Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN - Septiembre 2010

Normas del examen:

- Con libros y apuntes.
- Duración: 2 horas y 30 minutos.
- Responda a cada problema en hojas separadas.
- No se contestará ninguna pregunta durante el examen.

Fechas:

- notas provisionales: 17.9.2010
- revisión: 22.9.2010
- notas finales: 30.9.2010

Problema 1 (3 puntos)

Se quiere implantar un algoritmo para calcular el valor de un número n elevado a una cierta potencia e

n^e

done

- n es un número real, positivo o negativo
- e es un número entero, positivo o negativo

Para realizar el cálculo sólo se pueden utilizar operaciones de multiplicación o división, sabiendo que una división por cero debe lanzar una excepción aritmética.

Se pide codificar en Java dos soluciones para este problema, una solución recursiva y otra iterativa. Se ofrecen las siguientes cabeceras de los métodos pedidos:

```
/**
 * @param n número real, positivo o negativo.
 * @param e exponente entero, positivo o negativo.
 * @return n elevado a e.
 */
public double potenciaRecursiva (double n, int e) {
      // realiza el calculo de forma recursiva
}


/**
 * @param n número real, positivo o negativo.
 * @param e exponente entero, positivo o negativo.
 * @return n elevado a e.
 */
public double potenciaIterativa (double n, int e) {
      // realiza el calculo de forma iterativa
}
```

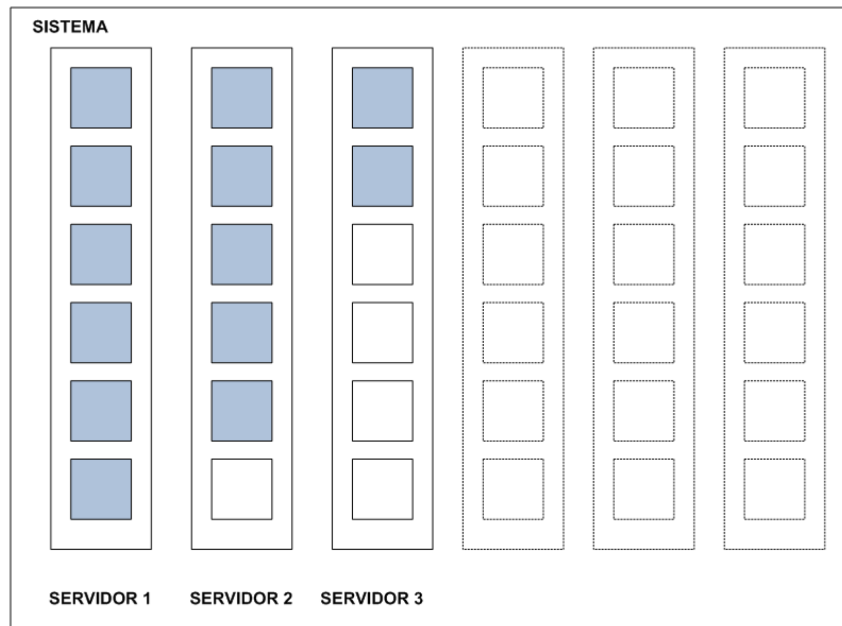
```java
    /**
     * @param n número real, positivo o negativo.
     * @param e exponente entero, positivo o negativo.
     * @return n elevado a e.
     * @throws ArithmeticException si hay alguna division por cero
     */
    public double potenciaRecursiva(double n, int e) {
        if (n == 0 && e < 0)
            throw new ArithmeticException("division por 0");
        if (e > 0)
            return n * potenciaRecursiva(n, e - 1);
        else if (e == 0)
            return 1.0;
        else if (e < 0)
            return 1.0 / potenciaRecursiva(n, -e);
        else
            return n;
    }


    /**
     * @param n número real, positivo o negativo.
     * @param e exponente entero, positivo o negativo.
     * @return n elevado a e.
     * @throws ArithmeticException si hay alguna division por cero
     */
    public double potenciaIterativa(double n, int e) {
        if (n == 0 && e < 0)
            throw new ArithmeticException("division por 0");
        double resultado = 1.0;
        for (int j = 0; j < Math.abs(e); j++) {
            resultado *= n;
        }
        if (e < 0)
            return 1.0 / resultado;
        return resultado;
    }
```

Problema 2 (4 puntos)

Se dispone de un sistema informático capaz de cargar y procesar tareas genéricas; estas tareas son procesadas en servidores internos del sistema. Estos servidores disponen de colas de tamaño limitado, y es en estas colas donde se van almacenando las tareas que entran en el sistema.

El sistema va creando servidores a medida que los va necesitando, hasta llegar a un límite predeterminado (puede verse esto como las cajas de un supermercado, donde a medida que va concentrándose público para pagar, se van abriendo nuevas cajas).



Cuando una tarea llega al sistema, esta se añade a la cola de proceso de uno de los servidores de acuerdo con las siguientes reglas:

- Si alguno de los servidores tiene capacidad para admitir tareas (si hay sitio en su cola), se añade en ese servidor.
- Si se ha alcanzado el límite de servidores posibles, la tarea se pierde.
- Si no, el sistema crea un nuevo servidor y carga la tarea en la cola de ese servidor.

En este ejercicio no nos preocuparemos de cómo o quién extrae las tareas de las colas de los servidores ni de si se desactivan o no los servidores cuando estos acaban de procesar todas sus tareas.

La documentación (javadoc) de las dos clases necesarias para modelar este escenario se muestran al final del enunciado.

Se pide:

1. Definir los atributos necesarios y el constructor para la clase *Servidor*, teniendo en cuenta que la cola debe implementarse utilizando un array.
2. Escribir el método *putTarea()* de la clase *Servidor*
3. Escribir el método *getTarea()* de la clase *Servidor*
4. Escribir el método *cargaTarea()* de la clase *Sistema*
5. Escribir el método *getCargaMedia()* del *Sistema.*

Documentación

La clase Servidor implementa la *interface InterfaceServidor*:

```
public interface InterfaceServidor {
    /**
     * Carga (encola) una tarea en un servidor.
     * Si la cola ya ha alcanzado su tamaño máximo, no se hace nada.
     * @param tarea El objeto tarea que se encola.
     * @return TRUE si la inserción se realizó con éxito; FALSE si no.
     */
    boolean putTarea(Tarea tarea);

    /**
     * Extrae y devuelve la primera tarea que se encuentra en la cola.
     * En caso de que no haya tareas encoladas, devuelve null.
     * @return El objeto tarea primero de la cola.
     */
    Tarea getTarea();

    /**
     * Devuelve el número de tareas encoladas que tiene un servidor.
     * @return El número de tareas.
     */
    int getNumTareas();

    /**
     * Indica si la cola del servidor está vacía.
     * @return TRUE si la cola está vacía.
     */
    boolean isVacia();

    /**
     * Indica si pueden encolarse más tareas en la cola de un servidor.
     * @return TRUE si la cola admite más tareas; FALSE en caso contrario.
     */
    boolean hayCapacidad();
}
```

La clase Sistema implementa la interface InterfaceSistema:

```
public interface InterfaceSistema {
    /**
     * Carga una tarea en el sistema.
     * Ver enunciado.
     * @param tarea Tarea que se carga en el sistema.
     * @return TRUE si la carga ha sido correcta; FALSE en caso contrario.
     */
    boolean cargaTarea(Tarea tarea);

    /**
     * Devuelve la carga media (número medio de elementos encolados)
     * en los servidores del sistema.
     * @return Media de la caraga del todos los servidores del sistema.
     */
    double getCargaMedia();
}
```

solución 1

```
public class Servidor
        implements InterfaceServidor {
    private int tamMax = 3;

    private int numTareas;
    private Tarea[] tareas;

    /**
     * Constructor.
     * Crea un servidor con una cola vacía.
     */
    public Servidor() {
        tareas = new Tarea[tamMax];
        numTareas = 0;
    }
```

solución 2

```
    /**
     * Carga (encola) una tarea en un servidor.
     * Si la cola del servidor ya ha alcanzado su tamaño máximo,
     * no se hace nada.
     *
     * @param tarea El objeto tarea que se encola.
     * @return TRUE si la inserción se realizó con éxito;
               FALSE en caso contrario.
     */
    public boolean putTarea(Tarea tarea) {
        if (numTareas < tamMax) {
            tareas[numTareas++] = tarea;
            return true;
        } else
            return false;
    }
```

solución 3

```
    /**
     * Extrae y devuelve la primera tarea que se encuentra
     * en la cola de un servidor.
     * En caso de que no haya tareas encoladas, devuelve null.
     *
     * @return El objeto tarea primero de la cola.
     */
    public Tarea getTarea() {
        Tarea tarea = null;
        if (numTareas > 0) {
            tarea = tareas[0];
            for (int i = 0; i < numTareas - 1; i++)
                tareas[i] = tareas[i + 1];
            tareas[numTareas - 1] = null;
            numTareas--;
        }
        return tarea;
    }
```

solución 4

```
/**
 * Carga una tarea en el sistema.
 * Si alguno de los servidores ya creados
 * tiene espacio en su cola de tareas,
 * la tarea se añade a ese servidor.
 * En caso contrario, el sistema crea un nuevo servidor
 * (si no se ha alcanzado el número máximo),
 * y añade la tarea a ese servidor
 *
 * @param tarea Tarea que se carga en el sistema
 * @return TRUE si la carga ha sido correcta; FALSE en caso contrario
 */
public boolean cargaTarea(Tarea tarea) {
    for (Servidor servidor : servidores) {
        if (servidor.hayCapacidad()) {
            servidor.putTarea(tarea);
            return true;
        }
    }
    // si no, activamos un nuevo procesador
    if (servidores.size() < NUM_MAX_SERVIDORES) {
        Servidor servidor = new Servidor();
        servidores.add(servidor);
        if (servidor.putTarea(tarea)) {
            return true;
        }
    }
    return false;
}
```

solución 5

```
/**
 * Devuelve la carga media (número medio de elementos encolados)
 * en los servidores del sistema.
 *
 * @return Media de la caraga del todos los servidores del sistema.
 */
public double getCargaMedia() {
    double suma = 0.0;
    int num = 0;
    for (Servidor servidor : servidores) {
        suma += servidor.getNumTareas();
        num++;
    }
    return suma / num;
}
```

Problema 3 (3 puntos)

Tenemos una clase *Persona* que tiene un nombre y un número de dni (número entero).

```
public class Persona {
    private String nombre;
    private int dni;
    public Persona(String nombre, int dni) {
        this.nombre = nombre;
        this.dni = dni;
    }
    public String getNombre() { return nombre; }
    public int getDni() { return dni; }
 }
```

La clase *Vehículo* que tiene una matrícula (un número entero) y una potencia (medida en caballos).

```
public Vehiculo (int matricula, int potencia) { … }
```

A partir de la clase *Vehículo* se definen las siguientes clases derivadas:

- *Coche* que tiene un número de plazas
  `public Coche(int matricula, int potencia, int numeroPlazas) { … }`
- *Camión* que tiene una cabina y un remolque
  `public Camion(int matricula, Cabina cabina, Remolque remolque) { … }`
  - Donde la *Cabina* es una clase que tiene una determinada potencia
    `public Cabina(int potencia) { … }`
  - y el *Remolque* es una clase que tiene una carga máxima a transportar medido en toneladas
    `public Remolque(int cargaMaxima) { … }`
  - El *Camión* permite conocer su carga máxima que es la misma que tiene su *Remolque*.
    `public int getCargaMaxima() { … }`

Se define la interface *Privado* como sigue:

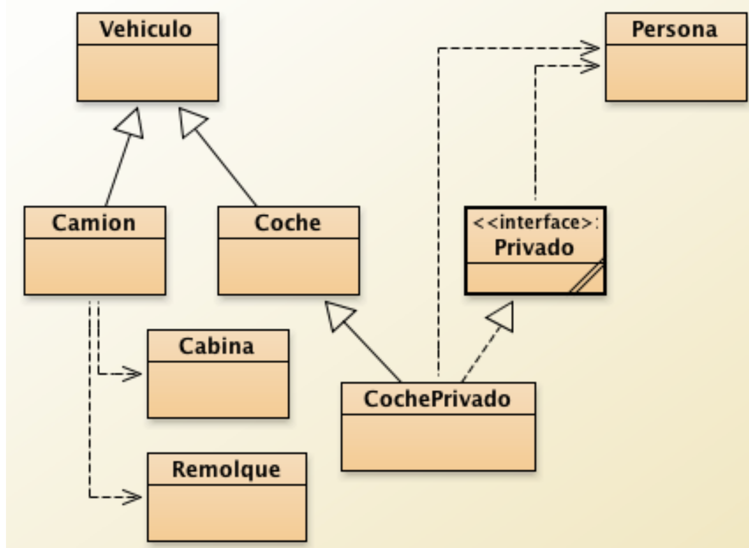
```
interface Privado  {
    // Devuelve la Persona propietaria del objeto
    public Persona getPropietario();
}
```

Se define la clase *CochePrivado* que extiende a la clase *Coche* e implementa el interface *Privado*.

```
public CochePrivado(int matricula, int potencia,
                     int numeroPlazas, Persona proprietario){ … }
```

Se pide:

1. Escriba la clase *Vehiculo*
2. Escriba la clase *Coche*
3. Escriba las clases *Cabina* y *Remolque*
4. Escriba la clase *Camión*
5. Escriba la clase *CochePrivado*



solución 1

```
public class Vehiculo {
    private int matricula;
    private int potencia;

    public Vehiculo (int matricula, int potencia) {
        this.matricula = matricula;
        this.potencia = potencia;
    }

    public int getMatricula() { return matricula; }

    public int getPotencia() { return potencia; }
}
```

solución 2

```
public class Coche
    extends Vehiculo {
    private int numeroPlazas;

    public Coche(int matricula, int potencia, int numeroPlazas) {
        super(matricula, potencia);
        this.numeroPlazas = numeroPlazas;
    }

    public int getNumeroPlazas() { return numeroPlazas; }
}
```

solución 3

```
public class Cabina {
    private int potencia;

    public Cabina(int potencia) {
        this.potencia = potencia;
    }

    public int getPotencia() { return potencia; }
}
```

```
public class Remolque {
    private int cargaMaxima;

    public Remolque(int cargaMaxima) {
        this.cargaMaxima = cargaMaxima;
    }

    public int getCargaMaxima() { return cargaMaxima; }
}
```

solución 4

```
public class Camion
    extends Vehiculo {
    private Cabina cabina;
    private Remolque remolque;

    public Camion(int matricula, Cabina cabina, Remolque remolque) {
        super(matricula, cabina.getPotencia());
        this.cabina = cabina;
        this.remolque = remolque;
    }

    public int getCargaMaxima() {
        return remolque.getCargaMaxima();
    }
}
```

solución 5

```
public class CochePrivado
    extends Coche
    implements Privado {
    private Persona propietario;

    public CochePrivado(int matricula, int potencia,
                        int numeroPlazas, Persona proprietario){
        super(matricula, potencia, numeroPlazas);
        this.propietario = propietario;
    }

    public Persona getPropietario() { return propietario; }
}
```