Examen de Prácticas de Programación
Ingeniería Informática

Junio 2009

1) Cuestiones

1.a) (0.5 puntos) Se dispone de la clase `Punto2D` que compila correctamente:

```
public class Punto2D {
    public int x;
    public int y;

    public Punto2D(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Se pretende compilar la clase `Punto3D` que extiende a `Punto2D`:

```
public class Punto3D extends Punto2D {
    public int z;
}
```

En BlueJ la compilación de la clase `Punto3D` produce el error:
`"cannot find symbol - constructor Punto2D()"`
(`"No se encuentra el símbolo - constructor Punto2D()"`).
Explica la causa del error y propón la modificación de la clase `Punto3D` que lo corrige.

Solución:

Causa del error: cuando el programador no escribe un constructor para una clase, el lenguaje añade el constructor por defecto: constructor sin parámetros que invoca al constructor sin parámetros de la superclase. Eso significa que, en nuestro caso, es como si hubiéramos definido para `Punto3D` el siguiente constructor:

```
public Punto3D() {
  Punto2D();
}
```

Por lo tanto el error se produce debido a que no existe el constructor sin parámetros de la superclase `Punto2D`.

La solución más sencilla sería añadir un constructor a la clase `Punto3D`:

```
public Punto3D(int x, int y, int z) {
  super(x,y);
  this.z = z;
}
```

1.b) (0.5 puntos) Describe el error de compilación que se produciría en la línea marcada con la flecha:

```
try {
   ... // instrucciones no relevantes para la pregunta planteada
} catch (Exception e) {
   ... // instrucciones no relevantes para la pregunta planteada
} catch (ArithmeticException e) {  <----
   ... // instrucciones no relevantes para la pregunta planteada
}
```

Solución:

Las sentencias "catch" se evalúan en el orden que están escritas y un manejador de excepción coge excepciones de la clase indicada y de sus subclases. Eso supone que el manejador para `Exception` cogerá cualquier posible excepción que se pueda producir (lo que incluye a `ArithmeticException` que es una subclase de `Exception`). El compilador genera un error en la línea señalada para indicar que ese "catch" nunca se puede alcanzar puesto que hay otro anterior que le engloba.

1.c) (1 punto) En el código mostrado a continuación:

- Indica las cláusulas "`throws`" necesarias para que compile correctamente.
- Indica la salida por consola que se produciría si se ejecuta el programa con "`CASO=0`", "`CASO=1`" y "`CASO=2`".

```
public class LanzaExcepciones {
   private static final int CASO=...;   <-- (puede valer 0, 1 o 2)
   public static class MiExcepción extends Exception {}
   public static class MiRuntimeExcepción
                              extends RuntimeException{}

   private static void método3(int caso) {
      switch (caso) {
      case 0: throw new MiExcepción();
      case 1: throw new MiRuntimeExcepción();
      }
   }

   private static int método2() {
      int ret = 1;
      try {
         System.out.println("2:antes");
         método3(CASO);
         System.out.println("2:después");
         return 2;
      } catch (RuntimeException e) {
         System.out.println("2:catch");
```

```
        } finally {
          System.out.println("2:finally");
        }
        System.out.println("2:final");
        return ret;
      }

      private static int método1() {
        int ret = 0;
        try {
          System.out.println("1:antes");
          ret = método2();
          System.out.println("1:después");
        } catch (MiExcepción e) {
          System.out.println("1:catch");
        }
        System.out.println("1:final");
        return ret;
      }

      public static void main(String[] args) {
        System.out.println("main:antes");
        int ret = método1();
        System.out.println("main:después. ret:"+ret);
      }
    }
```

Solución:

Hay que poner las siguientes cláusulas "`throws`" para que el programa compile correctamente:

```
private static void método3(int caso) throws MiExcepción
private static int método2() throws MiExcepción
```

Salida por pantalla:

```
CASO=0

main:antes
1:antes
2:antes
2:finally
1:catch
1:final
main:después. ret:0
```

```
CASO=1

main:antes
1:antes
2:antes
2:catch
2:finally
2:final
1:después
1:final
main:después. ret:1
```

```
CASO=2

main:antes
1:antes
2:antes
2:después
2:finally
1:después
1:final
main:después. ret:2
```

1.d) (1 punto) Indica la salida por consola que se produciría si se ejecuta el programa mostrado a continuación. Indica también las líneas en las que se produce creación de objetos.

```
import java.util.LinkedList;

public class UsaLista {

  public static void main(String[] args) {

    LinkedList<MiClase> lista = new LinkedList<MiClase>();

    lista.add(new MiClase(1, "AA"));
    MiClase m1 = new MiClase(2, "BB");
    lista.add(m1);
    lista.add(new MiClase("CC"));
    lista.add(m1);
    System.out.println("1:" + lista);

    for(MiClase m2:lista) {
      m2.i = 3;
      m2.str = m2.str + 's';
    }
    System.out.println("2:" + lista);

    m1.str.toLowerCase();
    if (lista.contains(m1)) {
      System.out.println(m1 + " contenido en lista");
    } else {
      System.out.println(m1 + " NO contenido en lista");
    }

    m1.str = "otro string";
    if (lista.contains(m1)) {
      System.out.println(m1 + " contenido en lista");
    } else {
      System.out.println(m1 + " NO contenido en lista");
    }
  }
}
```

La clase `MiClase` es:

```
public class MiClase {
  Integer i;
  String str;

  public MiClase(String str) {
    this.str=str;
  }

  public MiClase(Integer i, String str) {
    this.i=i;
    this.str=str;
  }
```

```
    @Override
    public String toString() {
      return "(" + i + "," + str + ")";
    }

  }
```

Solución:

Salida por consola

```
1:[(1,AA), (2,BB), (null,CC), (2,BB)]
2:[(3,AAs), (3,BBss), (3,CCs), (3,BBss)]
(3,BBss) contenido en lista
(3,otro string) contenido en lista
```

Líneas en las que se produce creación de objetos:

- En todas las líneas en las que aparece un "`new`".
- En todas las líneas en las que se concatenan dos strings (operador "+" para strings) se crea un nuevo string igual a la concatenación de los 2 strings originales.
- En todas las líneas en las que se utiliza un `int` en lugar de un `Integer` (llamadas al constructor de la clase `MiClase` con dos parámetros y asignación "`m2.i = 3`") se crea un objeto de la clase `Integer` con el valor del `int`.
- En la línea "`m1.str.toLowerCase();`" se crea un string igual a `str` pero en minúsculas.
- En la línea "`m1.str = "otro string";`" se crea un nuevo string que contiene los caracteres "`otro string`".
- (No exigido para obtener la máxima puntuación en la pregunta) Dependiendo de la implementación de la `LinkedList` es muy probable que también se cree un objeto cada vez que se invoca el método `add`.

2) (2.75 puntos) Se desea implementar los métodos:

```
public static void grabaPacientes(
  String nomFich, LinkedList<Paciente> pacientes)
            throws IOException

private static LinkedList<Paciente> leePacientes(String nomFich)
            throws IOException, FormatoDeFicheroIncorrecto
```

Los cuales permiten realizar la escritura y lectura de ficheros de texto que almacenan una lista de pacientes. El formato de los ficheros se deja a elección del alumno. El método `leePacientes` deberá tener en cuenta que el fichero podría tener errores de formato. En el caso de encontrar alguno de esos errores, deberá notificarlo lanzando la excepción `FormatoDeFicheroIncorrecto` e indicando en su mensaje asociado la causa del error.

Además del código de los métodos solicitados, el alumno deberá entregar el código correspondiente a la definición de la excepción `FormatoDeFicheroIncorrecto` y la descripción del formato elegido para el fichero.

Se dispone de las clases `Visita` y `Paciente` ya implementadas:

```
/**
 * Cada una de las visitas realizadas por un paciente a su centro
 * de salud. Una visita se caracteriza por el año en que se
 * realizó y la causa que la provocó.
 */
public class Visita {

  public enum CausaVisita {receta, traumatismo,
                           enfermedad, consulta}

  public int año;
  public CausaVisita causa;

  public Visita(int año, CausaVisita causa) {
    this.año = año;
    this.causa = causa;
  }

}
```

La documentación de la parte pública de la clase `Paciente` es:

```
Clase Paciente
Clase que permite representar un paciente con su nombre y las
visitas que ha realizado al centro de salud

public Paciente(String nombre, LinkedList<Visita> visitas)
  Constructor de la clase paciente
  Parameters:
    nombre - nombre del paciente
    visitas - visitas realizadas por el paciente al centro de
              salud

public nombre()
  Retorna el nombre del paciente
  Returns:
      nombre del paciente

public Visita visita(int i)
  Retorna la visita i-ésima realizada por el paciente.
  Parameters:
      i - posición de la visita en la lista
  Returns:
      visita que ocupa la i-ésima posición o null si i no es una
      posición válida (i<0 o i>=número de visitas)
```

Solución:

Formato fichero:

Cada paciente se escribe en dos líneas, una para el nombre (y apellidos) y otra para las visitas. Cada visita se escribe como su año y su causa separados por espacios. A su vez una visita se separa de la siguiente también por espacios. Si un paciente no ha realizado ninguna visita, su línea de visitas será una línea en blanco. Ejemplo de fichero:

```
Nombre Apellido1 Apellido2
2000 enfermedad 2001 traumatismo 2002 consulta 2002 receta
Pepe Sin Visitas

Lolo Pérez Pérez
1990 traumatismo 2000 consulta 2009 receta
...
```

Excepción:

```
public static class FormatoDeFicheroIncorrecto extends Exception{

  public FormatoDeFicheroIncorrecto(String msj) {
    super(msj);
  }
}
```

Graba pacientes:

```
private static void GrabaPacientes(String nomFich,
    LinkedList<Paciente> pacientes) throws IOException {
  PrintWriter out = null;

  try {
    // abre el fichero
    out = new PrintWriter(new FileWriter(nomFich));

    // graba cada paciente en dos líneas con el formato
    // nombre apellidos
    // año_visita_0 causa_0 año_visita_1 causa_1 ...
    for(Paciente paciente: pacientes) {
      // escribe el nombre en una línea
      out.println(paciente.nombre());

      // escribe las visitas todas en la misma línea
      int i=0;
      Visita v = paciente.visita(i);
      while (v!=null) {
        out.print("   " + v.año + " " + v.causa);
        i++;
        v = paciente.visita(i);
      }
      out.println(); // para saltar a la línea siguiente
    }

  } finally {
    if (out != null)
```

```
            out.close();
        }
    }
```

Lee pacientes:

```
    private static LinkedList<Paciente> LeePacientes(
        String nomFich)
              throws IOException, FormatoDeFicheroIncorrecto {
      LinkedList<Paciente> pacientes = new LinkedList<Paciente>();
      LinkedList<Visita> visitas = new LinkedList<Visita>();
      Scanner in=null;

      try {
        // abre el fichero
        in = new Scanner(new FileReader(nomFich));

        // lee el fichero línea a línea (cada paciente ocupa dos
        // líneas, una para el nombre y otra para las visitas)
        while(in.hasNextLine()) {
          // lee la línea correspondiente al nombre
          String nombre = in.nextLine();

          // lee la línea correspondiente a las visitas
          Scanner líneaVis = new Scanner(in.nextLine());

          visitas.clear();

          // lee palabra a palabra los datos de las visitas
          while(líneaVis.hasNext()) {
            int año = líneaVis.nextInt();
            visitas.add(
              new Visita(año,CausaVisita.valueOf(líneaVis.next())));
          }

          // añade el paciente a la lista
          pacientes.add(new Paciente(nombre, visitas));
        }

      } catch (IllegalArgumentException e) {
        // lanzada por CausaVisita.valueOf() cuando el string no se
        // puede convertir a ningún valor del enumerado
        throw new FormatoDeFicheroIncorrecto(
            "Error en la causa de la visita");

      } catch (InputMismatchException e) {
        // lanzada por nextInt() cuando el siguiente trozo no se
        // puede convertir a entero
        throw new FormatoDeFicheroIncorrecto(
            "Valor de año incorrecto");

      } catch (NoSuchElementException e) {
        // lanzada por next() o nextInt() cuando no hay más trozos
        // que leer
        throw new FormatoDeFicheroIncorrecto(
```

```
                "Fin de fichero alcanzado antes de tiempo");

        } finally {
          if (in!=null)
            in.close();
        }

        return pacientes;
    }
```

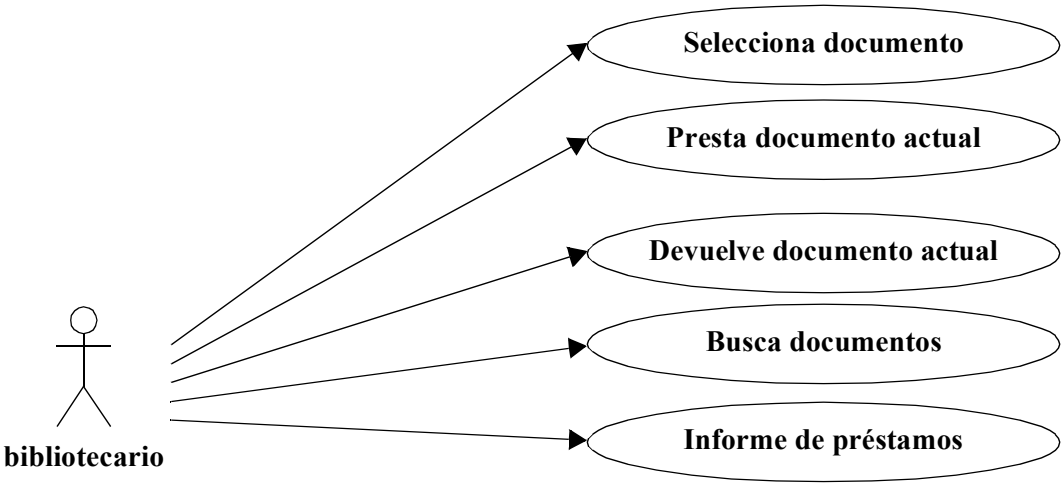
3) (4.25 puntos) Completa el diseño e implementación de la aplicación (únicamente de las partes solicitadas) que verifica los siguientes requisitos:

Se desea gestionar el préstamo de los documentos de una biblioteca. Existen dos clases de usuarios de la biblioteca: los socios y los usuarios ocasionales. Los socios pueden tener prestados simultáneamente 20 documentos como máximo, mientras que los clientes ocasionales sólo pueden tener 2. Los datos que componen la ficha de un usuario son su DNI y su nombre.

La biblioteca presta dos tipos de documentos: libros y revistas. La ficha de un documento se compone un código alfanumérico (que permite identificar el documento) y de su título. Además, la ficha de un libro tiene otro campo más: su año de publicación.

La duración máxima del préstamo de los libros a los socios es de 30 días, mientras que para los usuarios ocasionales la duración máxima del préstamo de los libros es de 15 días. La duración máxima del préstamo de una revista a un usuario es un tercio de la duración máxima del préstamo de un libro a ese mismo tipo de usuario.

Casos de uso:



(Una aplicación real incluiría también casos de uso para añadir y eliminar documentos, añadir y eliminar usuarios, etc. No se incluyen en el problema para limitar su complejidad)

A continuación se procede a describir los casos de uso. No se entra en detalles de la interacción entre el bibliotecario 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 "Selecciona documento":

1. El bibliotecario elige la opción "selecciona documento" e introduce el código del documento.
2. La aplicación pone como documento actual el documento con ese código.

   - En el caso de que no exista ningún documento con ese código se notifica

Caso de uso "Presta documento actual":

1. El bibliotecario elige la opción "presta documento actual" e introduce el DNI del usuario al que se va a prestar el documento actual.
2. La aplicación comprueba que es posible prestar el documento actual al usuario (existe un usuario con el DNI indicado, el usuario no ha alcanzado el límite de préstamos y el documento actual no se encuentra prestado)

   - si no es posible prestar el documento, lo notifica y finaliza el caso de uso
3. La aplicación registra el documento como prestado al usuario

Caso de uso "Devuelve documento actual":

1. El bibliotecario elige la opción "devuelve documento actual"
2. La aplicación finaliza el préstamo del documento actual

   - en el caso de que el documento actual no se encontrase prestado, se notifica y finaliza el caso de uso

Caso de uso "Busca documentos":

1. El bibliotecario elige la opción "busca documentos" e introduce un texto a buscar.
2. La aplicación busca todos los documentos que contengan ese texto en su título
3. La aplicación muestra los documentos encontrados

Caso de uso "Informe de préstamos":

1. El bibliotecario elige la opción "informe de préstamos".
2. La aplicación muestra por consola la información relativa a los documentos prestados: título, año de publicación (si es un libro), código, plazo de préstamo y usuario al que se le ha prestado. El formato debe ser exactamente el mostrado a continuación (respetando la ordenación por columnas).

```
El lenguaje de programación Java     (2000)  Cód:P101    Plazo:30días
  prestado a:12345678A (Socio)
El nombre de la rosa                 (1980)  Cód: A22    Plazo:15días
  prestado a:98765432C (Usuario Ocasional)
Science                                      Cód:  D1    Plazo: 3días
  prestado a:11223344G (Usuario Ocasional)
...
```

La aplicación contaría con un programa principal basado en menú. Para cada caso de uso, ese programa principal se encargaría de gestionar las ventanas que permiten al bibliotecario introducir los datos solicitados, llamaría al método apropiado de las clases desarrolladas por el alumno y mostraría los datos y/o errores correspondientes a las operaciones realizadas. (Dicho programa principal se supone encargado a otro programador, por lo que *no deberá ser realizado por el alumno*).

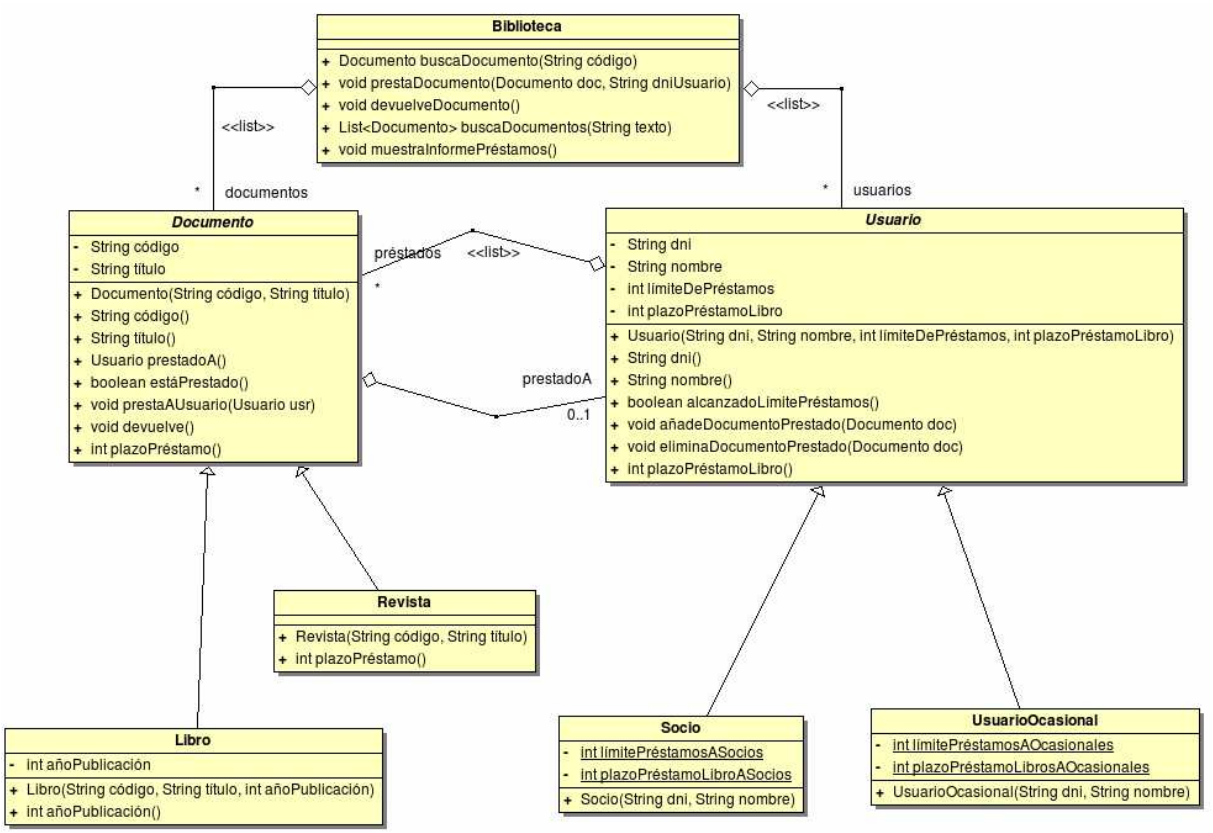
El alumno deberá desarrollar las clases que proporcionan las operaciones que permitirían al programa principal implementar los casos de uso descritos con anterioridad (dichas clases deben gestionar las listas de usuarios y libros, la situación de prestado o libre de los documentos, la información sobre los documentos prestados a un usuario, 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

(Utilizar la herencia en la implementación de las clases correspondientes a los documentos y los usuarios).

Solución:




```
/**
 * Usuario de la biblioteca. Clase abstracta que implementa el
 * comportamiento común de todos los tipos de usuarios de
 * la bibilioteca.
 */
public abstract class Usuario {

  public static class ErrorPréstamo extends RuntimeException {}
  public static class ErrorDevolución extends RuntimeException {}

  // datos del usuario
  private String dni;
  private String nombre;
```

```
// documentos prestados al usuario
private LinkedList<Documento> prestados=new LinkedList<Documento>();

// máximo número de documentos que puede tener prestados
private final int límiteDePréstamos;

// plazo de préstamo de un libro a este usuario
private final int plazoPréstamoLibro;

/**
 * Constructor
 * @param dni DNI del usuario
 * @param nombre nombre del usuario
 * @param límiteDePréstamos máximo número de documentos que
 * puede tener prestados simultáneamente
 * @param plazoPréstamoLibro plazo de préstamo de un libro al usuario
 */
public Usuario(String dni, String nombre, int límiteDePréstamos,
     int plazoPréstamoLibro) {
   this.dni = dni;
   this.nombre = nombre;
   this.límiteDePréstamos = límiteDePréstamos;
   this.plazoPréstamoLibro = plazoPréstamoLibro;
}

/**
 * Retorna el DNI del usuario
 * @return DNI del usuario
 */
public String dni() {
   return dni;
}

/**
 * Retorna el nombre del usuario
 * @return nombre del usuario
 */
public String nombre() {
   return nombre;
}

/**
 * Indica si se ha alcanzado el límite de préstamos
 * @return true si se ha alcanzado el límite de préstamos
 */
public boolean alcanzadoLímiteDePréstamos() {
   return límiteDePréstamos == prestados.size();
}

/**
 * Añade el documento a la lista de documentos prestados al usuario
 * @param doc documento a añadir
 * @throws ErrorPréstamo si no se puede prestar el documento porque
 * se ha alcanzado el límite de préstamos para este usuario o el
 * documento no se encuentra prestado a este usuario
 */
public void añadeDocumentoPrestado(Documento doc) {
```

```
      // detección de errores: se ha alcanzado el límite de préstamos
      // para este usuario o el documento no se encuentra prestado
      // a este usuario
      if (alcanzadoLímiteDePréstamos() || doc.prestadoA() != this)
        throw new ErrorPréstamo();

      prestados.add(doc);
    }

    /**
     * Elimina el documento de la lista de documentos prestados
     * @param doc documento a eliminar
     * @throws ErrorDevolución si el documento no está prestado
     */
    public void eliminaDocumentoPrestado(Documento doc) {
      // detección de error: el documento debe estar prestado a
      // este usuario
      if (doc.prestadoA() != this)
        throw new ErrorDevolución();

      if (!prestados.remove(doc))
        // el documento no estaba en la lista de prestados
        throw new ErrorDevolución();
    }

    /**
     * Retorna el plazo de préstamo de un libro al usuario
     * @return plazo de préstamo de un libro al usuario
     */
    public int plazoPréstamoLibro() {
      return plazoPréstamoLibro;
    }

    @Override
    public String toString() {
      return dni;
    }

}


/**
 * Socio de la biblioteca
 */
public class Socio extends Usuario {
  // límite de documentos que puede tener prestado un socio de la
  // biblioteca
  private static final int límiteDePréstamosASocios = 20;

  // plazo durante el que se puede prestar un libro a un socio de
  // la biblioteca
  private static final int plazoPréstamoLibroASocios=30;

  /**
   * Constructor
   * @param dni DNI del socio
   * @param nombre nombre del socio
   */
```

```
    public Socio(String dni, String nombre) {
      super(dni, nombre, límiteDePréstamosASocios,
                  plazoPréstamoLibroASocios);
    }

    @Override
    public String toString() {
      return super.toString() + " (Socio)";
    }
}

/**
 * Usuario ocasional de la biblioteca
 */
public class UsuarioOcasional extends Usuario {

    // límite de documentos que puede tener prestado un usuario
    // ocasional de la biblioteca
    private static final int límiteDePréstamosAOcasionales = 2;

    // plazo durante el que se puede prestar un libro a un usuario
    // ocasional de la biblioteca
    private static final int plazoPréstamoLibroAOcasionales=15;

    /**
     * Constructor
     * @param dni DNI del usuario
     * @param nombre nombre del usuario
     */
    public UsuarioOcasional(String dni, String nombre) {
      super(dni, nombre, límiteDePréstamosAOcasionales,
              plazoPréstamoLibroAOcasionales);
    }

    @Override
    public String toString() {
      return super.toString() + " (Usuario ocasional)";
    }
}


/**
 * Clase que engloba el comportamiento común a todos los
 * documento de la biblioteca
 */
public abstract class Documento {

    public static class YaPrestado extends RuntimeException {}
    public static class NoPrestado extends RuntimeException {}

    // datos del documento
    private String código;
    private String título;

    // usuario al que está prestado este documento. Vale null cuando
    // el documento no está prestado
    private Usuario prestadoA;
```

```java
/**
 * Constructor
 * @param código código identificativo del documento
 * @param título título del documento
 */
public Documento(String código, String título) {
   this.código = código;
   this.título = título;
}

/**
 * Retorna el código de este documento
 * @return código del documento
 */
public String código() {
   return código;
}

/**
 * Retorna el título de este documento
 * @return título del documento
 */
public String título() {
   return título;
}

/**
 * Retorna el usuario al que está prestado este documento
 * @return usuario al que está prestado este documento o null
 * en caso de que este documento no esté prestado
 */
public Usuario prestadoA() {
   return prestadoA;
}

public boolean estáPrestado() {
   return prestadoA != null;
}

/**
 * Presta este documento a un usuario. Marca el documento como
 * prestado y le añade a la lista de documentos prestados
 * al usuario
 * @param usr usuario al que se presta el documento
 * @throws YaPrestado si ya está prestado
 */
public void prestaAUsuario(Usuario usr) {
   // comprueba que no está prestado
   if (prestadoA != null)
      throw new YaPrestado();

   // apunta el usuario que lo tiene prestado
   prestadoA = usr;
   // añade el documento a la lista de documentos prestados al usuario
   prestadoA.añadeDocumentoPrestado(this);
}

/**
```

```
     * Devuelve este documento (deja de estar prestado). Le marca
     * como no prestado y le elimina de la lista de documentos prestados
     * al usuario
     * @throws NoPrestado si no está prestado
     */
    public void devuelve() {
      // comprueba que está prestado
      if (prestadoA == null)
        throw new NoPrestado();

      // elimina el documento de la lista de documentos prestados
      // al usuario
      prestadoA.eliminaDocumentoPrestado(this);
      // apunta que ya no está prestado
      prestadoA = null;
    }

    /**
     * Retorna el plazo durante el que está prestado este documento
     * @return plazo durante el que está prestado este documento
     * @throws NoPrestado si el documento no está prestado
     */
    public int plazoPréstamo() {
      // comprueba que está prestado
      if (prestadoA() == null)
        throw new NoPrestado();

      // el plazo depende del usuario que lo tenga prestado
      return prestadoA().plazoPréstamoLibro();
    }
}


/**
 * Libro de la biblioteca
 */
public class Libro extends Documento {

    // año de publicación del libro
    private int añoPublicación;

    /**
     * Constructor
     * @param código código identificativo del libro
     * @param título título del libro
     * @param añoPublicación año de publicación del libro
     */
    public Libro(String código, String título, int añoPublicación) {
      super(código, título);
      this.añoPublicación = añoPublicación;
    }

    /**
     * Retorna el año de publicación de este libro
     * @return año de publicación de este libro
     */
    public int añoPublicación() {
      return añoPublicación;
```

```
    }

}



/**
 * Revista de la biblioteca
 */
public class Revista extends Documento {

   /**
    * Constructor
    * @param código código identificativo de la revista
    * @param título título de la revista
    */
   public Revista(String código, String título) {
      super(código, título);
   }

   /* (non-Javadoc)
    * @see p3_biblioteca.Documento#plazoPréstamo()
    */
   @Override
   public int plazoPréstamo() throws NoPrestado {
      // el plazo de préstamo de una revista es un
      // tercio del de un libro
      return super.plazoPréstamo() / 3;
   }
}



/**
 * Biblioteca
 */
public class Biblioteca {

   public static class ErrorPréstamo extends Exception {
      public ErrorPréstamo(String msj) {
         super(msj);
      }
   }
   public static class ErrorDevolución extends Exception {
      public ErrorDevolución(String msj) {
         super(msj);
      }
   }

   // usuarios de la biblioteca
   private LinkedList<Usuario> usuarios = new LinkedList<Usuario>();

   // documentos de la biblioteca
   private LinkedList<Documento> documentos =
                                    new LinkedList<Documento>();

   /**
    * Busca un documento por su código
    * @param código código del documento a buscar
```

```
     * @return el documento encontrado o null si no existe ningún
     * documento en la biblioteca con ese código
     */
    public Documento buscaDocumento(String código) {
      for(Documento doc: documentos)
        if (doc.código().equals(código))
          return doc; // encontrado

      return null; // no encontrado
    }

    /**
     * Presta el documento a usuario con el DNI indicado
     * @param doc documento a prestar
     * @param dniUsuario DNI del usuario al que prestar el documento
     * @throws ErrorPréstamo si no se puede realizar el préstamo porque
     * el usuario no existe o ya ha alcanzado su límite de préstamos o
     * el documento ya se encuentra prestado
     */
    public void prestaDocumento(Documento doc,
        String dniUsuario) throws ErrorPréstamo {
      // busca el usuario
      Usuario usr = buscaUsuario(dniUsuario);
      if (usr==null)
        throw new ErrorPréstamo("Usuario inexistente. DNI:"+dniUsuario);

      // comprueba que se puede prestar
      if (usr.alcanzadoLímiteDePréstamos())
        throw new ErrorPréstamo("Alcanzado Límite de Préstamos");
      if (doc.estáPrestado())
        throw new ErrorPréstamo("Documento ya prestado. Cod:"+
            doc.código());

      // presta el documento al usuario
      doc.prestaAUsuario(usr);
    }

    /**
     * Devuelve el documento
     * @param doc documento a devolver
     * @throws ErrorDevolución si no se puede devolver el documento
     * porque no se encuentra prestado
     */
    public void devuelveDocumento(Documento doc) throws ErrorDevolución {
      // comprueba que el documento no está prestado
      if (!doc.estáPrestado())
        throw new ErrorDevolución("Documento no prestado");

      // devuelve el documento y le elimina de la lista
      // de préstamos del usuario
      doc.devuelve();
    }

    /**
     * Busca un usuario en base a su DNI
     * @param dni DNI del usuario buscado
     * @return el usuario encontrado o null si no existe ningún
     * usuario en la biblioteca con ese DNI
```

```
     */
    private Usuario buscaUsuario(String dni) {
      for(Usuario usr: usuarios)
        if (usr.dni().equals(dni))
          return usr; // encontrado

      return null; // no encontrado
    }

    /**
     * Retorna una lista con todos los documentos que contienen el texto
     * buscado en su título
     * @param parteTítulo texto buscado
     * @return lista de los documentos que contienen el texto
     * buscado en su título
     */
    public LinkedList<Documento> buscaDocumentos(String parteTítulo) {
      LinkedList<Documento> encontrados = new LinkedList<Documento>();

      // recorre los documentos de la biblioteca
      for(Documento doc: documentos) {
        // si el documento contiene parteNombre en su título, le añade
        // a la lista de encontrados
        if (doc.título().indexOf(parteTítulo) != -1)
          encontrados.add(doc);
      }

      return encontrados;
    }

    /**
     * Muestra por la consola el informe con información sobre los
     * documentos que se encuentran prestados
     */
    public void muestraInformePréstamos() {
      System.out.println("-- Informe de préstamos:");
      for(Documento doc: documentos) {
        if (doc.estáPrestado()) {
          if (doc instanceof Libro) {
            System.out.printf("%-30s", doc.título());
            System.out.printf("(%4s)  ",
                              ((Libro)doc).añoPublicación());
          } else {
            System.out.printf("%-38s", doc.título());
          }
          System.out.printf("Cód:%4s   Plazo:%2ddías\n",
               doc.código(), doc.plazoPréstamo());
          System.out.println("  prestado a:"+doc.prestadoA());
        }
      }
      System.out.println("--     fin informe de préstamos.");
    }

}
```