola no esté vacía, extraer la primera URL de la cola.
- Si la URL no se ha explorado con anterioridad, entonces pedir la fecha del documento, el tipo de formato del documento y descargar el fichero correspondiente.
- Si el documento tiene una fecha adecuada y contiene la palabra de entrada, entonces devolver la URL y terminar.
- Si no, extraer los enlaces del documento e introducirlos en la cola de candidatos.

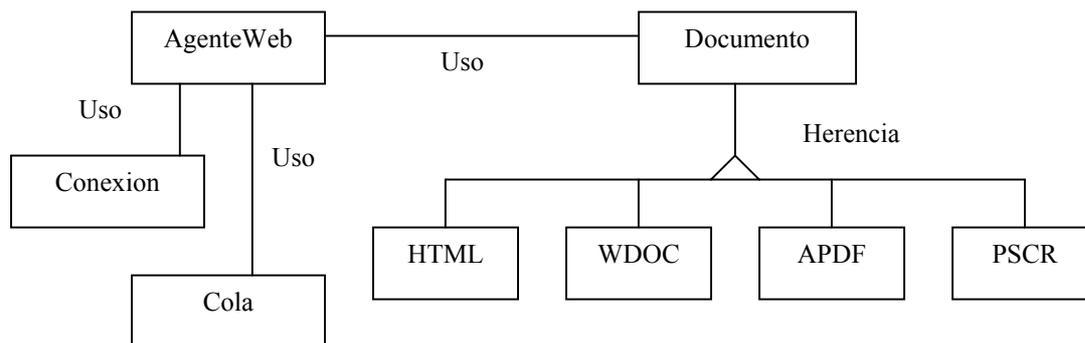
Los documentos pueden tener cuatro formatos diferentes: HTML, WDOC, APDF o PSCR. El proceso de comprobar la presencia de una palabra y el proceso de extraer la lista de enlaces depende del formato del documento.

Dentro del paradigma orientado a objetos, se pide:

1. Identificar las clases necesarias para implementar el sistema. Dibujar el esquema global de organización de estas clases, mostrando claramente los tipos de relaciones presentes entre las clases. (1,5 puntos)

Existen varias aproximaciones posibles. Aquí se desarrolla una versión en la que la cola contiene únicamente las URLs tal como sugiere el enunciado. Se deja como ejercicio el desarrollo de una versión en la que los elementos de la cola fueran de la clase Documento y la *url* fuera un atributo de dicha clase.

Es necesario una clase principal que en este caso hemos denominado *AgenteWeb*. El método que implementa la funcionalidad pedida se denomina *busca* y pertenece a esa clase. Este método hace uso de objetos de clase Conexión, Cola y Documento, así como las clases hijas de éste.



2. Dar la especificación de estas clases en un lenguaje orientado a objetos. (1 punto)

```
import java.util.HashSet;
import java.util.Vector;

public class AgenteWeb {

    public AgenteWeb() {}

    public String busca (String primeraURL, String fecha, String palabra) {
        String url, fechaURL, formato, fichero;
        Cola colaURLs = new Cola();
        HashSet explorados = new HashSet();
        Conexion conn = new Conexion();
        Vector listaURLs = new Vector();
        Documento doc;
        colaURLs.insertar(primeraURL);
        while (!colaURLs.esVacía()) {
            url = colaURLs.extraerPrimero();
            if (!explorados.contains(url)) {
                explorados.add(url);
                fechaURL = conn.fecha(url);
                formato = conn.fecha(url);
                fichero = conn.fichero(url);
                if (formato.equals("html")) {
                    doc = new HTML(fichero);
                }
                else if (formato.equals("wdoc")) {
```

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

```
        doc = new WDOC(fichero);
    }
    else if (formato.equals("apdf")) {
        doc = new APDF(fichero);
    }
    else if (formato.equals("pscr")) {
        doc = new PSCR(fichero);
    }

    if (posterior(fechaURL, fecha) &&
        doc.existePalabra(palabra) ) {
        return url;
    } else {
        listaURLs = doc.extraerURLs();
        colaURLs.insertar(listaURLs);
    }
}
}
return null;
}

private static boolean posterior(String fecha1, String fecha2) {}

public abstract class Documento {
    public Vector extraerURLs() {}
    public boolean existePalabra(String pal) {}
}

public class HTML extends Documento {
    public HTML (String fichero) {}
    public Vector extraerURLs() {}
    public boolean existePalabra(String pal) {}
}

public class WDOC extends Documento {
    public WDOC (String fichero) {}
    public Vector extraerURLs() {}
    public boolean existePalabra(String pal) {}
}

public class APDF extends Documento {
    public APDF (String fichero) {}
    public Vector extraerURLs() {}
    public boolean existePalabra(String pal) {}
}

public class PSCR extends Documento {
    public PSCR (String fichero) {}
    public Vector extraerURLs() {}
    public boolean existePalabra(String pal) {}
}

public class Cola {
    public Cola () {}
    public void insertar (String elemento){}
    public void insertar (Vector elementos) {}
    public boolean esVacia () {}
    public String extraerPrimero () {}
}

public class Conexion {
    public Conexion () {}
    public String fecha (String url) {}
    public String formato (String url) {}
    public String fichero (String url) {}
}
```

3. Implementar la función principal. (1 punto)

(Ver método busca en la sección anterior).

NO ESTÁ PERMITIDO EL USO DE NINGÚN MATERIAL ADICIONAL

---

4. ¿Qué partes del programa sería conveniente paralelizar? (0,5 puntos)

La parte más lenta es la conexión para descargar los datos y contenido del documento asociado a la URL porque depende de la velocidad de la red. Es un tiempo que puede dedicarse a otras tareas. Por ejemplo, implementar una versión que descargue varios documentos simultáneamente, mientras se realiza la búsqueda de una palabra y la extracción de enlaces de un documento.



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN

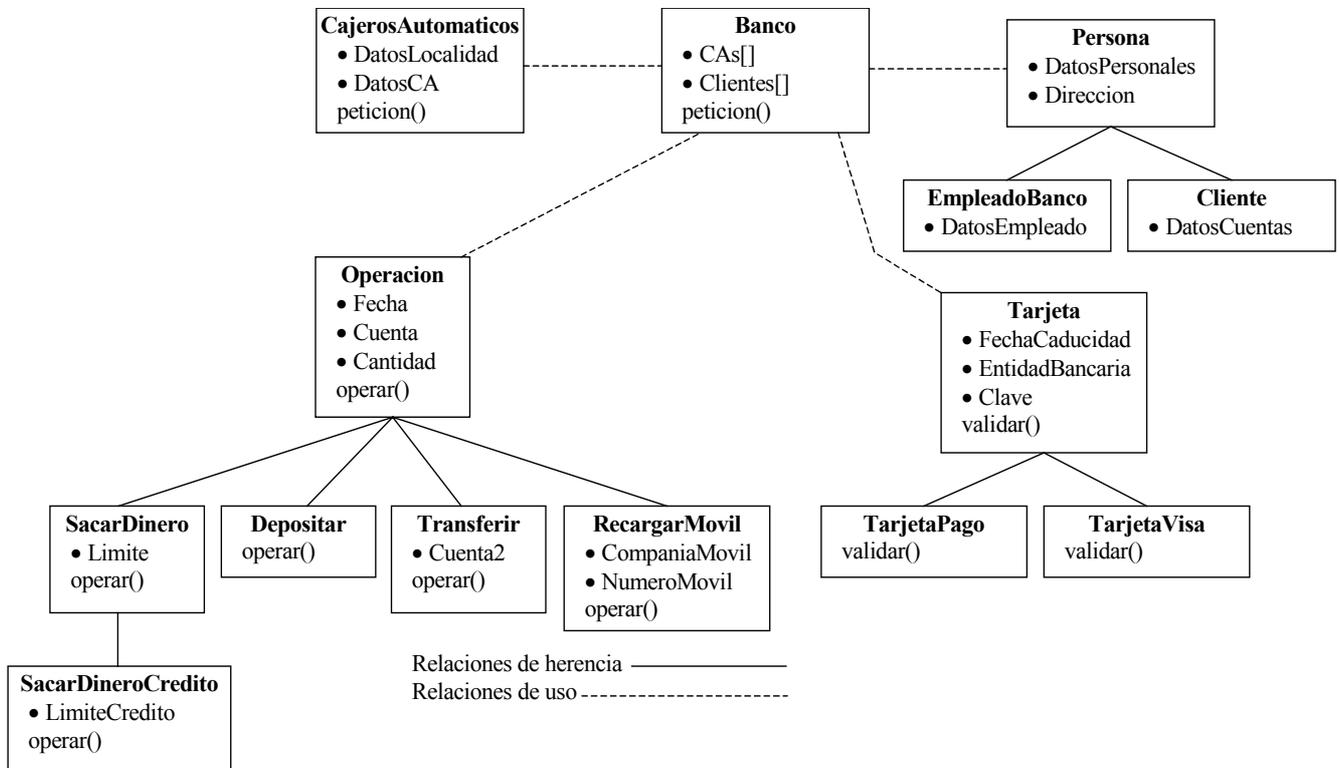
JUNIO 2004 - SEGUNDA SEMANA - SOLUCIÓN

PREGUNTA 1

4 puntos

1. Se pueden distinguir cuatro tipos de factores que hay que tener en cuenta a la hora de diseñar un sistema de este tipo:
- **Factores relacionados con el rendimiento de la red:** el sistema tiene que funcionar en tiempo real, sin fallos, etc., así que tiene que haber distintos caminos entre los componentes distribuidos del sistema.
  - **Factores relacionados con la arquitectura de la base de datos:** la base de datos tiene que estar distribuida entre las distintas localidades en las que hay sucursales del banco, así que tiene que haber mecanismos para garantizar la comunicaciones entre cualquier CA y cualquier parte de la base de datos. También tiene que haber un mecanismo de transacciones que asegure el estado de la base de datos en cualquier momento.
  - **Factores relacionados con las conexiones:** las conexiones entre los CA y las bases de datos dentro de las sucursales tienen que usar comunicaciones seguras y cifradas para evitar pérdidas de datos o evitar que los hackers puedan conseguir los datos de cuentas de los clientes.
  - **Factores relacionados con la arquitectura general del sistema:** hay distintas arquitecturas que se pueden usar para este tipo de sistema, como por ejemplo el modelo servidor – cliente o un sistema completamente distribuido. En este caso se podría usar un modelo servidor – cliente, lo que conllevaría un conjunto de factores de diseño como por ejemplo, la tecnología que se va a usar para el sistema (Java, PHP, Perl, etc.).

2. Se pueden ver el esquema global de organización de las clases a continuación. Nota: no es estrictamente necesario tener tres jerarquías de herencia (para Persona, Operación y Tarjeta), se pueden haber colapsado en una o dos. Se ha hecho así en esta figura para demostrar todas las opciones disponibles (para que sea completo).



3. No se incluye la especificación de estas clases en un LPOO porque el alumno tiene la práctica y las soluciones de preguntas parecidas de exámenes de otros años para comprobar como se hace.



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN

JUNIO 2004 - SEGUNDA SEMANA - SOLUCIÓN

4. La idea fundamental de un diseño OO es que los cambios a la especificación del problema (como la adición de los puntos de venta VISA) no deberían cambiar fundamentalmente el esquema de clases, sino implicar la adición de otras pocas clases. En este caso, como se puede ver en la figura anterior, la adición de un procesamiento de tarjetas VISA implica:

- Una clase TarjetaVisa, para encapsular los datos y procesos de validación VISA
- Una clase SacarDinerCredito, para encapsular la operación de la tarjeta VISA
- La definición de los puntos de venta como instancias de los CajerosAutomaticos

Y como los métodos validar() y operar() son abstractos, no es necesario modificar las demás clases.

## PREGUNTA 2

3 puntos

1. ¿En qué se diferencian las gramáticas BNF y las gramáticas libres de contexto? (0,5 puntos)

Únicamente en la notación que utiliza cada una (página 81).

2. Defina una gramática BNF no ambigua que reconozca cualquier expresión aritmética con los operadores de asignación, suma, resta y producto y división asumiendo el orden de precedencia usual entre operadores (1 punto).

La gramática debe garantizar que una expresión como  $3*4+5$  se evalúa correctamente, es decir, primero se realiza el producto ( $3*4$ ) y luego la suma ( $12+5$ ). Cualquier gramática ambigua que permita evaluar primero la suma y después realizar el producto es una gramática incorrecta. Por tanto, las operaciones pedidas no pueden estar en el mismo nivel. Por ejemplo, reglas como la siguiente invalidan el ejercicio:

```
<expresión> ::= <variable> |  
             <expresión>+<expresión> |  
             <expresión>-<expresión> |  
             <expresión>*<expresión> |  
             <expresión>/<expresión>
```

En la página 84 (figura 3.3) hay un ejemplo de la gramática que se pide. En la página 101 hay otro ejemplo aunque no en notación BNF.

3. Producir una gramática de atributos para la gramática definida en el punto anterior (1 puntos).

En la misma página 101 del libro existe un ejemplo del atributo que habría que asociar a cada regla para que se evaluaran las expresiones reconocidas por cada una de las reglas de la gramática.

4. Escriba la gramática del punto 1 en notación BNF extendida de la manera más sintética posible (0,5 puntos)

En la página 88 (figura 3.7) se muestra un ejemplo de la gramática que se pide.



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN

JUNIO 2004 - SEGUNDA SEMANA - SOLUCIÓN

PREGUNTA 3

PUNTUACIÓN 3 puntos

1. (1 punto) ¿En qué consiste la sustitución?

Es el resultado de aplicar nuevos valores a los argumentos de un patrón (o plantilla).

¿En qué consiste la unificación general?

Es el resultado de realizar una o varias sustituciones simultáneas en varios patrones (o plantillas) para demostrar que son equivalentes bajo cierto conjunto de sustituciones simultáneas.

¿En qué se diferencian?

En la sustitución se tiene un único patrón y un ejemplar y se trata de dar nuevos nombres a los parámetros del patrón con los valores reales del ejemplar.

En la unificación se tienen varios patrones y un ejemplar y se trata de realizar sustituciones simultáneas en los patrones con los valores reales del ejemplar, de forma que al final los patrones sean iguales o equivalentes.

2. (1 punto) ¿Cómo se realiza la concordancia de patrones en Prolog?

Mediante unificación: las consultas se unifican con reglas o con hechos de la base de datos hasta que el resultado sea “cierto”. Si la unificación no es posible entonces el resultado es “falso”.

¿Es posible unificar los siguientes predicados? ¿Cuál sería el efecto en cada caso?

- a. `saluda(maría,Y)` con `despide(X,juan)`

No, porque `saluda` y `despide` son funtores diferentes, pertenecen a predicados diferentes. El resultado es “fallo”.

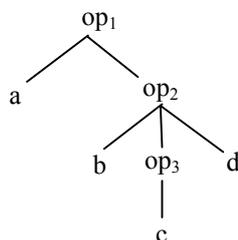
- b. `saluda(maría,Y)` con `saluda(juan,Y)`

No, porque aunque coincide el nombre de functor (`saluda`), el primer argumento está ligado a valores diferentes. El resultado es “fallo”.

- c. `saluda(maría,X)` con `saluda(maría,Y)`

Sí, porque coincide el nombre de functor (`saluda`), coincide el primer argumento (`maría`) y el segundo argumento corresponde a sendas variables libres. El resultado es “cierto” quedando ligadas ambas variables (aunque todavía sin valor) es decir,  $X=Y$ .

3. Sabiendo que el operador  $op_1$  es binario,  $op_2$  es ternario y que  $op_3$  es unario, dibuja el árbol de la siguiente expresión prefija:  $op_1 a op_2 b op_3 c d$ .



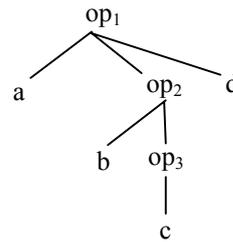
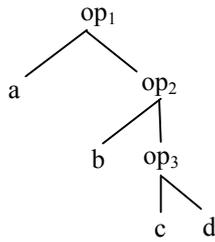


UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA  
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA  
EXAMEN DE LENGUAJES DE PROGRAMACIÓN

JUNIO 2004 - SEGUNDA SEMANA - SOLUCIÓN

¿Qué pasaría si  $op_2$  pudiera ser también binario? Dibuja el árbol correspondiente.

Obtendríamos el mismo árbol puesto que la expresión no encaja con la posibilidad de que  $op_2$  sea binario, a menos que  $op_3$  pudiera ser binario u  $op_1$  pudiera ser ternario, en cuyo caso podríamos tener los dos árboles siguientes:



¿Qué aridad tendría que tener  $op_1$  si  $op_2$  fuera unario? Dibuja el árbol correspondiente (1 punto)

Aridad 4.

