Examen de Introducción al Software
(Ingeniería Informática)

Febrero 2018

Primera parte (5 puntos, 50% nota del examen)

1) Se desea hacer un método para calcular y retornar el coste total de una reserva en un espectáculo, para un grupo. La tabla muestra las tarifas. Sobre estas tarifas se aplica un descuento del 10% para grupos de cinco personas o más.

Código de Tipo	Tipo Reserva	Precio por persona
ES	Solo espectáculo	25
EC	Espectáculo y consumición	32
EM	Espectáculo y menú	52
EE	Espectáculo y menú especial	62

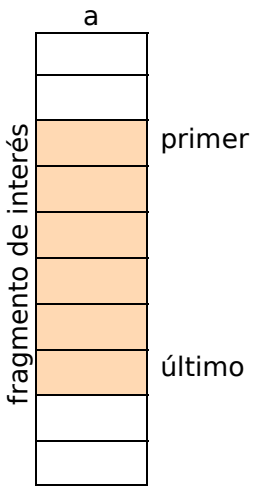
El parámetro `numPersonas` indica el número de personas del grupo. El parámetro `codigoTipo` es el código del tipo de reserva según los códigos que aparecen en la tabla. Si este parámetro no se corresponde con ningún código válido o si `numPersonas` es negativo o superior al aforo de 200 personas se retornará `Double.NaN` para indicar el error.

La cabecera del método será:

```
public static double costeReserva(int numPersonas, String codigoTipo)
```

Nota: Se valorará la eficiencia.

2) Se desea escribir un método *recursivo* que calcula el máximo de los elementos de un fragmento de un array de números reales, basándose en la división del trabajo. Se supone que el fragmento tiene al menos un elemento. El método recibe como parámetros el array y el primer y último índices del fragmento a considerar en el array. Ambos índices se consideran incluidos en el fragmento. El método retorna el valor del máximo. Su cabecera es:




```
public static double máximo
   (double[] a, int primer, int último)
```

En este algoritmo el caso directo se da cuando el tamaño del fragmento del array es 1 o 2. Si es uno, se retorna el valor de la única casilla contemplada. Si es dos, se retorna el máximo de las dos casillas contempladas.

En el caso recursivo se calcula el punto medio del fragmento como `medio=(primer+último)/2` y se consideran dos fragmentos del array: `[primer..medio]` y `[medio+1..último]`. En consecuencia se calcula recursivamente el máximo de cada fragmento, llamando al propio método `máximo()`. Y finalmente se retorna el valor máximo de los dos obtenidos.

Nota: recordar que el método `Math.max(x,y)` retorna el máximo de `x` e `y`.

3) Escribir un método iterativo que calcule y retorne el siguiente desarrollo en serie de la función coseno:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$$

El método debe hacer un bucle que se repite desde `0` hasta `n`, en el que se añade al resultado un término del desarrollo y luego se recalculan el signo, potencia y factorial. Para hacer más eficiente el cálculo no debe usarse la función "elevar a". Observaremos que cada potencia se obtiene multiplicando la anterior por x^2. El signo se obtiene cambiando de signo el anterior. El factorial del término `i+1` se obtiene multiplicando el del término `i` por `(2i+1)(2i+2)`.

El método recibe como parámetros los valores de `x` y `n`.

4) Contestar *razonadamente* a las siguientes preguntas. Utilizar un *máximo* de *5 líneas* para cada respuesta:

   a. En una hoja de cálculo tenemos en la celda `A1` un valor numérico que representa un factor de escala. En las columna `B` tenemos escrita una serie de valores numéricos. Queremos que en la columna `C` aparezcan los valores de la columna `B` multiplicados por el factor de escala de la celda `A1`. ¿Qué fórmula debemos escribir en la celda `C1` para ello? Queremos poder copiar esa fórmula en el resto de casillas de la columna.

   b. Disponemos de una hoja de cálculo con valores entre las filas 3 a 53 en la columna `C`. Indicar los pasos que darías para que en las mismas filas de la columna `D` aparezca el valor de la columna `C` de su misma fila o cero si este es negativo.

   c. Explica con brevedad qué ventajas tiene establecer una relación entre dos tablas de una base de datos relacional.

   d. Describe al menos cuatro propiedades que conozcas de los campos de un registro de una tabla en una base de datos relacional, indicando su utilidad.

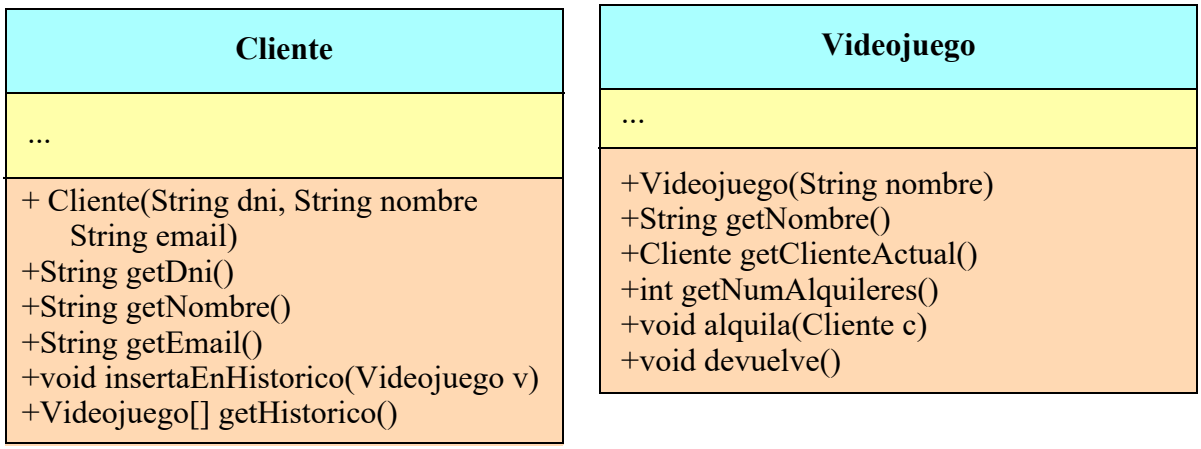
Nota: En esta cuestión, las respuestas que pasen de 5 líneas no se evaluarán. Se valora la *precisión* de la respuesta.

Examen de Introducción al Software
(Ingeniería Informática)

Febrero 2018

Segunda parte (5 puntos, 50% nota del examen)

Se desea escribir parte de una aplicación de una tienda de alquiler online de videojuegos. Se dispone ya de dos clases realizadas cuyos diagramas de clase se muestran aquí. La clase `Cliente` contiene los datos de un cliente de la tienda y `Videojuego` los de uno de los videojuegos:



Los métodos de la clase `Cliente` hacen lo siguiente:

- *Constructor*: Construye el cliente con los datos que se pasan como parámetros
- `getDni()`, `getNombre()`, `getEmail()`: son observadores de los respectivos atributos
- `insertaEnHistorico()`: Añade un videojuego al histórico de videojuegos alquilados por este cliente
- `getHistorico()`: Retorna el histórico de videojuegos alquilados por este cliente

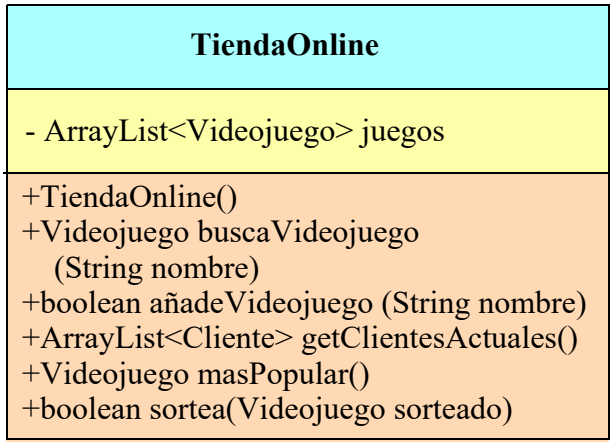
Y los de `Videojuego`:

- *Constructor*: Construye el videojuego con el nombre indicado en el parámetro. Lo deja inicialmente disponible y con cero alquileres.
- `getNombre()`: observador del atributo nombre
- `getClienteActual()`: Obtiene el cliente que ha alquilado un videojuego, o `null` si está disponible
- `getNumAlquileres()`: Retorna el número de alquileres que ha tenido este videojuego
- `alquila()`: Alquila el videojuego al cliente `c`. Anota el cliente en el videojuego. Además lo añade al histórico de videojuegos de ese cliente y aumenta el número de alquileres
- `devuelve()`: Devuelve el videojuego. Lo deja disponible.

Se pide escribir la clase `TiendaOnline`, excepto su método `getClientesActuales()` que se supone ya hecho. Los métodos aparecen en el diagrama de clase y deben hacer lo siguiente:

- *Constructor*: crea la lista de videojuegos vacía

- buscaVideojuego(): Busca en la tienda un videojuego por su nombre. Retorna el videojuego encontrado, o null si no se ha encontrado.
- añadeVideojuego(): Añade un videojuego nuevo a la tienda. Se comprueba que no exista ya un videojuego con ese mismo nombre. Si existe no se añade nada a a lista y se retorna false. Si no existe se crea el videojuego con el nombre indicado en el parámetro, se añade a la lista de videojuegos y se retorna true.



- getClientesActuales(): Obtiene la lista de clientes que tienen algún videojuego alquilado, evitando los clientes repetidos (se da ya hecho).
- masPopular(): Obtener el videojuego disponible más popular, que es el que mayor número de alquileres tiene. Retorna el videojuego actualmente disponible más popular o null si no hay ninguno disponible.
- sortea(): Sortea el videojuego indicado en el parametro sorteado entre los mejores clientes actuales, que son los que han alquilado el mayor número de videojuegos. El sorteo solo es posible si hay al menos un cliente actual. Retorna un booleano que indica si fue posible el sorteo. Para este método se seguirá el siguiente pseudocódigo:

```
método sortea (Videojuego sorteado) retorna booleano
   ArrayList<Cliente> actuales = getClientesActuales()
   si tamaño de actuales es 0 entonces
      // No hay ningún cliente actual
      retorna false
   si no
      // calculamos el máximo de alquileres de un cliente único
      entero max=-1
      para cada cli en actuales
         entero num= tamaño del histórico de alquileres de cli
         si num>max entonces
            max=num
         fin si
      fin para
      // obtenemos los clientes que tienen el máximo num de alquileres
      ArrayList<Cliente> maximos= nuevo ArrayList de clientes
      para cada cli en actuales
         si tamaño del histórico de alquileres de cli es max entonces
            añadir cli a maximos
         fin si
      fin para
      entero numMaximos=tamaño de maximos
      // Hacemos el sorteo
      entero ganador=número aleatorio entre cero y numMaximos menos uno
      sorteado.alquila(casilla ganador de maximos)
      retorna true
   fin si
fin método
```

Escribir además en una clase aparte un programa de prueba con un main que haga:

- Crea una tienda
- Añade tres videojuegos, mostrando en pantalla el resultado del método de añadir
- Añade un videojuego repetido, mostrando en pantalla el resultado del método de añadir
- Crea dos clientes
- Alquila el primer videojuego al primer cliente y el segundo al segundo cliente
- Obtiene el videojuego más popular
- Muestra en pantalla el nombre y numero de alquileres del videojuego más popular, o un mensaje de error si éste vale null
- Sortea el videojuego más popular, mostrando en pantalla si el sorteo fue posible

Puedes inventar los nombres y datos de videojuegos y clientes.

Valoración (sobre 10):

1) Encabezamiento de la clase, atributo y constructor: 1 punto
2) buscaVideojuego: 2 puntos
3) añadeVideojuego: 1 punto
4) masPopular: 2 puntos
5) sortea: 2 punto
6) main: 2 puntos