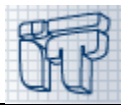


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 70 minutos
Puntuación máxima: 7 puntos
Fecha: 14 marzo 2019

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.
- Utiliza el espacio de los recuadros en blanco para responder a bolígrafo a cada uno de los apartados de los problemas (no usar lápiz para las respuestas finales).

NOTA: No está permitido crear más atributos ni métodos de los que figuran en el enunciado (no son necesarios).

Problema 1 (7 puntos)

Un hotel decide implementar un sistema de información para dar mejor servicio a sus Clientees.

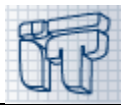
- El hotel dispone de tres tipos de Clientes (`Cliente`): Normal (`ClienteNormal`), de empresa (`ClienteEmpresa`) y Vip (`ClienteVip`).
- Todos los Clientes independientemente del tipo (`tipo`) asignado tienen además un nombre (`nombre`) y un identificador de Cliente (`idCliente`).
- El identificador de Cliente (`idCliente`) se asigna de forma consecutiva cada vez que se crea un nuevo Cliente en la aplicación, pero también es necesario llevar un contador con el número de Cliente es total (`numClientes`).
- Los diferentes tipos de Clientes pueden acceder a servicios de fidelización que consisten en el envío de regalos. El regalo recibido depende del tipo de Cliente.

Para colaborar en el modelado de los diferentes elementos del sistema deberás realizar las tareas indicadas en cada apartado.

Apartado 1: Interfaz Fidelizable (0,2 puntos)

Declara la interfaz `Fidelizable` y el método `enviarRegalo` que no tiene tipo de retorno y recibe como parámetro un objeto `regalo` de la clase `Regalo` que representa un regalo (No es necesario que programes la clase `Regalo`).

```
Apartado 1 (0,2 puntos)
```

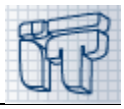


Apartado 2: Clase Cliente (1,5 puntos)

Declara la clase `Cliente` que implemente la interfaz declarada en el apartado anterior teniendo en cuenta las siguientes especificaciones:

- Declara tres atributos que no puedan ser accedidos ni modificados por ninguna otra clase: `Nombre` de tipo cadena, `tipo` de tipo carácter y `idCliente` de tipo entero.
- Declara un atributo `numClientes` que sea el mismo para todos los objetos de la clase.
- Crea tres constantes que puedan ser accedidas por cualquier clase que se llamen: `EMPRESA`, `VIP` y `NORMAL` que tomen los valores 'e', '`v`' y '`n`' respectivamente para representar los tres tipos de Clientes permitidos por la aplicación.
- Crea un método `getTipo` y un método `setTipo` que permitan devolver y asignar el valor a la variable `tipo`. El método `setTipo` deberá además controlar que el valor del tipo asignado es uno de los 3 valores permitidos. Si se introduce un valor incorrecto el método `setTipo` asignará por defecto el valor 'n' al tipo es decir el mismo valor que para un `ClienteNormal`.
- La clase `Cliente` no dispone de información suficiente para implementar el método `enviarRegalo` de la interfaz

```
Apartado 2 (1,5 puntos)
```



Apartado 3: Constructores y método toString clase Cliente (1 puntos)

Implementa dos constructores para la clase `Cliente`

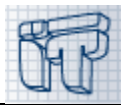
- El primer constructor deberá recibir como parámetros el nombre (`nombre`) y tipo de Cliente (`tipo`) y asignar valor a los atributos, `nombre`, `tipo`, `numClientes` y `idCliente` asegurándose sin repetir código que el valor asignado para el tipo es el valor correcto.
- Recuerda que el identificador de Cliente (`idCliente`) se asigna de forma consecutiva cada vez que se crea un nuevo Cliente en la aplicación pero también es necesario llevar un contador genérico del número de Clientes (`numClientes`) que sea el mismo para todos los objetos de la clase.
- El segundo constructor es un constructor por defecto que no recibe parámetros y asigna a cada atributo su valor por defecto. Este constructor deberá llamar al anterior para no repetir código.

Implementa el método `toString` de la clase Cliente que devuelva la información del Cliente en el siguiente formato:

```
Nombre: <nombre>
Tipo: <tipo>
ID:<idCliente>
```

Figura1: Formato de Impresión

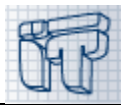
```
Apartado 3 (1 puntos)
```



Apartado 4: Clase ClienteEmpresa (atributos, constructores y métodos) (1 puntos)

Declara la clase `ClienteEmpresa` que hereda de la clase `Cliente`

- Declara un atributo `empresa` que no pueda ser accedido ni modificado por ninguna otra clase para almacenar el nombre de la empresa a la que pertenece el Cliente.
- Implementa un constructor de la clase `ClienteEmpresa` que reciba como parámetro el nombre del Cliente (`nombre`) y el nombre de la empresa a la que pertenece (`empresa`) y asigne valor al resto de atributos tal y como se indica en la especificación anterior.
- Implementa un constructor por defecto para la clase `ClienteEmpresa` que no reciba parámetros pero asigne adecuadamente el valor del tipo (`tipo`).
- Implementa el método `toString` de la clase `ClienteEmpresa` que imprima toda la información del Cliente en el formato que aparece en la figura 1 incluido el nombre de la empresa a la que pertenece el cliente.
- Implementa el método `enviarRegalo` de la clase `ClienteEmpresa` que imprima en pantalla el mensaje "`Enviando regalo :`" seguido de lo que devuelva el método `toString` de la clase `Regalo` (no es necesario que implementes la clase `Regalo`, simplemente asume que tiene un constructor por defecto y un método `toString()`).

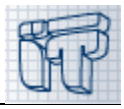


Apartado 4 (1 puntos)

Apartado 5: Clase Test y Método Main (1,8 puntos)

Programa una clase `Test` que contenga un método `main` que realice las siguientes operaciones:

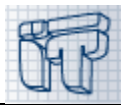
- Declara un array llamado `datos` de tres posiciones del tipo que consideres más apropiado.
- Crea un Cliente de cada tipo (`ClienteEmpresa`, `ClienteVip`, `ClienteNormal`) y almacénalos en el array. Para el Cliente de tipo `ClienteEmpresa` utiliza el constructor más completo para los otros dos tipos de Cliente puedes usar el constructor por defecto.
- Recorre el array creado
    - invocando al método `toString` de cada uno de sus elementos para imprimir la información correspondiente a todos los elementos almacenados en el array.
    - enviando el regalo correspondiente para cada cliente



Apartado 5 (1,8 puntos)

Apartado 6: Pruebas de programa: Cobertura (1,5 puntos)

- Observa la llamada al constructor de la clase `ClienteEmpresa` que has realizado e indica qué cobertura de ramas tiene el método `setTipo` que diseñaste en el apartado 1 al hacer dicha llamada.
- Cuantos objetos deberías crear para garantizar que has cubierto todas las ramas del método `setTipo` que diseñaste en el apartado 1. Escribe las sentencias correspondientes a la creación de dichos objetos (No es necesario que escribas los `assert`).



Apartado 6 (1,5 puntos)

Criterios de corrección

Apartado 1 Interfaz (0,2 puntos)

- (0,2) Declaración de interfaz y método abstracto.
  - (0 si no demuestra conocimiento de interfaz porque pone abstract en la declaración o implementa el método o pone {} en vez de ;)
  - No penalizar si pone public en el método aunque al ser una interfaz no es necesario porque todos sus métodos lo son.

Apartado 2 Clase Abstracta (1,5 puntos)

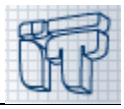
- (0,2) Declaración de la clase abstracta que implementa la interfaz
  - 0 si no pone abstract o no demuestra conocimiento de lo que es una clase abstracta. No es necesario que ponga el método de la interfaz. Pero no penalizar si lo pone de forma correcta.
  - Penalizar con -0,1 si pone abstract pero no pone implements
- (0.3) Declaración de los atributos privados nombre, tipo y idCliente.
  - 0 si hay algún fallo en los modificadores o tipos de datos.
- (0,2) Declaración del atributo static
  - 0 si hay algún fallo en modificadores o tipos de datos
- (0,2) Declaración de constantes
  - 0 si hay algún fallo en modificadores o tipos de datos
  - -0.1 si le falta alguno de los tres valores pero los demás son correctos.
- (0,2) Método get() (0 si tiene cualquier fallo)
- (0,4) Método set()
  - (-0.1) si controla todos los valores menos el valor por defecto.
  - No penalizar si lo hace de forma correcta aunque ineficiente

Apartado 3. Constructor y método toString de la clase abstracta (1 punto)

- (0,6) Constructor con parámetros
  - (0,1) declaración
  - (0,1) manejo y asignación del nombre
  - (0,2) manejo y asignación del tipo invocando a set (0 si lo hace directamente).
  - (0,1) manejo y asignación del atributo estático
  - (0,1) manejo y asignación del idCliente
- (0,2) Constructor sin parámetros
  - 0.1 declaración
  - 0.1 asignación
  - Penalizar con -0,1si hace asignación correcta pero repitiendo código (sin llamar al otro constructor)
  - 0 si no controla que el valor de categoría sea uno de los correctos.
- (0,2) Método toString. (0 si tiene algún fallo importante como no devolver un String o si la declaración no es correcta).
  - 0,1 declaración
  - 0,1 contenido
  - Penalizar con -0,1 si tiene algún fallo menor como olvidar los saltos de línea.

Apartado 4: Clase derivada (1 punto)

- (0,2) Declaración de la clase
- (0,2) Constructor con parámetros
  - 0 si declaración no es correcta o si no hace llamada a super porque no hay otro modo de asignar valor a los atributos al ser privados porque nombre no tiene método set.
- (0,2) Constructor sin parámetros
  - considerar válido llamar al constructor de la misma clase (si es correcto) o al constructor de la clase padre
  - 0 si declaración no es correcta o si hace asignación directamente porque los atributos son privados
- (0,2) Declaración e implementación del método enviarRegalo



- (0,2) Declaración e implementación del método toString

Apartado 5: Clase Test y Método main (1,8 puntos)

- (0,2) Declaración de la clase y método main
- (0,3) Declaración y creación de objetos ClienteEmpresa, ClienteVip, ClienteNormal (0 si hay algún fallo en la declaración o la llamada a los constructores)
- (0,3) Declaración y asignación de valores al array
  - (0 si no conoce manejo de arrays).
  - Sumar al apartado siguiente si lo hace directamente al crear el equipo.
- (0,25) Recorrer el array (0 si está mal el recorrido o los límites son incorrectos)
- (0,25) Imprimir información de cada elemento del array
  - 0 si invocan directamente a datos[i].toString() sin ponerlo dentro de System.out.println() o almacenando el valor en una variable que luego imprimen.
  - Considerar correcto tanto System.out.println(datos[i]) como System.out.println(datos[i].toString())
- (0,25) Crear correctamente el objeto Regalo. Pueden hacerlo directamente como parámetro del método enviarRegalo()
- (0,25) Invocar correctamente al método enviarRegalo

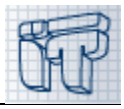
Apartado 6: Pruebas de programa (1,5 puntos). Puntuar sólo si la respuesta es correcta y consistente con el código creado por ellos mismos. La gráfica adjunta se corresponde a la cobertura del método propuesto como solución pero puede haber otras soluciones correctas con coberturas diferentes.

- (0,5) Cobertura de métodos.
- (0,5) Número correcto de llamadas.
- (0,5) Creación correcta de objetos

```
Apartado 1 (0,2 puntos)
public interface Fidelizable{
  void enviarRegalo(Regalo regalo);
}
```

```
Apartado 2 (1,5 puntos)
public abstract class Cliente implements Fidelizable{
  private String nombre;
  private char tipo;
  private int idCliente;
  private static int numClientes;
  public static final char VIP ='v';
  public static final char EMPRESA ='e';
  public static final char NORMAL ='n';

  public char getTipo(){
    return tipo;
  }
  public void setTipo(char tipo){
    if(tipo == VIP || tipo == EMPRESA){
        this.tipo = tipo;
      }else    {
        this.tipo = NORMAL;
      }
  }
}
```



Apartado 3 (1 puntos)

```
  public Cliente(String nombre, char tipo){
    this.nombre = nombre;
      setTipo(tipo);
      numClientes++;
      idCliente = numClientes;
  }
  public Cliente (){
    this(null, 'n');
  }
  public String toString(){
    // Se prodría poner en una sola línea return ....
    String result = "Nombre: "+ nombre + "\n" +
                                "Tipo: " + tipo + "\n" +
                                "ID: " + idCliente + "\n" ;
      return result;
  }
```

Apartado 4 (1 puntos)

```
public class ClienteEmpresa extends Cliente{
  private String empresa;
  public ClienteEmpresa(String nombre, String empresa){
    super(nombre, Cliente.EMPRESA);
      this.empresa = empresa;
  }
  public ClienteEmpresa(){
    this(null, null);
  }
  public String toString(){
    return super.toString() + "Empresa: " + empresa +"\n";
  }
  public void enviarRegalo(Regalo regalo){
     System.out.println("Enviando regalo empresa: " + regalo);
  }
}
```

Apartado 5 (1,8 puntos)

```
public class Test{
  public static void main(String[] args){
    Cliente[] datos = new Cliente[3];
      ClienteEmpresa e = new ClienteEmpresa("Pedro Perez", "Accenture");
      ClienteVip v = new ClienteVip();
      ClienteNormal n = new ClienteNormal();
      datos[0]= e;
      datos[1] = v;
      datos [2] = n;
      for(int i=0; i< datos.length;i++){
        System.out.println(datos[i]);
        datos[i].enviarRegalo(new Regalo());
      }
  }
}
```

Apartado 6 (1,5 puntos)

Cobertura de acuerdo al código de la solución propuesta para el método setTipo: