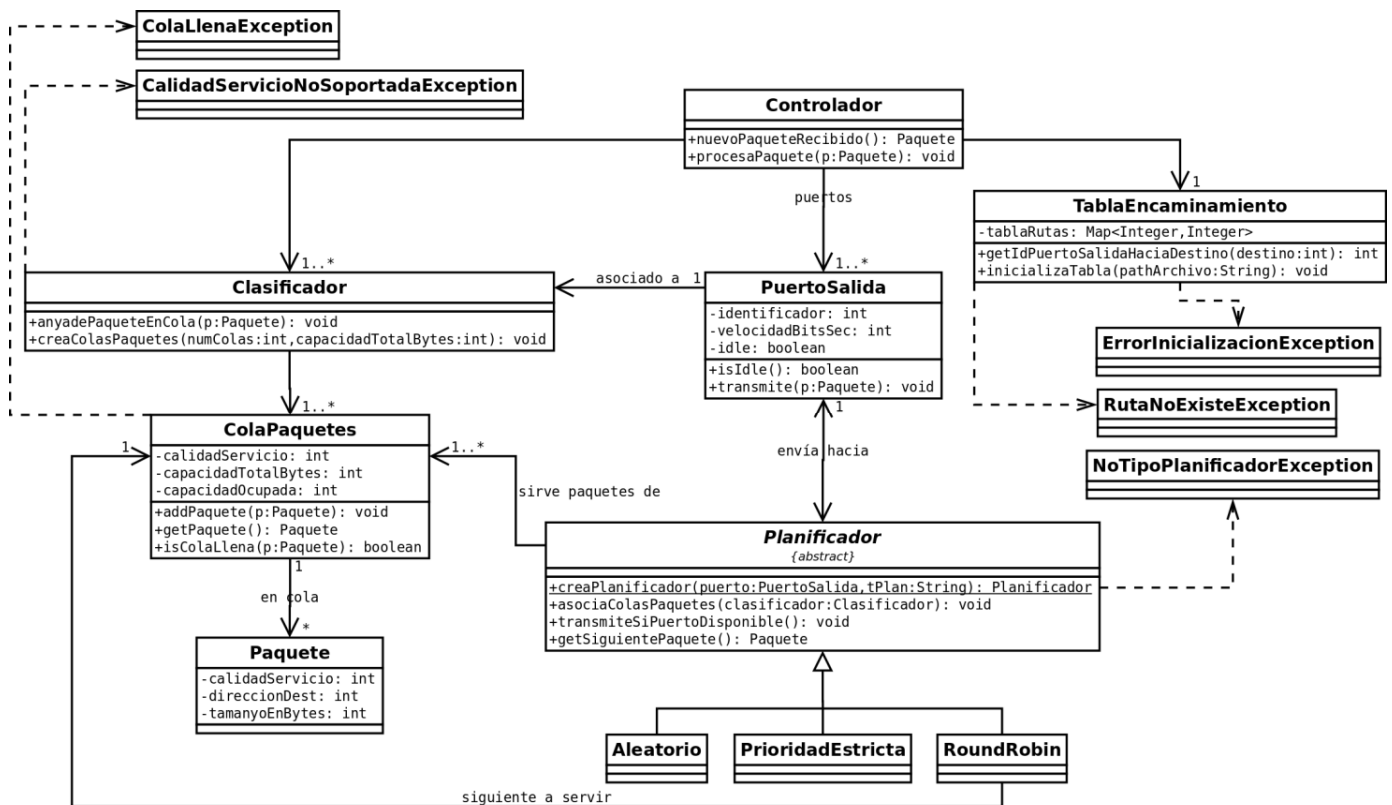


Curso 2015-2016. Cuatrimestre de otoño. 18 de Enero de 2016
 Programación Orientada a Objetos. Examen Final.
 Publicación notas provisionales: jueves 21 de Enero por la tarde.
 Revisión de examen: viernes 22 de Enero. Hora: 10:30 horas. Lugar: sala C6-E106.

Se pretende desarrollar el código que controle la actuación de un nodo de una red de conmutación de paquetes (Internet es una red de redes de conmutación de paquetes). En este tipo de redes, cuando un ordenador desea enviar información a otro ordenador, el ordenador remitente trocea la información en varias partes. Luego, encapsula cada una de las partes en un paquete (un paquete está formado pues por los bytes de información, además de bytes añadidos que se usarán para encaminarlo desde el ordenador remitente al ordenador destinatario) y transmite cada paquete a un nodo de la red. A partir de ese momento, cada nodo de la red, al recibir un paquete, procesa sus contenidos y lo reenvía a uno de sus nodos vecinos, consiguiendo que finalmente el paquete llegue al ordenador destino.

Obviamente, en este examen no se pretende desarrollar todo el código necesario. La figura que sigue muestra el diagrama de clases del código que se pretende desarrollar.



Cada nodo se conecta a uno o varios nodos de la red a través de sus puertos de salida (objetos instancia de la clase PuertoSalida en el diagrama). Cada puerto de salida queda conectado a un nodo vecino. Cada nodo se encarga de recibir paquetes y de reenviarlos a sus nodos vecinos por uno de los varios puertos de salida que posee.

Cada paquete contiene, a efectos de este examen, la siguiente información: la calidad de servicio asociada a su tránsito, la dirección de destino (valor entero) y su tamaño en bytes. La calidad de servicio permite priorizar el reenvío de ciertos paquetes en detrimento de otros que tienen asociadas calidades de servicios peores; la calidad de servicio se indicará mediante un número entero, siendo 1 el asociado a la máxima calidad. Números mayores indican calidades de servicios peores.

Cada nodo contiene un objeto instancia de la clase TablaEncaminamiento que encapsula un mapa (tabla de encaminamiento de ahora en adelante). Este mapa contiene parejas <dirección destino de paquete, identificador de puerto de salida>.

A cada puerto de salida se asocian varias colas en las que los paquetes esperan su turno para ser enviados al nodo vecino conectado a ese puerto. Cada una de las colas guardará paquetes la misma calidad de servicio (identificada también por el atributo calidadServicio de la clase ColaPaquetes, lo que exige que los atributos calidadServicio del objeto instancia de ColaPaquetes y del objeto instancia de Paquete sean iguales). Cada una de las colas tiene una capacidad máxima expresada en bytes (atributo capacidadTotal) y consecuentemente la suma de los tamaños de los paquetes almacenados en la cola NO puede superar dicha capacidad. Finalmente todo objeto instancia de cola guarda constancia de la suma de los tamaños de los paquetes almacenados en el atributo capacidadOcupada.

A cada puerto de salida se asocia un objeto instancia de la clase Clasificador (clasificador de ahora en adelante). El clasificador decide en cuál de las colas debe ser depositado un determinado paquete. El clasificador deposita un paquete en la cola cuya calidad de servicio es idéntica a la calidad de servicio indicada en dicho paquete. **Debe, por tanto, diseñarse el código de forma tal que sea inmediato acceder a una cola a partir del valor de calidad de servicio.**

A cada puerto de salida se asocia finalmente un objeto instancia de la clase Planificador (planificador de ahora en adelante). El planificador **gestiona las mismas colas que el clasificador** asociado al mismo nodo, pero en su caso **ordenadas en orden decreciente de calidad de servicio**: la cola que sirve a la mayor calidad de servicio (valor del atributo calidadServicio de la cola = 1) es la primera, la segunda cola sirve a la segunda mejor calidad de servicio y así sucesivamente. El planificador decide, cuando el puerto asociado está preparado para transmitir un nuevo paquete, de qué cola se extrae el paquete a transmitir. El planificador puede implementar tres políticas de selección de la cola, cada una de ellas implementada por las subclases de Planificador en el diagrama:

1. Los objetos instancia de la clase Aleatorio eligen la cola aleatoriamente.
2. Los objetos instancia de la clase PrioridadEstricta eligen de entre las colas no vacías aquella cuyos paquetes tengan la mayor calidad de servicio.
3. Los objetos instancia de la clase RoundRobin implementan una estrategia de turno rotatorio, esto es, primero intentan enviar un paquete de la primera cola, después uno de la segunda, después uno de la tercera y así sucesivamente. Cuando se ha intentado enviar un paquete de la última cola, se vuelve a comenzar por la primera. Para identificar la cola que “dispone del siguiente turno”, esto es, la cola a la que el planificador irá a buscar el siguiente paquete, los objetos instancias de esta clase deben disponer de mecanismos para identificar la siguiente cola a servir (observad la asociación entre RoundRobin y ColaPaquetes en el diagrama de clases).

Finalmente cada nodo posee un objeto instancia de la clase Controlador (controlador de ahora en adelante). Es el objeto que recibe los paquetes y gestiona la actuación del resto de componentes del nodo para que éstos salgan del nodo por el puerto pertinente.

Cuando un paquete llega al controlador, éste debe dilucidar el puerto por el que el paquete debe salir del nodo. Para ello, el controlador inspecciona el destino de dicho paquete y a continuación consulta la tabla de encaminamiento para determinar el identificador del puerto por el que debe ser reenviado.

Cuando se ha determinado el puerto de salida, el controlador ordena al clasificador de dicho puerto depositar el paquete en la cola pertinente.

Por su parte, cada planificador inspecciona constantemente el estado de su correspondiente puerto de salida, y cuando detecta que está inactivo (esto es, que puede reenviar un nuevo paquete) toma un paquete de las colas asociadas al nodo según la política que implemente y se lo da al puerto de salida para que éste lo reenvíe al nodo vecino, haciéndolo avanzar hacia su destino final.

El trabajo del planificador y el puerto de salida tiene lugar al mismo tiempo que el del controlador y el clasificador (es decir, el planificador le va enviando al puerto paquetes de sus colas al mismo tiempo que el controlador y el clasificador van depositando paquetes en dichas colas, si hay paquetes que depositar). **Sin embargo, esto no afectará al código que tendréis que desarrollar en este examen.**

PREGUNTA 1 (1,5 PUNTOS) A partir del diagrama de clases escribid para todas las clases (**excepto las clases excepción**) la parte de código de la definición que comienza con el “public class...” correspondiente a la declaración de los todos los atributos de las clases, incluyendo aquellos que aparecen al implementar las relaciones mostradas en el diagrama. **NO** debéis añadir nada más en el código de estas clases en vuestras respuestas a esta pregunta.

```
public class Controlador {
    private List<Clasificador> clasificadores;
    private Map<Integer, PuertoSalida> puertos;    //Clave: Id puerto
    private TablaEncaminamiento tabla;
}

public class Clasificador {
    private Map<Integer, ColaPaquete> colas;    //Clave: Clase servicio
}

public class PuertoSalida {
    private int identificador, velocidadBitsSec;
    private boolean idle;
    private Clasificador clasificador;
    private Planificador planificador;
}

public class TablaEncaminamiento {
    private Map<Integer, Integer> tablaRutas; //Clave: direccion destino
}

public class ColaPaquetes {
    private int calidadServicio, capacidadTotalBytes, capacidadOcupada;
    private List<Paquete> cola;
}

public class Paquete {
    private int calidadServicio, direccionDest, tamañoEnBytes;
}

public abstract class Planificador {
    protected PuertoSalida puerto;
    protected List<ColaPaquete> colas;
}
```

```

public class Aleatorio extends Planificador {
}

public class PrioridadEstricta extends Planificador {
}

public class RoundRobin extends Planificador {
    private ColaPaquetes siguiente;
}

```

PREGUNTA 2 (0,5 PUNTOS) Proponed constructores de las siguientes clases: Paquete, ColaPaquetes y TablaEncaminamiento

```

public class Paquete {
    public Paquete (int calidadServicio, int direccionDest, in
tamañoEnBytes) {
        this.calidadServicio = calidadServicio;
        this.direccionDest = direccionDest;
        this.tamañoEnBytes = tamañoEnBytes;
    }
}

public class ColaPaquetes {
    public ColaPaquetes (int calidadServicio, int capacidadTotalBytes){
        this.calidadServicio = calidadServicio;
        this.capacidadTotalBytes = capacidadTotalBytes;
        this.capacidadOcupada = 0;
        this.cola = new ArrayList<Paquete>();
    }
}

public class TablaEncaminamiento {
    public TablaEncaminamiento() {
        this.tablaRutas = new HashMap<Integer, Integer>();
    }
}

```

PREGUNTA 3 (0,5 PUNTOS) Implementad, en la clase TablaEncaminamiento, el siguiente método:

```

public int getIdPuertoSalidaHaciaDestino(int destino) throws
RutaNoExisteException;

```

Este método admite como argumento de entrada un entero identificando el destino de un cierto paquete y devuelve un entero identificando el puerto por el que el paquete deberá ser reenviado. Si la tabla no contiene ninguna pareja que tenga como clave el valor del argumento, lanza una excepción instancia de RutaNoExisteException.

```

public class TablaEncaminamiento {
    public int getIdPuertoSalidaHaciaDestino (int direccionDest) throws
RutaNoExisteException {
        Integer idPuertoSalida = this.tablaRutas.get(direccionDest);
        if (idPuertoSalida == null) {

```

```
        throw new RutaNoExisteException();
    }
    return idPuertoSalida;
}
}
```

PREGUNTA 4 (0,5 PUNTOS) Implementad, en la clase ColaPaquetes, el siguiente método:

```
public Paquete getPaquete();
```

Este método **extrae** el primer paquete de la cola en cuestión y lo devuelve. Si la cola está vacía devuelve null.

NOTA: recordad que las clases que implementan el interface List<E> disponen del método:

```
public E remove(int index);
```

Que extrae el elemento que ocupa la posición index dentro de la lista. Esta operación devuelve el elemento extraído. Si el valor de index está fuera de rango ($\text{index} < 0$ || $\text{index} \geq \text{size}()$) lanza una excepción `IndexOutOfBoundsException`

```
public class ColaPaquetes {
    public Paquete getPaquete() {
        if (this.cola.size() == 0) {
            return null;
        }
        else {
            return this.cola.remove(0); //Eliminamos paquete y lo devolvemos
        }
    }
}
```

PREGUNTA 5 (1 PUNTO) Implementad, en la clase Aleatorio, el siguiente método:

```
public Paquete getSiguientePaquete();
```

Este método devolverá el primer paquete de una de las colas elegida al azar. Si la cola seleccionada está vacía, devolverá null.

NOTA: para elegir una de las colas al azar cread un objeto instancia de la clase Random (usando su constructor con lista de argumentos vacía) y utilizad su método:

```
public int nextInt(int bound);
```

Este método devuelve un entero pseudoaleatorio entre 0 (inclusive) y el valor bound-1 (inclusive).

Este método puede, por tanto generar números que permitan identificar una de las colas servidas por el objeto instancia de clase Aleatorio.

```
public class Aleatorio extends Planificador {
    public Paquete getSiguientePaquete() {
        Random rnd = new Random();
        int pos = rnd.nextInt(this.colas.size());
        return this.colas.get(pos).getPaquete();
    }
}
```

```
}  
}
```

PREGUNTA 6 (1 PUNTO) Implementad, en la clase PrioridadEstricta, el siguiente método:

```
public Paquete getSiguientePaquete();
```

Este método devolverá el primer paquete de una de las colas no vacías, concretamente aquella en la que se guarden los paquetes con la calidad de servicio más alta. Si todas las colas están vacías, el método devolverá null.

NOTA: recordad que para el planificador las colas están ordenadas en orden decreciente de calidad de servicio.

```
public class PrioridadEstricta extends Planificador {  
    public Paquete getSiguientePaquete() {  
        for (ColaPaquetes cola : this.colas) {  
            Paquete p = cola.getPaquete();  
            if (p != null) {  
                return p;  
            }  
        }  
        return null;  
    }  
}
```

PREGUNTA 7 (1 PUNTO) Implementad, en la clase RoundRobin, el siguiente método:

```
public Paquete getSiguientePaquete();
```

Este método devolverá el primer paquete de la cola que indique el turno rotatorio explicado anteriormente. Si dicha cola está vacía, el método devolverá null. Tanto si la cola está vacía como si no, este método debe actualizar el indicador de la cola de la que se intentará extraer el próximo paquete.

```
public class RoundRobin extends Planificador {  
    public Paquete getSiguientePaquete() {  
        Paquete p = this.siguiente.getPaquete();  
        int pos = this.colas.indexOf(this.siguiente);  
        if (pos == this.colas.size()) {  
            this.siguiente = this.colas.get(0);  
        }  
        else {  
            this.siguiente = this.colas.get(pos+1);  
        }  
        return p;  
    }  
}  
  
/* En caso de desconocerse el método indexOf() para obtener la posición de la  
cola siguiente en la lista de colas, se podría haber obtenido a partir de su  
calidad de servicio, puesto que las colas del planificador deben estar ordenadas  
por calidad de servicio [1,..,numColas]. Es decir:  
int pos = this.siguiente.getCalidadServicio()-1; */
```

PREGUNTA 8 (0,5 PUNTOS) Implementad, en la clase Clasificador, el siguiente método:

```
public void anyadePaqueteEnCola (Paquete p) throws
CalidadServicioNoSoportadaException, ColaLlenaException;
```

Este método intenta añadir el paquete en aquella cola cuya calidad de servicio coincide con la calidad de servicio del paquete. Si no hay ninguna cola para la calidad de servicio del paquete, el método lanza una excepción instancia de `CalidadServicioNoSoportadaException`. Si esa cola existe pero no queda espacio suficiente en ella para guardar el paquete, el método lanza una excepción instancia de `ColaLlenaException`.

```
public class Clasificador {
    public void anyadePaqueteEnCola (Paquete p) throws
CalidadServicioNoSoportadaException, ColaLlenaException {
        ColaPaquetes cola = this.colas.get(p.getCalidadServicio());
        if (cola == null) {
            throw new CalidadServicioNoSoportadaException();
        }
        cola.addPaquete(p);
    }
}
```

PREGUNTA 9 (0,5 PUNTOS) Implementad, en la clase Controlador, el siguiente método:

```
public void procesaPaquete(Paquete p) throws RutaNoExisteException,
CalidadServicioNoSoportadaException, ColaLlenaException;
```

Este método intenta dilucidar el puerto de salida por el que el paquete pasado como argumento debe ser reenviado. Una vez identificado el puerto, ordena al clasificador asociado a dicho puerto que lo deposite en la cola pertinente. Si la tabla de encaminamiento no contiene ninguna anotación para el destino del paquete, el método lanza una excepción instancia de `RutaNoExisteException`. Además este método propaga cualquier excepción que se lance cuando ordena al clasificador añadir el paquete en una cola.

```
public class Controlador {
    public void procesaPaquete(Paquete p) throws RutaNoExisteException,
CalidadServicioNoSoportadaException, ColaLlenaException {
        int idPuerto = this.tabla.getIdPuertoSalidaHaciaDestino
(p.getDireccionDest());
        PuertoSalida puerto = this.puertos.get(idPuerto);
        Clasificador clasificador = puerto.getClasificador();
        clasificador.anyadePaqueteEnCola(p);
    }
}
```

PREGUNTA 10 (0,75 PUNTOS) Implementad, en la clase Planificador, el siguiente método:

```
public static Planificador creaPlanificador(PuertoSalida puerto, String tPlan)
throws NoTipoPlanificadorException;
```

Este método crea un planificador instancia de `Aleatorio` si el argumento `tPlan` es el string "aleatorio". Crea un planificador instancia de `PrioridadEstricta` si el argumento `tPlan` es el string "prioridad". Crea un planificador instancia de `RoundRobin` si el argumento `tPlan` es el string "RR". Este método asocia al planificador creado al puerto de salida que se pasa a través del argumento `puerto`. Si el argumento `tPlan` es un String con un contenido distinto a los tres valores indicados, el método lanza una excepción instancia de `NoTipoPlanificadorException`.

```

public abstract class Planificador {
    public static Planificador creaPlanificador(PuertoSalida puerto, String
tPlanificador) throws NoTipoPlanificadorException {
        Planificador planificador = null;
        if (tPlanificador.equals("aleatorio") {
            planificador = new Aleatorio(puerto);
        }
        else if (tPlanificador.equals("prioridad") {
            planificador = new PrioridadEstricta(puerto);
        }
        else if (tPlanificador.equals("RR") {
            planificador = new RoundRobin(puerto);
        }
        else {
            throw new NoTipoPlanificadorException();
        }
        puerto.setPlanificador(planificador);
        return planificador;
    }
}

```

PREGUNTA 11 (0,75 PUNTOS) Implementad, en la clase Clasificador, el siguiente método:

```

public void creaColasPaquetes(int numColas, int capacidadTotalBytes);

```

Este método crea tantas colas de paquetes como indica el valor del argumento numColas. Todas ellas tendrán una capacidad en bytes igual al valor del argumento capacidadTotalBytes. Las colas creadas deben quedar asociadas al objeto instancia de Clasificador que posee este método. **Los valores de calidad de servicio para las colas irán de 1 a numColas.** La asociación debe ser tal que minimice el código a generar para, dado un valor de calidad de servicio, acceder a la cola destinada a guardar paquetes con esa calidad de servicio.

```

public class Clasificador {
    public void creaColasPaquetes(int numColas, int capacidadTotalBytes) {
        for (int calServ = 1; calServ <= numColas; calServ++) {
            ColaPaquetes cola = new ColaPaquetes(calServ, capacidadTotalBytes);
            this.colas.put(calServ, cola);
        }
    }
}

```

PREGUNTA 12 (0,75 PUNTOS) Implementad, en la clase Planificador, el siguiente método:

```

public void asociaColasPaquetes(Clasificador clasificador);

```

Este método asocia las colas que previamente ha creado el objeto clasificador pasado en el argumento clasificador, con el objeto Planificador que posee a este método. En esta asociación las colas deben quedar ordenadas en orden decreciente de clase de servicio.

```

public class Planificador {
    public void asociaColasPaquetes(Clasificador clasificador){
        int numColas = clasificador.getNumeroColas();
    }
}

```



```

        for (int calServ = 1; calServ <= numColas; calServ++) {
            ColaPaquetes cola = clasificador.getColaPaquetes(calServ);
            this.colas.add(cola);
        }
    }
}

/* Se han asumido 2 métodos a implementar en Clasificador: getNumeroColas()
devuelve el número de parejas <clave,valor> en el mapa de colas de paquetes que
gestiona el clasificador; getColaPaquetes(calServ) devuelve la cola de paquetes
con la calidad de servicio proporcionada como parámetro de entre las colas que
gestiona el clasificador */

```

PREGUNTA 13 (0,75 PUNTOS) Implementad, en la clase TablaEncaminamiento, el siguiente método:

```

public void inicializaTabla (String pathArchivo) throws IOException,
ErrorInicializacionException;

```

Este método lee de un archivo de texto (cuyo nombre se pasa por el argumento pathArchivo) los detalles de la tabla de encaminamiento. El archivo está constituido por una serie de líneas de texto cuyos contenidos siguen el siguiente patrón:

DirDestino IdPuertoSalida

Donde DirDestino será la representación textual de un entero que indica una dirección e IdPuertoSalida será la representación textual de un entero que indica el identificador del puerto de salida por el que hay que reenviar todo paquete enviado a la dirección antes mencionada.

El método debe leer este archivo de texto y debe construir la tabla de encaminamiento. Si se produce algún error en el parseado de alguna de las representación textuales, el método lanzará una excepción instancia de ErrorInicializacionException.

NOTA: Recordad que la clase Integer contiene el método:

```

public static int parseInt(String s) throws NumberFormatException;

```

Este método devuelve un objeto Integer que encapsula el entero cuya representación textual se pasa en el argumento s. Si este argumento no contiene una representación textual de un entero, el método lanza la excepción instancia de NumberFormatException.

Para acceder al valor entero encapsulado en un objeto instancia de Integer debe utilizarse el método:

```

public int intValue();

```

```

public class TablaEncaminamiento {
    public void inicializaTabla (String pathArchivo) throws
ErrorInicializacionException, IOException
    {
        try {
            FileInputStream fis = new FileInputStream (pathArchivo);
            InputStreamReader isr = new InputStreamReader (fis);
            BufferedReader br = new BufferedReader (isr);
            String line = br.readLine();
            while (line != null) {
                String[] partes = line.split(" ");

```

```
        int direccionDest = Integer.parseInt(partes[0]);
        int idPuerto = Integer.parseInt(partes[1]);
        this.tabla.put(direccionDest, idPuerto);
        line = br.readLine();
    }
    br.close();
}
catch(NumberFormatException e) {
    throw new ErrorInicializacionException();
}
}
```