Examen de Prácticas de Programación
Ingeniería Informática

Junio 2008

1) Cuestiones

1.a) (0.75 puntos) Mostrar los contenidos del fichero `salida.txt` tras la ejecución del método main de la clase principal:

```
/**
 * Clase MiClase
 */
import java.util.*;

public class MiClase {
  private int entero;
  private String[] palabras = new String[2];

  public MiClase(int entero, String p1, String p2) {
    this.entero = entero;
    palabras[0]=p1;
    palabras[1]=p2;
  }

  public MiClase() {}

  public String toString() {
    return "{"+entero+","+Arrays.toString(palabras)+"}";
  }
}

/**
 * Clase principal
 */
import java.io.*;
import java.util.*;

public class Principal {
  public static void main(String[] args) throws IOException {
    MiClase[] a = new MiClase[2];
    a[0]=new MiClase();
    a[1]=new MiClase(4,"uno","dos");
    PrintWriter out=null;
    try {
      out = new PrintWriter(new FileWriter("salida.txt"));
      out.println(Arrays.toString(a));
    } finally {
      if (out!=null)
        out.close();
    }
  }
}
```

Solución:

```
[{0,[null, null]}, {4,[uno, dos]}]
```

1.b) (0.75 puntos) Escribir la salida por consola obtenida con la ejecución de la siguiente línea de código:

```
PropagaExcepción.método1();
```

Escribir también la salida que se obtendría si la excepción lanzada fuera `NumberFormatException` en lugar de `ArrayIndexOutOfBoundsException` y si no se lanzara ninguna excepción.

La clase `PropagaExcepción` es:

```
public class PropagaExcepción {

  private static void método3() {
    try {
      System.out.println("3.1");
      instrucción que provoca que se lance la
                          excepción ArrayIndexOutOfBoundsException
      System.out.println("3.2");
    }  finally {
      System.out.println("3.3");
    }
  }

  private static void método2() {
    try {
      System.out.println("2.1");
      método3();
      System.out.println("2.2");
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("2.3");
      throw e;
    } catch (Exception e) {
      System.out.println("2.4");
    }  finally {
      System.out.println("2.5");
    }
  }

  public static void método1() {
    try {
      System.out.println("1.1");
      método2();
      System.out.println("1.2");
    } catch (Exception e) {
      System.out.println("1.3");
    } finally {
      System.out.println("1.4");
    }
  }
}
```

Solución:

Lanza `ArrayIndexOutOfBoundsException`:

```
1.1
2.1
3.1
3.3
2.3
2.5
1.3
1.4
```

Lanza `NumberFormatException`:

```
1.1
2.1
3.1
3.3
2.4
2.5
1.2
1.4
```

No lanza excepción:

```
1.1
2.1
3.1
3.2
3.3
2.2
2.5
1.2
1.4
```

1.c) (0.75 puntos) ¿Cuál sería la salida por consola producida por este código? Indicar en qué instrucciones se produce la creación de objetos.

```
String str1="hola";
String str2=str1;
str1.toUpperCase();
str1=str1+"hola";
System.out.println("String1:"+str1+" String2:"+str2);
```

Solución:

Salida por consola:

```
String1:holahola String2:hola
```

Creación de objetos:

```
String str1="hola";    <- se crea el objeto "hola"
String str2=str1;
str1.toUpperCase();    <- se crea el objeto "HOLA"
str1=str1+"hola";      <- se crea el objeto "holahola"
System.out.println("String1:"+str1+" String2:"+str2); <- se crea
                      el objeto "String1:holahola String2:hola"
```

1.d) (0.75 puntos) Suponer que se ejecuta el método `main` de la clase `CreaciónYBasura`. Indicar claramente las instrucciones en que se produce la creación de objetos y también las instrucciones que causan que un objeto se convierta en "basura".

```
public class CreaciónYBasura {

  private static Integer método1(Integer[] a) {
    a= new Integer[2];
    a[0]= new Integer(1);
    a[1]= new Integer(2);
    return a[0];
  }

  public static void main(String[] args) {
    Integer[] a=null;
    Integer i = método1(a);
    do {
      Integer j = new Integer(3);
      i=j;
    } while (false);
    i = new Integer(4);
  }
}
```

Para que no haya confusión utilizar frases del tipo:

"En la instrucción ... se crea el objeto ..."

"Como consecuencia de la instrucción ... el objeto ... se convierte en basura"

"Tras la ejecución de la instrucción ... el objeto ... se convierte en basura"

Solución:

```
public class CreaciónYBasura {

  private static Integer método1(Integer[] a) {
    a= new Integer[2];  <- crea el array
    a[0]= new Integer(1); <- crea el Integer de valor 1
    a[1]= new Integer(2); <- crea el Integer de valor 2
    return a[0];
  }  <- el array y el Integer de valor 2 se convierten en basura

  public static void main(String[] args) {
    Integer[] a=null;
    Integer i = método1(a);
    do {
      Integer j = new Integer(3);  <- crea el Integer de valor 3
      i=j;   <- el Integer de valor 1 se convierte en basura
    } while (false);
    i = new Integer(4); <- crea el Integer de valor 4 y el
                           Integer de valor 3 se convierte en basura
  }
}
```

2) (2 puntos) Se desea implementar el método:

```
public static ArrayList<Punto> leeFicheroPuntos(String nomFich)
                        throws FileNotFoundException
```

que permite realizar la lectura de un fichero de texto que contiene un conjunto de puntos (uno por línea) y retorna esos puntos en un `ArrayList`.

El formato del fichero es el mostrado a continuación:

```
Punto    (3.45, -34.02 )
Punto ( -2.3 ,3.0)
Punto     (4.56,23.14)
...
```

Aclaraciones al formato:

- El número de espacios entre la palabra "Punto" y el paréntesis abierto es variable (uno o más)
- Puede haber espacios entre el paréntesis abierto y el primer número, antes o después de la coma y antes del paréntesis cerrado.

El fichero puede contener errores de formato. Cuando se encuentre una línea con un error de formato se introducirá en el `ArrayList` un punto en el que sus coordenadas `x` e `y` tengan el máximo valor que es posible almacenar en un `double`. Después de encontrar una línea errónea el método no debe finalizar, sino que deberá continuar procesando las restantes líneas del fichero.

La clase `Punto` es:

```
public class Punto {
  double x;
  double y;

  public Punto(double x, double y) {
    this.x=x;
    this.y=y;
  }
}
```

Solución:

```
public static ArrayList<Punto> leeFicheroPuntos(String nomFich)
                        throws FileNotFoundException,IOException {
  ArrayList<Punto> puntos = new ArrayList<Punto>();
  String str;

  BufferedReader ent = null;
  try {
    // abre el fichero
    ent = new BufferedReader(new FileReader(nomFich));

    // lee el fichero línea a línea
    do {
      // lee una línea
      str = ent.readLine();
```

```
        try {
          if (str!=null) {
            // procesa la línea. Como el tratamiento de todos
            // los errores es el mismo, usa el manejador de
            // NumberFormatException para todos

            // busca '(', ',' y ')'
            int posParenAbierto=str.indexOf('(');
            int posComa=str.indexOf(',');
            int posParenCerrado=str.indexOf(')');
            if (posParenAbierto==-1 || posComa==-1 ||
                posParenCerrado==-1) {
              // no encontrado '(' o ',' o ')'
              throw new NumberFormatException();
            }
            // comprueba que empieza por "Punto"
            if (!str.substring(0, posParenAbierto).trim().
                equals("Punto")) {
              // la línea no empieza con "Punto"
              throw new NumberFormatException();
            }
            // busca los números
            double x = Double.parseDouble(
                str.substring(posParenAbierto+1,posComa));
            double y = Double.parseDouble(
                str.substring(posComa+1, posParenCerrado));
            // añade el punto a la lista
            puntos.add(new Punto(x, y));
          }
        } catch (NumberFormatException e) {
          // tras cualquier error de formato de la línea se
          // ejecuta este manejador
          puntos.add(new Punto(Double.MAX_VALUE,
              Double.MAX_VALUE));
        }
      } while (str!=null);
    } finally {
      if (ent != null)
        ent.close();
    }
    return puntos;
  }
```

3) (5 puntos) Completar el diseño e implementación de la aplicación (únicamente de las partes solicitadas) que verifica los siguientes D-Requerimientos:

Se desea gestionar el acceso de vehículos a un aparcamiento de pago. El aparcamiento no se encuentra automatizado, por lo que existe un empleado encargado de registrar las entradas y salidas de vehículos.

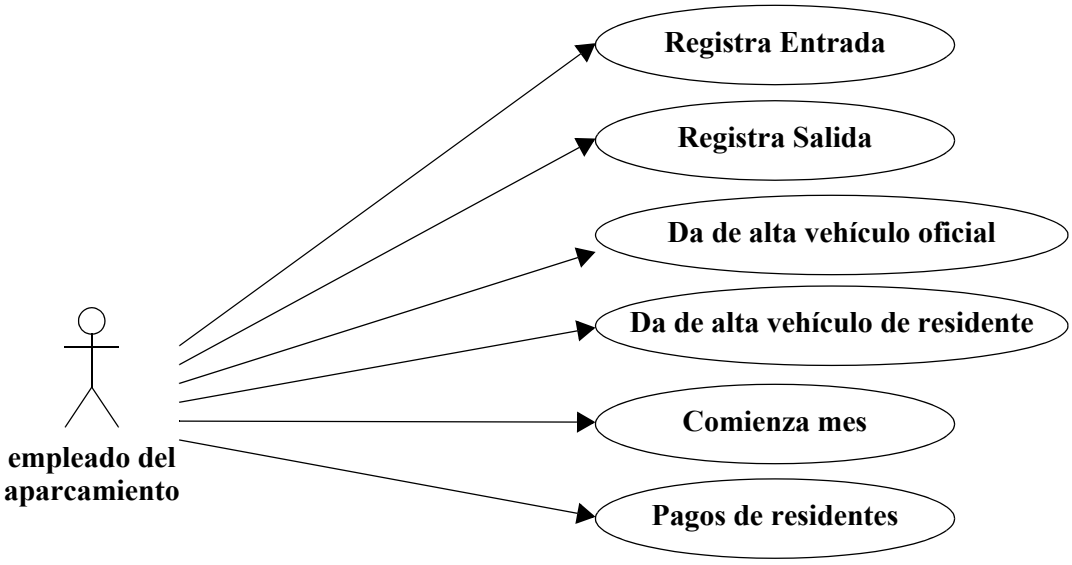
Los vehículos se identifican por su matrícula. Cuando un vehículo entra en el aparcamiento el empleado registra su entrada y al salir registra su salida y, en algunos casos, cobra el importe correspondiente por el tiempo de estacionamiento.

El importe cobrado depende del tipo de vehículo:

- Los vehículos oficiales no pagan, pero se registran sus estancias para llevar el control. (Una estancia consiste en una hora de entrada y una de salida)
- Los residentes pagan a final de mes a razón de 0.002€ el minuto. La aplicación irá acumulando el tiempo (en minutos) que han permanecido estacionados.
- Los no residentes pagan a la salida del aparcamiento a razón de 0.02€ el minuto.

Se prevé que en el futuro puedan incluirse nuevos tipos de vehículos, por lo que la aplicación desarrollada deberá ser fácilmente extensible en ese aspecto.

Casos de uso:



A continuación se procede a describir los casos de uso. No se entra en detalles de la interacción entre el empleado y la aplicación (punto 1 de cada caso de uso), puesto que no va a ser tarea del alumno desarrollar esa parte.

Caso de uso "Registra entrada":

1. El empleado elige la opción "registrar entrada" e introduce la matrícula del coche que entra.
2. La aplicación apunta la hora de entrada del vehículo.

Caso de uso "Registra salida":

1. El empleado elige la opción "registrar salida" e introduce la matrícula del coche que sale.
2. La aplicación realiza las acciones correspondientes al tipo de vehículo:
   - oficial: asocia la estancia (hora de entrada y hora de salida) con el vehículo

- residente: suma la duración de la estancia al tiempo total acumulado
- no residente: obtiene el importe a pagar

Caso de uso "Da de alta vehículo oficial":

1. El empleado elige la opción "dar de alta vehículo oficial" e introduce su matrícula.
2. La aplicación añade el vehículo a la lista de vehículos oficiales

Caso de uso "Da de alta vehículo de residente":

1. El empleado elige la opción "dar de alta vehículo de residente" e introduce su matrícula.
2. La aplicación añade el vehículo a la lista de vehículos de residentes.

Caso de uso "Comienza mes":

1. El empleado elige la opción "comienza mes".
2. La aplicación elimina las estancias registradas en los coches oficiales y pone a cero el tiempo estacionado por los vehículos de residentes.

Caso de uso "Pagos de residentes":

1. El empleado elige la opción "genera informe de pagos de residentes" e introduce el nombre del fichero en el que quiere generar el informe.
2. La aplicación genera un fichero que detalla el tiempo estacionado y el dinero a pagar por cada uno de los vehículos de residentes. El formato del fichero será el mostrado a continuación:

```
Matrícula     Tiempo estacionado (min.)    Cantidad a pagar
S1234A            20134                        40.27
4567ABC            4896                         9.79
...               .....                        .....
```

La aplicación contará con un programa principal basado en menú que permitirá al empleado interactuar con la aplicación (dicho programa principal se supone encargado a otro programador, por lo que *no deberá ser realizado por el alumno*).

El alumno deberá desarrollar las clases que permitan gestionar los vehículos con sus datos asociados (estancias, tiempo, ...), las listas de vehículos registrados como oficiales y residentes, etc.

Se pide:

- diseño arquitectónico de la parte de la aplicación encargada al alumno
- código de las clases correspondientes a la parte de la aplicación encargada al alumno

Para obtener la fecha y hora actual se utiliza la clase `Calendar`:

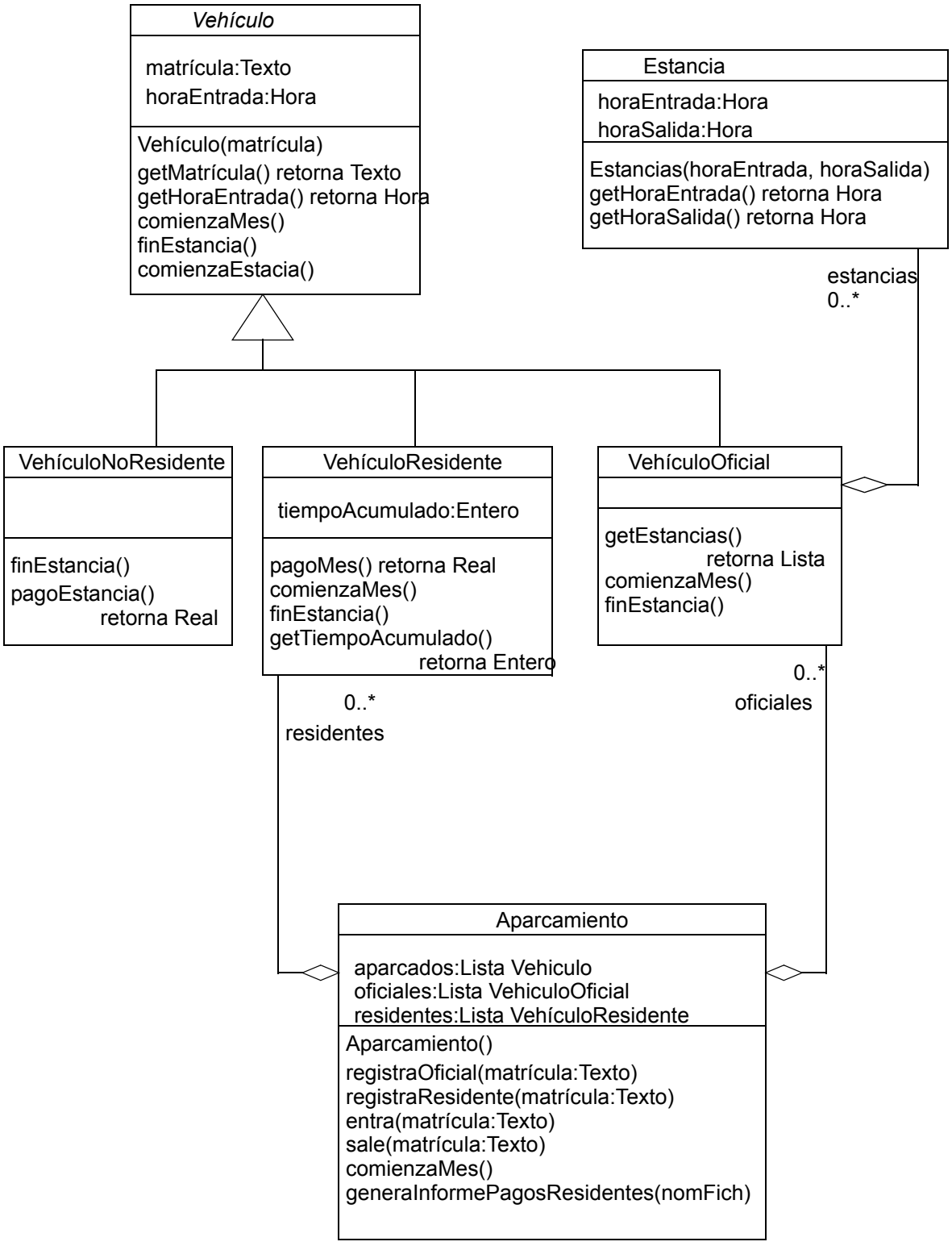
```
Calendar unaFecha; // para almacenar una fecha
...
unaFecha=Calendar.getInstance(); // obtiene la fecha actual
```

Para obtener intervalos de tiempos entre dos fechas suponer que se dispone del método:

```
/** Obtiene la diferencia en minutos entre dos fechas
 * @param inicial fecha inicial
 * @param final fecha final
 * @return diferencia final-inicial en minutos
 */
private static int difEnMinutos(Calendar inicial,
                                Calendar final) {...}
```

Solución:

Diseño arquitectónico:



```java
import java.util.*;

/**
 * Vehículo abstracto. Superclase de las clases de vehículos
 * reales
 */
public abstract class Vehiculo {

  private String matrícula;
  private Calendar horaEntrada;

  public Vehiculo(String matrícula) {
    this.matrícula = matrícula;
  }

  public String getMatrícula() {
    return matrícula;
  }

  public Calendar getHoraEntrada() {
    return horaEntrada;
  }

  public void comienzaMes() {
  }

  public void finEstancia() {
  }

  public void comienzaEstancia() {
    this.horaEntrada=Calendar.getInstance();
  }

}
```

```java
import java.util.*;
/**
 * Vehículo no residente. Debe pagar la estancia al salir.
 */
public class VehiculoNoResidente extends Vehiculo {

  private static final double precioMinuto = 0.02;

  private double pagoEstancia=0.0;

  public VehiculoNoResidente(String matrícula) {
    super(matrícula);
  }

  @Override
  public void finEstancia() {
    // calcula la cantidad a pagar
```

```
      pagoEstancia =
        difEnMinutos(getHoraEntrada(),Calendar.getInstance())
        * precioMinuto;
    }

    public double pagoEstancia() {
      return pagoEstancia;
    }

  }
```

```
  import java.util.Calendar;
  import java.util.LinkedList;
  /**
   * Vehículo oficial. Lleva la lista de las estancias
   * en el aparcamiento realizadas en el mes en curso
   */
  public class VehiculoOficial extends Vehiculo {

    // lista de las estancias en el mes en curso
    private LinkedList<Estancia> estancias =
                          new LinkedList<Estancia>();

    public VehiculoOficial(String matrícula) {
      super(matrícula);
    }

    public LinkedList<Estancia> getEstancias() {
      return estancias;
    }

    @Override
    public void comienzaMes() {
      estancias.clear(); // borra la lista de estancias
    }

    @Override
    public void finEstancia() {
      // añade la estancia a la lista
      estancias.add(
          new Estancia(getHoraEntrada(),Calendar.getInstance()));
    }

  }
```

```
  import java.util.Calendar;
  /**
   * Vehículo de residente. Lleva la cuenta del tiempo
   * de estancia acumulado en el mes en curso
```

```
 */
public class VehiculoResidente extends Vehiculo {

  private static final double precioMinuto = 0.002;

  // tiempo de estancia acumulado en el mes en curso
  private int tiempoAcumulado = 0;

  public VehiculoResidente(String matrícula) {
    super(matrícula);
  }

  public int getTiempoAcumulado() {
    return tiempoAcumulado;
  }

  @Override
  public void comienzaMes() {
    // pone a 0 el tiempo acumulado
    tiempoAcumulado=0;
  }

  @Override
  public void finEstancia() {
    // incrementa el tiempo acumulado en la duración de
    // la estancia que finaliza en este instante
    tiempoAcumulado +=
      difEnMinutos(getHoraEntrada(),Calendar.getInstance());
  }

  public double pagoMes() {
    return tiempoAcumulado*precioMinuto;
  }
}
```

```
import java.util.*;
import java.io.*;

/**
 *  Gestiona el aparcamiento
 */
public class Aparcamiento {

  public static class VehiculoYaRegistrado extends Exception {}
  public static class VehiculoYaAparcado extends Exception {}
  public static class VehiculoNoAparcado extends Exception {}

  // Registro de vehículos oficiales
  private LinkedList<VehiculoOficial> oficiales =
    new LinkedList<VehiculoOficial>();
```

```
// Registro de vehículos de residentes
private LinkedList<VehiculoResidente> residentes =
  new LinkedList<VehiculoResidente>();

// Vehículos que se encuentran en un momento dado en el
// aparcamiento. Pueden ser tanto vehículos de residentes,
// como oficiales como de no residentes
private LinkedList<Vehiculo> aparcados =
   new LinkedList<Vehiculo>();

/**
 * Busca el coche con la matrícula indicada en la
 * lista de vehículos oficiales
 * @param matrícula matrícula del vehículo buscado
 * @return el vehículo o null en caso de que no lo encuentre
 */
private VehiculoOficial buscaOficial(String matrícula) {
  for(VehiculoOficial v:oficiales) {
    if (v.getMatrícula().equals(matrícula))
      return v;
  }
  return null;
}

/**
 * Busca el coche con la matrícula indicada en la
 * lista de vehículos de residentes
 * @param matrícula matrícula del vehículo buscado
 * @return el vehículo o null en caso de que no lo encuentre
 */
private VehiculoResidente buscaResidente(String matrícula) {
  for(VehiculoResidente v:residentes) {
    if (v.getMatrícula().equals(matrícula))
      return v;
  }
  return null;
}

/**
 * Busca el coche con la matrícula indicada en la
 * lista de vehículos aparcados
 * @param matrícula matrícula del vehículo buscado
 * @return el vehículo o null en caso de que no lo encuentre
 */
private Vehiculo buscaAparcado(String matrícula) {
  for(Vehiculo v:aparcados) {
    if (v.getMatrícula().equals(matrícula))
      return v;
  }
  return null;
}

/**
 * Añade el coche con la matrícula indicada a la lista de
```

```
     * vehículos oficiales
     * @param matrícula matrícula del nuevo coche oficial
     * @throws VehiculoYaRegistrado ya existe un coche con
     * esa matrícula en la lista
     */
    public void registraOficial(String matrícula)
                        throws VehiculoYaRegistrado {
      VehiculoOficial v = buscaOficial(matrícula);
      if (v!=null)
        throw new VehiculoYaRegistrado();

      v = new VehiculoOficial(matrícula);
      oficiales.add(v);
    }

    /**
     * Añade el coche con la matrícula indicada a la lista de
     * vehículos de residentes
     * @param matrícula matrícula del nuevo coche de residente
     * @throws VehiculoYaRegistrado ya existe un coche con
     * esa matrícula en la lista
     */
    public void registraResidente(String matrícula)
                                  throws VehiculoYaRegistrado {
      VehiculoResidente v = buscaResidente(matrícula);
      if (v!=null)
        throw new VehiculoYaRegistrado();

      v = new VehiculoResidente(matrícula);
      residentes.add(v);
    }

    /**
     * Un vehículo entra al aparcamiento
     * @param matrícula matrícula del coche que entra
     * @throws VehiculoYaAparcado ya existe un coche con esa
     * matrícula dentro del aparcamiento
     */
    public void entra(String matrícula)
                                    throws VehiculoYaAparcado {
      Vehiculo v = buscaAparcado(matrícula);
      if (v!=null) {
        // error: ya existe un coche dentro del aparcamiento
        // con esa matrícula
        throw new VehiculoYaAparcado();
      }

      // vemos si es un vehículo de residente
      v = buscaResidente(matrícula);
      if (v==null) {
        // no es un vehículo de residente, vemos si es oficial
        v = buscaOficial(matrícula);
        if (v==null) {
          // tampoco es oficial, luego es de no residente.
```

```
            // Crea un vehículo de no residente
            v = new VehiculoNoResidente(matrícula);
          }
        }

        // sea del tipo que sea, llamamos al método correspondiente
        // a comenzar la estancia y le añadimos a la lista de
        // vehículos aparcados
        v.comienzaEstancia();
        aparcados.add(v);
    }

    /**
     * Un vehículo sale del aparcamiento
     * @param matrícula matrícula del vehículo que sale
     * @return el vehículo que sale para que, si es necesario,
     * puedan consultarse sus datos (pago, estancias, ..)
     * @throws VehiculoNoAparcado si la matrícula no corresponde
     * a ningún vehículo aparcado
     */
    public Vehiculo sale(String matrícula)
                                    throws VehiculoNoAparcado {
        Vehiculo v = buscaAparcado(matrícula);
        if (v==null) {
          // error: el vehículo debería estar en el aparcamiento!!
          throw new VehiculoNoAparcado();
        }

        // finaliza la estancia y se elimina de la lista de aparcados
        v.finEstancia();
        aparcados.remove(v);

        return v;
    }

    /**
     * Pone a 0 los registros de todos los vehículos
     */
    public void comienzaMes() {
        for(VehiculoResidente v: residentes)
          v.comienzaMes();
        for(VehiculoOficial v: oficiales)
          v.comienzaMes();
    }

    /**
     * Genera un informe con los pagos que deben realizar los
     * residentes
     * @param nomFich fichero en el que se escribe el informe
     * @throws IOException error de E/S
     */
    public void generaInformePagosResidentes(String nomFich)
                                            throws IOException {
        PrintWriter sal = null;
```

```
      try {
        sal = new PrintWriter(new FileWriter(nomFich));

        sal.println("Matrícula    Tiempo estacionado (min.)"
                    + "    Cantidad a pagar");
        for(VehiculoResidente v: residentes) {
          sal.printf("%-20s %7d %20.2f%n", v.getMatrícula(),
              v.getTiempoAcumulado(), v.pagoMes());
        }
      } finally {
        if (sal!=null)
          sal.close();
      }
    }
}
```

```
import java.util.*;
/**
 * Estancia de un vehículo oficial
 */
public class Estancia {

  private Calendar horaEntrada;
  private Calendar horaSalida;

  public Estancia(Calendar horaEntrada, Calendar horaSalida) {
    super();
    this.horaEntrada = horaEntrada;
    this.horaSalida = horaSalida;
  }

  public Calendar getHoraEntrada() {
    return horaEntrada;
  }

  public Calendar getHoraSalida() {
    return horaSalida;
  }
}
```