UPSAM 2012-2013

# Programación. Tema 2 Prácticas obligatorias

**Normativa:**

- Las siguientes prácticas **son obligatorias** para aprobar la asignatura y tienen que **ser entregadas y explicadas antes de la fecha fijada** por el profesor.
- El material a entregar será un **diagrama UML** y **el código fuente de la práctica en el lenguaje requerido.**
- **No se corregirán prácticas que no compilen.**
- Las prácticas **se realizará de forma individual**.

## Práctica 1. Java

Represente un **Semáforo de Tráfico** mediante una clase. El estado del semáforo podrá ser ROJO, AMBAR y VERDE. La clase deberá permitir cambiar el estado interno del semáforo mediante una **única** operación denomina **cambiar estado**. Esta operación modificará en cada llamada el estado interno del semáforo de la siguiente forma (VERDE->AMBAR->ROJO→ VERDE…). Además la clase semáforo debe definir una operación que permita obtener el valor del estado actual.

**Se pide**

- Representación UML de la clase Semáforo
- Diseñe una clase que represente un semáforo (use un **tipo enum** para representar el estado)
- El estado inicial del semáforo debe ser VERDE
- Mediante un programa principal pruebe la siguiente secuencia
    - Crear un semáforo, cambiar su estado interno a ROJO, mostrar por pantalla el estado interno del semáforo

**NOTA:** Para esta práctica investigue como declarar y usar un tipo enum en java

## Práctica 2. Java

Represente un **Servidor** mediante una clase. Para ello considere la siguiente información

- Un servidor escucha solicitudes de conexión en un puerto determinado (Ej 8080) y en una dirección IP (Ej 10.16.1.20 use una cadena para representarla)
- El nombre del servidor representa el servicio que ofrece (Ej servidor de correo, servidor de archivos, servidor de base de datos, etc).
- Un servidor mantiene internamente un recuento del número de hilos activos en cada momento
- La capacidad máxima del servidor ( en número de hilos) viene definida por la constante MAX_CON ( que se configura al crear el servidor)
- Cada hilo se corresponde con una solicitud de servicio activa. El servidor no atenderá solicitudes si ha llegado al límite de su capacidad MAX_CON

**Se pide:**

- Representación en UML de la clase Servidor
- Implemente en Java la clase Servidor
- Defina en la clase Servidor la **operación solicitar servicio**, cuyo efecto es el de incrementar en uno el número de hilos activos que están procesando solicitudes dentro del servidior o devuelve -1 si es rechazada
- Defina en la clase Servidor **la operación estadoCapacidad** que devuelve la capacidad actual en cada momento
- Realice un programa principal que basándose en la clase Servidor realice lo siguiente

    - Crear dos servidores : servidor base de datos (SBD), servidor de archivos(SA) con capacidades máximas de 3 y 5 hilos respectivamente
    - Realizar 4 solicitudes de servicio al servidor de base de datos y 2 al de archivos
    - **Mostrar por pantalla la capacidad actual de cada servidor**

## Práctica 3. Java

Se requiere que la Cuenta Bancaria **(ver prácticas guiadas)** tenga asociada además información de la nómina del cliente (sólo será necesario la cantidad mensual cobrada) de manera que **se pueda calcular el riesgo** de determinadas operaciones. Para ello se incorpore a la cuenta el **método *calcularRiesgo (float cuota-prestamo)*** que devolverá **el grado de riesgo** que tiene el préstamo a un cliente en función de su nómina. La lógica de este método es la siguiente:

- Si <cuota_prestamo> representa el 50% o más de la nómina ,el riesgo es alto
- Si <cuota_prestamo> representa menos del 50% y más del 31% ,el riesgo es medio
- Si <cuota_prestamo> representa 30% o menos ,el riesgo bajo

**Se pide:**

- Representación en UML de la clase CuentaBanacaria
    - Diseñe un programa principal que dada una CuentaBancaria (saldo=100 ,interés=0.0.3,nomina=1000)
    - **Que solicite por teclado una cuota y compruebe el riesgo** que representa esa cantidad frente a la nómina de la cuenta indicada.

## Practica 4.C++

Diseñe una clase C++ que represente una Calculadora. La calculadora es un poco especial y debe permitir realizar lo siguiente:

- Realizar **operaciones aritméticas** :suma, resta, división ,multiplicación
- Tienen una **operación** denominada **deshacerOperacion** que **deshará sólo la última** operación realizada.
- La operación **reset**. Que establece a cero el estado interno de la calculadora.
- Para ver el resultado de la operación se utilizará un método denominado **verestado** que devuelve el valor de la calculadora tras la **última operación** realizada.

**IMPORTANTE**

El **estado interno** de la calculadora **debe poder representar la última operación realizada y el resultado antes de hacer esta última operación.**

La **calculadora opera con números reales** (tipo double)

Todas **las operaciones aritméticas son procedimientos.**

Para ver el resultado hay que invocar al método verResultado()

Considerando que se trabaja con una notación infija *<operando1> operaciónX <operando2>*. La clase calculadora **debe definir dos prototipos** diferentes para las funciones aritméticas:

- **operacionX(operando1,operando2) :**
    - Si el estado interno es diferente de 0 se pone a 0 antes de hacer la operación
    - La función **resuelve internamente** la expresión **<**operando1> operaciónX <operando2>

- **operaciónX(operando2):** Este método considera el estado interno de la última operación como operando 1 .
- **X representa las operaciones posibles que serán:**suma(SUM),resta(RES),multiplicación(MUL),división(DIV)

Veamos a continuación todo esto con un ejemplo .Si queremos calcular ((1+5)-3) la secuencia de invocación de operaciones sobre la calculadora será:

- Operación sumar 1 y 5
- Operación restar 3 //Toma el **resultado almacenado internamente** como operando1 y le suma 3.
- Ver el resultado //Mostraría 3
- Deshacer operación //Devolvería 6
- Deshacer operación // Sigue devolviendo 6 no deshace 1+5

**Nota importante:** Deshacer operación **sólo puede mostrar el resultado anterior a la última llamada** de operacionX realizada que en el ejemplo anterior sería operacionRES(3).

**Se pide:**

- Diagrama UML de la clase Calculadora
- Un programa principal que pruebe la siguiente secuencia de operaciones
  - **(((((((5+1)-3)deshacerOP)+6)/4)deshacerOP)+12) → resultado 24**

## Práctica 4. Java

Un datagrama IP es una unidad de transferencia y organización en las redes de comunicación IP .El datagrama IP se divide en campos de control (cabecera) y datos .La siguiente figura muestra el detalle de su estructura.

| | | | |
|---|---|---|---|
| Versión | Longitud de cabecera | Tipo de Servicio | Longitud total |
| Identificador | | Flags | Offset |
| TTL | Protocolo | Checksum de cabecera | |
| Dirección IP origen del datagrama | | | |
| Dirección IP destino del datagrama | | | |
| Opciones | | | Relleno |
| Datos | | | |

Figura. Estructura de un datagrama IP

**Considerando sólo la parte de control**, nos centraremos **sólo en los siguientes campos**:

| Campo de control | Descripción y observaciones |
|---|---|
| **Longitud total:** | Tamaño en bytes del datagrama(cabecera y datos). Máximo valor 65535 octetos |
| **Flag DF** | **Simplificamos** este campos a los siguiente:<br>○ Si vale DF=0 significa que se puede fragmentar<br>○ Si vale DF=1 significa que no se puede fragmentar |
| **Dirección IP origen y destino** | Una dirección IP es una etiqueta numérica que identifica, de manera lógica y jerárquica, a una interfaz (elemento de comunicación/conexión) de un dispositivo |

| | |
|---|---|
| **(DIR.IP ORIGEN Y DIR.IP.DESTINO)** | (habitualmente una computadora) dentro de una red que utilice el protocolo IP (Internet Protocol), A través de Internet, los ordenadores se conectan entre sí mediante sus respectivas direcciones IP.<br>Ejemplo de dirección IP destino 192.146.123.45<br>o IMPORTANTE :Suponiendo siempre una **red de clase C** tenemos que 192.146.123 es la **dirección de red** |
| **TTL** | Time to live .Indica el máximo número de enrutadores que un paquete puede atravesar. Cada vez que algún nodo procesa este paquete disminuye su valor en 1 como mínimo, una unidad. |

**NOTA IMPORTANTE**

Las **direcciones IP** serán representadas **con tipo cadena s (Ej "192.156.124.45")**

**Para obtener la red** de la dirección IP investigue las funciones lastIndexOf(…) y substring(…) de la clase String en Java.Para ello considere que **la dirección de red** de la IP **"192.156.124**.45**" sería "192.156.124"**

La clase **Datagrama** deberá contemplar la siguiente funcionalidad:

- Construir un Datagrama con todos y cada uno de los campos
- Que se pueda modificar el estado de la propiedad TTL del datagrama.
- Obtener el valor de longitud del datagrama.
- Y comprobar si el paquete se puede fragmentar. El método devolverá true si se puede y false en otro caso

**Se pide:**

- **clase UML de la clase DatagramaIP**
- Realice una clase denominada **ProgramaPrincipal** que realice lo siguiente:
  - Construya un datagrama (46000,1,"10.16.0.2","10,16.0.15",5)
  - Cambie el estado del su TTL a 4.
  - Imprima por pantalla si el datagrama de puede fragmentar.
  - Imprimir por pantalla la **dirección de red de la dirección origen**