Universidad de Valladolid

Departamento de Informática

Programación III. I.T. Informática de Sistemas

Ejercicios de introducción a la herencia

Curso 2008–09

Ejercicio 1

Completar las definiciones de las clases polígono y rectángulo que se utilizaron en clase, incluyendo procedimientos de creación. (Puede ser útil consultar la implementación de Bertrand Meyer en el capítulo 14 de OOSC, pero es necesario tener en cuenta las diferencias entre la biblioteca estándar de SmartEiffel e ISE Eiffel).

Ejercicio 2

En un puerto se alquilan amarres para barcos de distinto tipo. Para cada *ALQUILER* se guarda el nombre y DNI del cliente, las fechas inicial y final de alquiler, la posición del amarre y el barco que lo ocupará. Un *BARCO* se caracteriza por su matrícula, su eslora en metros y año de fabricación.

Un alquiler se calcula multiplicando el número de días de ocupación (incluyendo los días inicial y final) por un módulo función de cada barco (obtenido simplemente multiplicando por 10 los metros de eslora) y por un valor fijo (2 € en la actualidad).

Sin embargo ahora se pretende diferenciar la información de algunos tipos de barcos:

- Número de mástiles para veleros
- Potencia en CV para embarcaciones deportivas a motor
- Potencia en CV y número de camarotes para yates de lujo

El módulo de los barcos de un tipo especial se obtiene como el módulo normal mas:

- El número de mástiles para veleros
- La potencia en CV para embarcaciones deportivas a motor
- La potencia en CV mas el número de camarotes para yates de lujo

Utilizando la herencia de forma apropiada, diseñe el diagrama de clases y sus relaciones, con detalle de atributos y métodos necesarios. Programe en Eiffel los métodos que permitan calcular el alquiler de cualquier tipo de barco.

Ejercicio 3

Dentro de una biblioteca Eiffel en funcionamiento disponemos de una clase empleado definida del siguiente modo:

```
class EMPLEADO
creation {ANY}
  make
feature {ANY}
  nombre: STRING;
```

UVa **Universidad** de **Valladolid.** Dpto. de Informática. Edificio de Tecnologías de la Información y las Telecomunicaciones, Campus Miguel Delibes. 47011 Valladolid. Tfno. 983.423000 ext.5617/Fax 983.423671.
www.infor.uva.es/~felix e-mail:felix@infor.uva.es

```
  edad: INTEGER;
  nif: STRING;
  make(elnombre: STRING; laedad: INTEGER; elnif: STRING) is
    do
      nombre := clone(elnombre);
      edad := laedad;
      nif := clone(elnif);
    end; -- make
  muestra is
    do
      io.put_string("Nombre: ");io.put_string(nombre);io.put_string("%N");
      io.put_string("edad: ");io.put_integer(edad);io.put_string("%N");
      io.put_string("NIF: ");io.put_string(nif);io.put_string("%N");
    end; -- muestra
end -- class EMPLEADO
```

Al añadir nuevas capacidades a la biblioteca descubrimos que necesitamos modelar nuevos tipos de empleados:

Empleado temporal del que nos interesa saber la fecha de alta y de baja en la empresa.

Empleado por horas Nos interesa el precio de la hora trabajada, y el número de horas que ha trabajado este mes. El primero es un dato fijo, mientras el segundo varía todos los meses.

Empleado fijo Debemos añadir a la información que almacenamos sobre él el año de alta en la empresa.

Además debemos añadir a todos los empleados la funcionalidad de cálculo del sueldo con las siguientes consideraciones:

- En los empleados temporales el sueldo mensual es fijo.
- En los empleados fijos el sueldo es el resultado de sumarle a la base un complemento anual fijo multiplicado por el número de años en la empresa.
- En los empleados por horas el sueldo se calcula multiplicando su sueldo por hora por el número de horas de este mes.

Diseñe (y escriba en Eiffel) las clases necesarias y sus relaciones para solucionar las nuevas necesidades detectadas.

Ejercicio 4

En una biblioteca universitaria multimedia se está construyendo un sistema de control de los fondos disponibles que incluyen documentos de distinto tipo. La forma de acceso es doble: Consultas en sala y préstamos temporales.

La consulta en sala requiere registrar el número del documento, la fecha y el DNI del alumno. El procedimiento de préstamo, en su caso, incluye los siguientes pasos: reservar el documento, recogerlo y devolverlo. Se manejan los mismos datos que en el caso de la consulta.

Los tipos de documentos que se contemplan son:

Los libros clásicos, en papel Los datos que interesa conocer son: Título, autor o autores, editorial, año de publicación. Estos libros se pueden prestar a los alumnos, salvo excepciones (diccionarios, normas ISO, etc., que sólo se pueden consultar en la sala).

Las revistas en papel que tienen las mismas características que los libros, más algunas peculiaridades: volumen, número y mes de salida. Se pueden consultar y prestar a los alumnos.

Documentos en formato CD (libros, software) Se pueden prestar, al igual que los libros. En este caso interesa mantener algún dato más (formato del CD, tipo de licencia).

Revistas de investigación microfilmadas que tienen las mismas características que las revistas en papel pero no se prestan y sólo se pueden consultar en la sala mediante terminales. Como dato adicional hay que mantener el código de microfilm.

Diseñe las clases y relaciones que representen una solución para este problema. Se pide en concreto, la estructura de herencia implicada con el detalle de características atribuidas a cada clase y sus posibles redefiniciones. Escriba en Eiffel la clase origen de la jerarquía de documento y la interfaz (la forma corta) de la clase que incluya los procedimientos de préstamo.

Ejercicio 5

Elaborar una jerarquía de herencia que modele los seres vivos capaces de hablar. Las clases deben modelar al menos a los loros, los profesores y los alumnos.

Todas las clases Eiffel elaboradas deben disponer de un método `habla` sin argumentos que proporcione una salida por pantalla similar a la siguiente:

```
Hola, me llamo Pedro y se hablar.
Soy racional.
Tengo 40 años.
Nací el 1 de enero de 1965
Soy profesor.
```

Para que el ejercicio sea interesante es necesario que todos los objetos habladores tengan un conjunto de características que les diferencian de los demás, por ejemplo, que los loros no sean conscientes de su edad o su fecha de nacimiento.

Ejercicio 6

Los loros del ejercicio anterior no pueden ser universitarios simultáneamente, pero un profesor puede ser también alumno. Elaborar un conjunto de clases que permitan modelar esta situación de forma que un objeto pueda cambiar su forma de hablar en tiempo de ejecución en función de la recepción de algún mensaje adecuado.

Ejercicio 7

Cualquiera de las jerarquías anteriores permite que una aplicación disponga de una lista de objetos capaces de hablar y les solicite que hablen de forma secuencial. Sin embargo, suponiendo que todos los universitarios dispongan de un NIU (Número de Identificación del Universitario), ¿Es posible que la aplicación se lo solicite sólo a los objetos que disponen de él?

Ejercicio 8

Un loro puede aprender a decir su fecha de nacimiento, pero no es práctico enseñarle a decir su edad cada año. Los profesores disponen ya de poca memoria y prefieren calcular su edad cada vez que son preguntados, los alumnos disponen de mucha memoria. ¿Cómo podemos implementar esta situación?

Ejercicio 9

En un sistema Eiffel que gestiona los usuarios de un servicio telemático se utiliza una clase *USUARIO* con la siguiente forma corta:

```
class interface USUARIO
creation
  nuevo (d: STRING; n: STRING)
    -- Inicializa el usuario con nombre 'n' y dni 'd'
    require
      d /= Void;
      n /= Void
feature(s) from USUARIO
  conexion (s: INTEGER)
    -- Contabiliza 's' segundos en la cuenta
    require
      s >= 0
  importe: DOUBLE
    -- Calcula el importe facturable
  reset
    -- Pone a cero la cuenta
end of USUARIO
```

El departamento de marketing ha diseñado un cojunto de ofertas no acumulables para los usuarios con las siguientes condiciones:

Oferta1 No se contabilizan los tres primeros minutos de cada conexión del cliente

Oferta2 Se aplica un tanto por ciento de descuento sobre el importe facturable facturable total. Este tanto por ciento se negocia por separado con cada cliente, pero una vez fijado no es modificable.

Oferta3 No se contabiliza la conexión más larga de cada periodo de facturación

Grandes clientes Se aplican simultáneamente las ofertas 2 y 3.

Elaborar las clases necesarias para resolver las nuevas necesidades del sistema considerando que no podemos modificar la clase *USUARIO*.
Nota: Ejercicio de examen del curso 2005/06

Ejercicio 10

En un sistema Eiffel que gestiona los usuarios de un servicio telemático se utiliza una clase *USUARIO* con la siguiente forma corta:

```
class interface USUARIO
creation
  nuevo (d: STRING; n: STRING)
    -- Inicializa el usuario con nombre 'n' y dni 'd'
    require
      d /= Void;
      n /= Void
feature(s) from USUARIO
  conexion (s: INTEGER)
    -- Contabiliza 's' segundos en la cuenta
    -- una conexión de cero segundos también supone
    -- un gasto en la cuenta.
```

```
    require
      s >= 0
  importe: DOUBLE
    -- Calcula el importe facturable
  reset
    -- Pone a cero la cuenta
end of USUARIO
```

El departamento de marketing ha diseñado un conjunto de ofertas no acumulables para los usuarios con las siguientes condiciones:

Vitamina 60x1 Los primeros sesenta minutos de cada conexión se facturan como un minuto.

Universal 25 30% de descuento en la factura, pero, independientemente del consumo, se facturará un importe mínimo de 25€

Bienvenida No se contabiliza la conexión más larga de cada periodo de facturación

Elaborar el diagrama de clases de la solución del problema e implementar en Eiffel las clases necesarias para resolver las nuevas necesidades del sistema considerando que no podemos modificar la clase *USUARIO*.
Nota: Ejercicio de examen del curso 2006/07

Ejercicio 11

En el sistema informático de un banco tenemos una clase *CUENTA* una parte de cuya forma corta es la siguiente:

```
class interface CUENTA
creation
  make
-- Secciones de la interfaz omitidas
feature(s) from CUENTA
  ingreso (c: DOUBLE)
    require
      c > 0
    ensure
      saldo = old saldo + c
  reintegro (c: DOUBLE)
    require
      c >= 0;
      c <= saldo
    ensure
      saldo = old saldo - c
  saldo: DOUBLE
  interes: DOUBLE
    ensure
      Result >= 0
  comision: DOUBLE
    ensure
      Result >= 0
end of CUENTA
```

Ahora se desean implementar nuevas versiones de la cuenta que cuenten con las siguietnes particularidades:

- Cuenta roja: Permite descubiertos, esto es, que el saldo quede en negativo, pero si al calcular el interés estamos en esa situación, el interés se vuelve cero. Del mismo modo la comisión se multiplica por dos cuando se calcula en situación de descubierto.
- Cuenta verde: Es una cuenta con comisión cero, pero en la que el interés calculado es cero si no se supera un valor mínimo de saldo, fijado para cada cuenta.
- Cuenta morada: Es una cuenta con interés cero y una comisión fija, determinada en el momento de creación de la cuenta.

Diseñar y construir las clases necesarias para resolver las nuevas necesidades del banco, considerando que no conocemos el código fuente de la clase cuenta y utilizando adecuadamente las técnicas relacionadas con la herencia.
Nota: Ejercicio de examen del curso 2007/08

Ejercicio 12

En el sistema informático de un banco tenemos una clase *CUENTA* una parte de cuya forma corta es la siguiente:

```
class interface CUENTA
creation
  make
-- Secciones de la interfaz omitidas
feature(s) from CUENTA
  ingreso (c: DOUBLE)
    require
      c > 0
    ensure
      saldo = old saldo + c
  reintegro (c: DOUBLE)
    require
      c >= 0;
      c <= saldo
    ensure
      saldo = old saldo - c
  saldo: DOUBLE
  interes: DOUBLE
    ensure
      Result >= 0
  comision: DOUBLE
    ensure
      Result >= 0
end of CUENTA
```

Ahora se desean implementar nuevas versiones de la cuenta con las siguientes particularidades:

- Cuenta roja: Dispone de una característica nueva *intereses_acumulados* que devuelve la suma de todos los intereses que se han calculado para esa cuenta
- Cuenta verde: Se trata de una cuenta roja con comisión cero si se supera en el momento del cálculo un saldo mínimo fijado para cada cuenta y renegociable.
- Cuenta morada: Dispone de una característica nueva *movimientos* con el número de ingresos y reintegros que se han realizado para esta cuenta.

Diseñar y construir las clases necesarias para resolver las nuevas necesidades del banco, considerando que no conocemos el código fuente de la clase cuenta y utilizando adecuadamente las técnicas relacionadas con la herencia.
Nota: Ejercicio de examen del curso 2007/08

Ejercicio 13

La facturación del agua en una pequeña ciudad se basa en tres conceptos: consumo, saneamiento y residuales. Para la gestión de los cobros del agua se ha elaborado una clase de cuya forma corta conocemos el siguiente extracto:

```
class interface CLIENTE
creation
   make
      -- Creación
-- Secciones de la interfaz omitidas
feature(s) from CLIENTE
   consumo: DOUBLE
      -- Importe por consumo
   saneamiento: DOUBLE
      -- Importe por uso del alcantarillado
   residuales: DOUBLE
      -- Importe por depuración de aguas
end of CONTADOR
```

Ahora necesitamos añadir al sistema clases que permitan facturar a diversos tipos de cliente:

- Los clientes normales, pagan la suma de los tres conceptos.
- Las familias numerosas de tipo 1, que pagan todo el consumo pero tienen un descuento del 25% en el saneamiento y la depuración.
- Las familias numerosas de tipo 2, que pagan todo el consumo pero tienen un descuento del 75% en el saneamiento y del 50% en la depuración, con un máximo en cada uno de estos dos conceptos de 16 €.

Considerando que los clientes pueden cambiar de tipo con cierta facilidad y que la previsible aparición de nuevos tipos de clientes desaconseja completamente la utilización de una sola clase para calcular el importe para los tres tipos de cliente, diseñar e implementar el conjunto de clases Eiffel que permitan calcular el importe de la factura del agua.
Nota: Ejercicio de examen del curso 2008/09

Ejercicio 14

GestAgua gestiona el cobro de los recibos del agua en una pequeña ciudad. Para el cálculo del importe final, la empresa utiliza una complicada fórmula que implica tramos de consumo entre otros parámetros. Esta complicada fórmula está implementada en una clase `CLIENTE`, de cuya forma corta conocemos el siguiente extracto:

```
class interface CLIENTE
creation
   make
      -- Creación
-- Secciones de la interfaz omitidas
feature(s) from CLIENTE
   consumo: DOUBLE
      -- Importe por consumo
   saneamiento: DOUBLE
      -- Importe por uso del alcantarillado
   residuales: DOUBLE
      -- Importe por depuración de aguas
   importe: DOUBLE
      -- Importe total de la última factura
      ensure
         importe=consumo+saneamiento+residuales
end of CONTADOR
```

Con la expansión de la empresa, se ha pasado a factuar agua a clientes de la provincia e incluso de capitales cercanas. Con este cambio se necesita facturar de modos diferentes a los nuevos clientes:

- Los «clientes del alfoz» no pagan saneamiento ni residuales, pero el consumo les sale un 10% más caro
- A los ayuntamientos del alfoz se les aplica un descuento del 25% sobre toda la factura.

Diseñar y construir las clases Eiffel que satisfagan las nuevas necesidades del sistema.
Nota: Ejercicio de examen del curso 2008/09