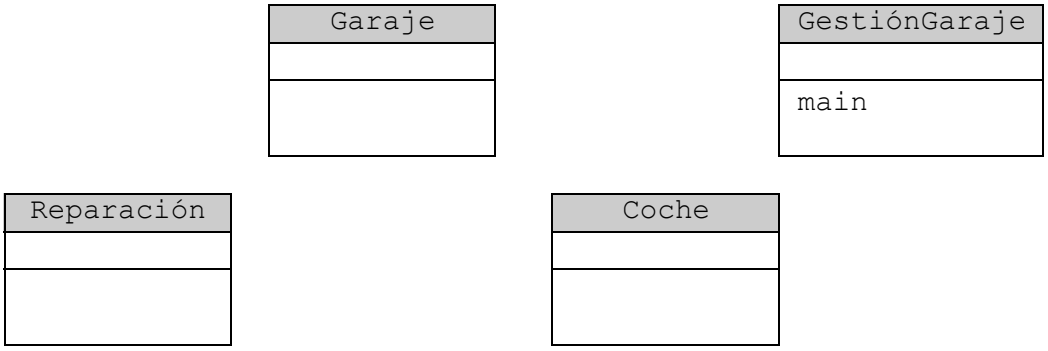
Examen de Prácticas de Programación
Ingeniería Informática

Septiembre 2007

1) (3.5 puntos) Se pretende desarrollar un programa para gestionar las reparaciones de coches realizadas en un garaje. Del análisis de requerimientos del problema se han obtenido los siguientes datos:
   - El garaje debe mantener una lista con todos los coches que ha reparado.
   - Cada coche se identifica por su matrícula. Además un coche debe tener la dirección de su dueño y un historial con todas las reparaciones a las que se ha visto sometido.
   - Las reparaciones deben contener la información relativa a la avería reparada y al kilometraje del coche en el momento de la reparación.

   Se han identificado los siguientes casos de uso:
   - Asociar una reparación a un coche. Debe identificarse como un error el hecho de que la reparación tenga un kilometraje inferior al de la última reparación realizada.
   - Obtener la última reparación realizada a un coche.
   - Buscar todas las reparaciones realizadas sobre un coche que contengan una palabra clave.
   - Incluir un coche en la lista de coches reparados por el garaje. Se considera un error tratar de incluir un coche con matrícula igual a la de alguno de los coches ya existentes en la lista.

   Diseño arquitectónico (incompleto):

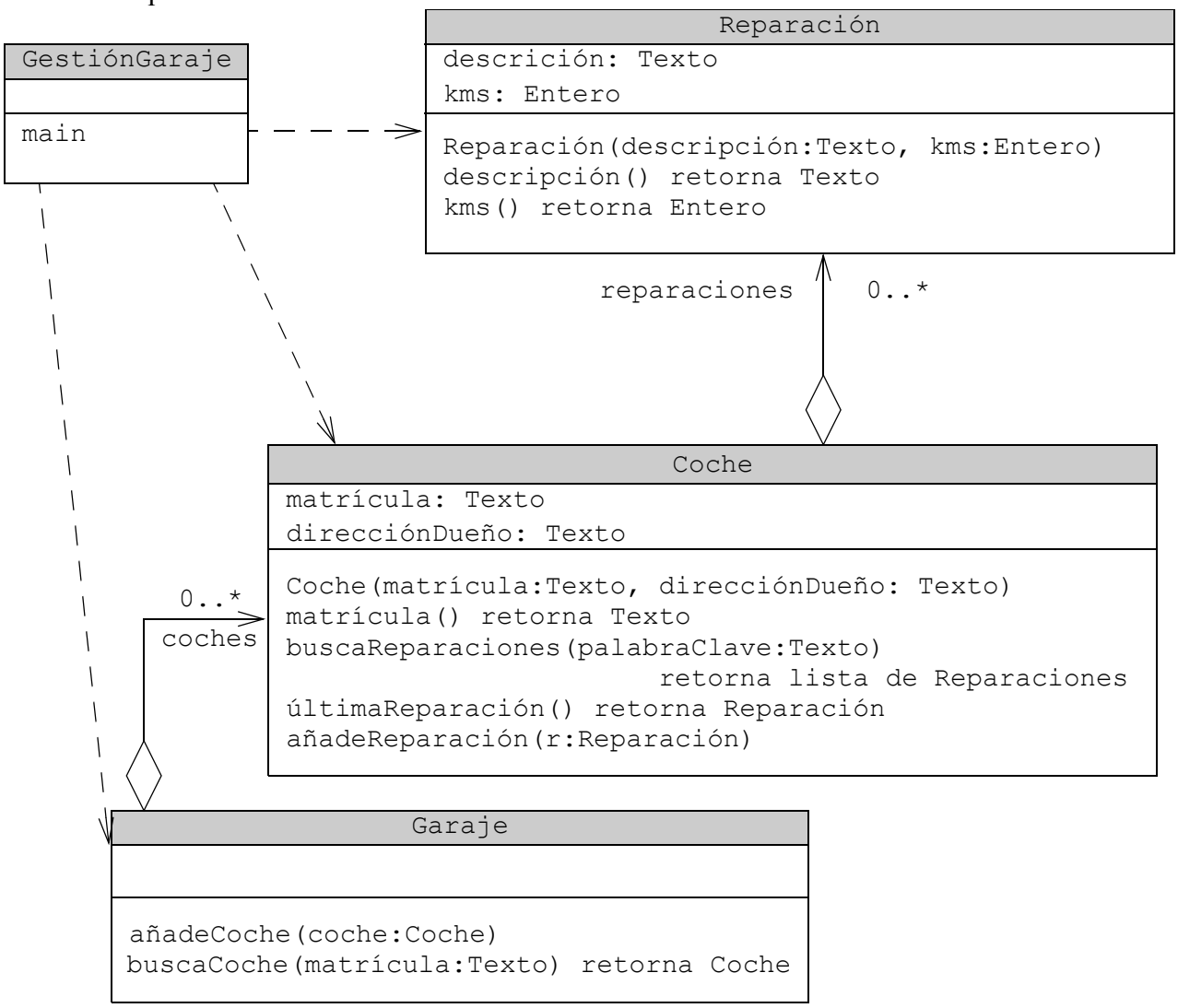


   Completar el diseño arquitectónico. Realizar el diseño detallado (únicamente de los métodos más complejos y en especial del método `main`). Realizar la codificación de las clases `Garaje`, `Coche` y `Reparación`.

   (Nota: enunciado ligeramente modificado para adaptarle a la nomeclatura utilizada en el curso 07/08)

Solución:

Diseño arquitectónico



Diseño detallado:

```
método main
  crea menú
  hacer
    intenta {
      lee opción menú
      si (opción) {
      ASOCIA_REPARACIÓN=>
        pide datos reparación (descripción y kms)
        pide matrícula coche
        coche = garaje.buscaCoche(matrícula)
        reparación = nueva Reparacion(descripción, kms)
        coche.añadeReparación(reparación)

      ÚLTIMA_REPARACIÓN=>
        pide la matrícula del coche
        coche = garaje.buscaCoche(matrícula);
        intenta {
          reparación = coche.últimaReparación();
          muestra datos reparación
```

```
            coge NoHayReparaciones
              mensaje "no hay reparaciones registradas"
            fintenta

          BUSCAR=>
            pide matrícula del coche
            pide palabra clave
            coche = garaje.buscaCoche(matrícula);
            listaRep = coche.buscaReparaciones(palabraClave);
            muestra todas las reparaciones en listaRep

          AÑADIR=>
            pide datos coche (matrícula y dirección del dueño)
            coche = nuevo Coche(matrícula, dirección);
            garaje.añadeCoche(coche);

          SALIR=>
            finaliza programa
          fsi

        coge MatriculaRepetida
          mensaje error "ya existe un coche con esa matrícula"
        coge CocheNoExiste
          mensaje error "no existe ningún coche con esa matrícula"
        coge KmsIncorrectos
          mensaje error "error en kilometraje"
        fintenta
      fhacer
    fmétodo
```

Codificación:

```
  public class Garaje {
    private LinkedList<Coche> coches = new LinkedList<Coche>();

    public void añadeCoche(Coche coche) throws MatriculaRepetida {
      // comprueba si ya existe algún coche con esa matrícula
      if (buscaCoche(coche.matrícula()) != null) {
        // ya existe un coche con esa matrícula
        throw new MatriculaRepetida();
       }
      // si llega hasta aquí es porque no hay ningún coche con
      // esa matrícula
      coches.add(coche);
    }

    public Coche buscaCoche(String matrícula) {
      for(Coche c: coches)
        if (c.matrícula().equals(c.matrícula()))
          return c;
      return null;
    }
  }
```

```
public class Coche {

  public static class KmsIncorrectos extends Exception {}

  private String matrícula;
  private String direcciónDueño;
  private LinkedList<Reparacion> reparaciones =
                                    new LinkedList<Reparacion>();
  // reparaciones también podría ser un array, aunque mejor una
  // LinkedList

  public Coche(String matrícula, String direcciónDueño) {
    this.matrícula=matrícula;
    this.direcciónDueño=direcciónDueño;
  }

  public String matrícula() {
    return matrícula;
  }

  public String direcciónDueño() {
    return direcciónDueño;
  }

  public LinkedList<Reparacion> buscaReparaciones(
                                     String palabraClave) {
    LinkedList<Reparacion> lista = new LinkedList<Reparacion>();
    for(Reparacion r: reparaciones)
      if (r.descripción().indexOf(palabraClave)!=-1)
        // incluye la palabra clave
        lista.add(r);
    return lista;
  }

  public Reparacion últimaReparación() {
    if (reparaciones.size()==0)
      return null;
    return reparaciones.getFirst();
  }

  public void añadeReparación(Reparacion r)
                                          throws KmsIncorrectos {
    if (r.kms()<reparaciones.getFirst().kms())
      throw new KmsIncorrectos();
    reparaciones.addFirst(r);
  }
}
```

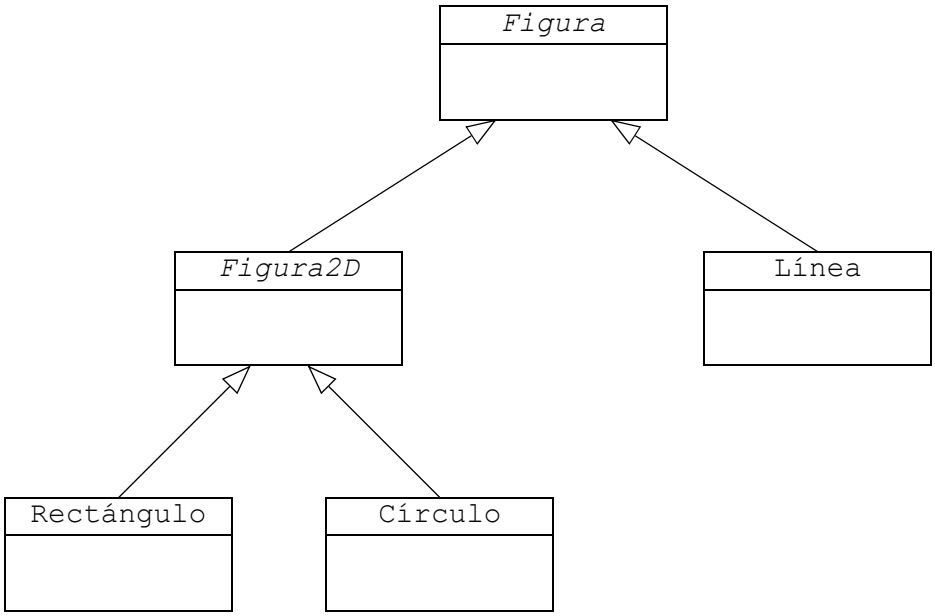
```
public class Reparacion {
  private String descripción;
  private int kms;

  public Reparacion(String descripción, int kms) {
    this.descripción=descripción;
    this.kms=kms;
  }

  public String descripción() {
    return descripción;
  }

  public int kms() {
    return kms;
  }
}
```

2) (2.5 puntos) Escribir el código (atributos y métodos) correspondiente a la jerarquía de clases mostrada a continuación:



Descripción de las clases:

- Una característica común a todas las figuras es el color con el que se dibujan sus bordes (representado mediante un número entero). El color de borde de una figura se especifica en el momento de su creación. También existen operaciones que permiten cambiar y obtener el color de borde de una figura ya creada.
- Todas las figuras tienen la operación de dibujar. Esta operación no está definida para las clases `Figura` y `Figura2D` y dibuja la figura correspondiente en el resto de las clases (no hay que escribir el código de esta operación, pero sí indicar el o los métodos en que debería incluirse dicho código)
- Las figuras de dos dimensiones tienen un color de relleno (representado mediante un número entero) que es el color con el que se rellena el interior de la figura. El color de relleno se especifica en el momento de su creación. También existen operaciones que permiten cambiar y obtener el color de relleno de una figura de dos dimensiones ya creada.
- Cada figura (`Rectángulo`, `Círculo` y `Línea`) debe tener los atributos propios de su geometría que serán especificados en el momento de su creación.

Además se desea disponer de un mecanismo que permita conocer el número total de objetos de las clase Figura y sus clases derivadas que han sido creados desde el comienzo de la ejecución de la aplicación.

(Nota: en el curso 06/07 no se vieron las clases abstractas. La pregunta ha sido modificada en 2008 para que las utilice)

Solución:

```
public abstract class Figura {
  private int colorBorde;
  private static int cuenta=0;

  public Figura(int colorBorde) {
    this.colorBorde=colorBorde;
    cuenta++;
  }
```

```
  public int colorBorde() {
    return colorBorde;
  }

  public void asignaColorBorde(int colorBorde) {
    this.colorBorde=colorBorde;
  }

  public abstract void dibuja();
  // abstracto, se define en las subclases

  public static int cuentaFiguras() {
    return cuenta;
  }
}



public abstract class Figura2D extends Figura {
  private int colorRelleno;

  public Figura2D(int colorBorde, int colorRelleno){
    super (colorBorde);
    this.colorRelleno=colorRelleno;
  }

  public int colorRelleno() {
    return colorRelleno;
  }

  public void asignaColorRelleno(int colorRelleno) {
    this.colorRelleno=colorRelleno;
  }
}


public class Linea extends Figura {
  private int xIni, yIni; // coordenadas del punto inicial
  private int xFin, yFin; // coordenadas del punto final

  public Linea(int xIni, int yIni, int xFin, int yFin,
               int colorBorde) {
    super(colorBorde);
    this.xIni=xIni;
    this.yIni=yIni;
    this.xFin=xFin;
    this.yFin=yFin;
  }

  public void dibuja() {
    // código correspondiente a dibujar una línea en base a sus
    // coordenadas xIni, yIni, xFin, yFin
  }
}
```

```
public class Circulo extends Figura2D{
  private int xCentro, yCentro; // coordenadas centro
  private int radio; // radio del círculo

  public Circulo(int xCentro, int yCentro, int radio,
       int colorBorde, int colorRelleno) {
    super(colorBorde, colorRelleno);
    this.xCentro=xCentro;
    this.yCentro=yCentro;
    this.radio=radio;
  }

  public void dibuja() {
    // código correspondiente a dibujar un círculo en base a su
    // centro (xCentro,yCentro) y a su radio
  }
}
```

```
public class Rectangulo extends Figura2D {
  private int x0, y0; // vértice superior izquierdo
  private int x1, y1; // vértice inferior derecho

  public Rectangulo(int x0, int y0, int x1, int y1,
       int colorBorde, int colorRelleno) {
    super(colorBorde, colorRelleno);
    this.x0=x0;
    this.y0=y0;
    this.x1=x1;
    this.y1=y1;
  }

  public void dibuja() {
    // código correspondiente a dibujar un rectángulo en base a
    // sus vértices (x0,y0) y (x1,y1)
  }

}
```

3) (2.5 puntos) Se está desarrollando un programa de gestión de los alumnos de una carrera universitaria. Se dispone de las clases `Alumno` y `FichaAsignatura`, cuya interfaz es la mostrada a continuación:

```
public class FichaAsignatura {
   /**
    * constructor
    */
   public FichaAsignatura(String nombre, double nota,
                  boolean matriculado, boolean superada) { xxx }

   /**
    * nombre de la asignatura
    */
   public String nombre() { xxx }

   /**
    * nota obtenida en la asignatura (sólo tiene sentido en
    * asignaturas ya superadas por el alumno)
    */
   public double nota() { xxx }

   /**
    * indica si un alumno se encuentra matriculado de
    * la asignatura
    */
   public boolean alumnoMatriculado() { xxx }

   /**
    * indica si la asignatura ha sido superada
    */
   public boolean superada() { xxx }
}

public class Alumno {
   /**
    * crea un alumno que no ha superado ni se encuentra
    * matriculado en ninguna asignatura
    */
   public Alumno(String nombre) { xxx }

   /**
    * retorna un array que indica la situación del alumno en TODAS
    * las asignaturas de la carrera (NO sólo aquellas en las que
    * el alumno se haya matriculado o ya haya superado)
    */
   public FichaAsignatura[] asignaturas() { xxx }

   /**
    * retorna el nombre del alumno
    */
   public String nombre() { xxx }
```

```
  /**
   * permite indicar que el alumno está matriculado en
   * una asignatura
   */
  public void matriculaEnAsignatura(String nombreAsig) { xxx }

  /**
   * permite indicar que el alumno ha superado una asignatura.
   * En el caso de que la nota no esté comprendida en el
   *intervalo [5.0,10.0] lanza la excepción NotaIncorrecta.
   */
  public void superaAsignatura(String nombreAsig, double nota)
        throws NotaIncorrecta { xxx }
}
```

Se está desarrollando la clase `ListaAlumnos`:

```
public class ListaAlumnos {
  /**
   * lista con todos los alumnos de la carrera
   */
  private LinkedList<Alumno> lista = new LinkedList<Alumno>();

  otros métodos no relevantes para el problema
}
```

Se pide añadir a la clase `ListaAlumnos` los dos métodos descritos a continuación:

- `guardaEnFichero`: guarda en un fichero de texto los alumnos existentes en `lista` (el nombre del fichero se pasa como parámetro). Junto con el nombre de cada alumno se indican las asignaturas en las que se encuentra matriculado o que ya ha superado (en el caso de estas últimas, junto al nombre de la asignatura aparece la nota obtenida). El formato del fichero será el mostrado a continuación:

```
Pedro García González
CIENCIA Y TECNOLOGIA DE MATERIALES:matriculado
QUIMICA:6.5
FUNDAMENTOS FISICOS:matriculado

Antonio López López
CIENCIA Y TECNOLOGIA DE MATERIALES:9.5
EXPRESION GRAFICA:matriculado
FUNDAMENTOS FISICOS:matriculado
FUNDAMENTOS MATEMATICOS:7.2

...
```

- constructor que recibe como parámetro el nombre de un fichero con el formato anteriormente mencionado. Lanzará la excepción `ErrorDeFormato` si detecta que falta el carácter ':' en la línea correspondiente a una asignatura o si el valor numérico de la nota no es correcto. También propagará el resto de excepciones que pueden producirse (`NotaIncorrecta` y las debidas a la utilización del fichero).

Solución:

```
public ListaAlumnos(String nomFich) throws ErrorFormatoFichero,
                                           FileNotFoundException,
                                           IOException,
                                           NotaIncorrecta {
  BufferedReader ent =null;
  try {
    // Abre el fichero
    ent = new BufferedReader(new FileReader(nomFich));
    // lee la primera línea
    String línea = ent.readLine();
    while(línea!=null && !línea.equals("")) { // nuevo alumno
      // crea un alumno con el nombre leído
      Alumno a = new Alumno(línea);
      lista.add(a);

      // lee la primera asignatura del alumno
      línea = ent.readLine();
      while(línea!=null && !línea.equals("")) {
        try {
          // separa el nombre de la asignatura de la nota
          int p = línea.indexOf(':');
          if (p==-1)
            throw new ErrorFormatoFichero();
          String nombreAsig = línea.substring(0,p);
          String nota=línea.substring(p+1);

          // mira si el alumno está evaluado en esa asignatura
          if (nota.equals("matriculado")) {
            // matricula al alumno en la asignatura
            a.matriculaEnAsignatura(nombreAsig);
          } else {
            // asignatura superada
            a.superaAsignatura(nombreAsig,
                               Double.parseDouble(nota));
          }
        } catch (NumberFormatException e) {
          throw new ErrorFormatoFichero();
        }

        // lee la siguiente asignatura del alumno o una línea
        // vacía o null
        línea = ent.readLine();
      }

      // lee la línea siguiente (nombre de alumno o null)
      línea = ent.readLine();
    }
  } finally {
    if (ent!= null) ent.close();
  }
}
```

```
public void guardaEnFichero(String nomFich)
                                    throws IOException{
  PrintWriter sal = null;
  try {
    sal = new PrintWriter(new FileWriter(nomFich));
    for(int i=0; i<lista.size(); i++) {
      Alumno a=lista.get(i);
      sal.println(a.nombre());
      FichaAsignatura[] asignaturas = a.asignaturas();
      for(FichaAsignatura asig: asignaturas) {
        if (asig.alumnoMatriculado() || asig.superada()) {
          sal.print(asig.nombre()+":");
          if (asig.superada())
            sal.println(asig.nota());
          else
            sal.println("matriculado");
        }
      }
      sal.println();
    }
  } finally {
    if (sal!=null)
      sal.close();
  }
}
```

4) (1.5 puntos) Escribir el código correspondiente a una cola genérica de longitud limitada:

```
public class Cola <T> {
   ...
}
```

La clase tendrá los siguientes métodos:

- Constructor: recibe como parámetro el número máximo de elementos que pueden almacenarse en la cola.
- `extraePrimero`: extrae y retorna el elemento que se encuentra en la cabeza de la cola. Lanza `ColaVacía` en el caso de que la cola esté vacía.
- `añadeÚltimo`: añade un elemento al final de la cola. Lanza `ColaLlena` en el caso de que se haya alcanzado el número máximo de elementos.

Utilizar una `LinkedList` para almacenar los elementos encolados. Para eliminar el primer elemento de la cola se debe utilizar el método `removeFirst` de la clase `LinkedList`. Este método lanza la excepción `NoSuchElementException` si la lista está vacía.

Suponer que las excepciones `ColaVacía` y `ColaLlena` se encuentran definidas en clases a parte.

Escribir método "main" que, utilizando la clase `Cola`, cree una cola de objetos de la clase `Integer` y realice la inserción y posterior extracción de dos objetos de dicha cola.

Solución:

```
public class Cola <T> {
  private LinkedList<T> cola = new LinkedList<T>();
  private int longMx;

  public Cola(int longMax) {
    this.longMx=longMax;
  }

  public T extraePrimero() throws ColaVacia {
    try {
      return cola.removeFirst();
    } catch (NoSuchElementException e) {
      throw new ColaVacia();
    }
  }

  public void añadeÚltimo(T elemento) throws ColaLlena {
    if (cola.size()==longMx)
      throw new ColaLlena();
    cola.addLast(elemento);
  }
}
```

```
public static void main(String[] args) {
  Cola<Integer> cola = new Cola<Integer>(3);
  try {
    cola.añadeÚltimo(1);
    cola.añadeÚltimo(5);
    Integer i = cola.extraePrimero();
    i=cola.extraePrimero();
  } catch (ColaVacia e){
    System.out.println("Cola vacía");
  } catch (ColaLlena e){
    System.out.println("Cola llena");
  }
}
```