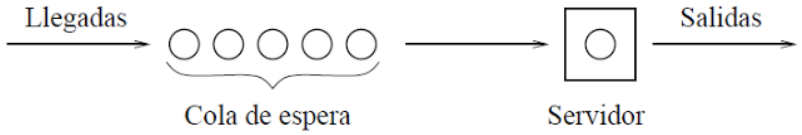
**Universidad del Cauca**
**Departamento de Sistemas**
**Ingeniería de Software II**

14 de enero de 2013 Profesor Julio Ariel Hurtado Alegría

# *Examen Final*

## 1. Abstracción y Reutilización - 1.0 Punto

Considere un simulador orientado a eventos genérico, es decir un simulador esqueleto que simula un rango amplio de sistemas en los que hay solicitudes de servicios y servidores, cuando los servidores están ocupados las solicitudes típicamente se ponen en una cola y son atendidos en orden de llegada. El simulador contiene una cola de eventos y una variable que indica la hora de simulación(contador).




Cada evento tiene grabado el tiempo que indica el momento en que éste debe ocurrir. La cola contiene eventos, los cuales son almacenados en orden ascendente respecto al tiempo en que el evento ocurre. El simulador orientado a eventos genérico ejecuta el siguiente ciclo infinito en el método simular de la clase SimuladorGenerico:

```
abstract class SimuladorGenerico
 {
  protected Estado nuevoEstado, viejoEstado;
  protected double tiempoSimulado;
  protected ColaEventos colaEventos;
  protected Evento eventoActual;
  public abstract Estado cambiar(Estado viejoEstado, Evento evento);
  public abstract void actuar(Estado nuevoEstado, Estado viejoEstado);
  public void simular()
   {
    while (true)
    {
```

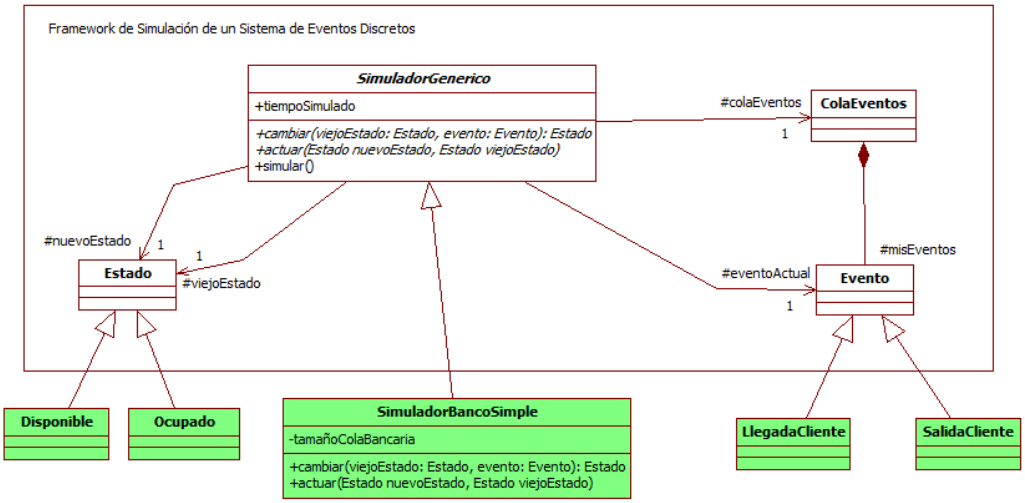



Figura 1: Solución al Punto 1a

```
    tiempoSimulado = colaEventos.top().getTiempo();
     do
     {
        eventoActual = colaEventos.pop();
        nuevoEstado = cambiar(viejoEstado, eventoActual);
        actuar(nuevoEstado, viejoEstado);
        viejoEstado = nuevoEstado;
            } while(tiempoSimulado <= colaEventos.top().getTiempo());
    }
   }
}
```

a) En la clase SimuladorGenerico descrita arriba, considere que Usted tiene que escribir un simulador para un sistema específico, por ejemplo la clase SimuladorBancoSimple (Un único Cajero, una única Fila (de clientes), eventos de llegada de un cliente y eventos de salida de un cliente de la atención del cajero). Dibuje las jerarquías de clases considerando el simulador específico y genérico, también considere en abstracto y en concreto los posibles eventos y la lógica específica del simulador del banco. Solución se presenta en la Figura a.

b) El diseño del simulador orientado a eventos genérico esta basado parcialmente sobre un patrón de diseño. ¿Cuál es el patrón? sustente su respuesta mediante

un diagrama de clases y un diagrama de secuencia. Si está inseguro, sustente de la misma forma otras alternativas que Usted considere. Solución: el patrón es el Plantilla. El diagrama de clases es el de la Figura refsolucionPunto1, la clase Plantilla es la Clase *SimuladorGenerico*, el método plantilla es *simular()* y los métodos que representan los pasos del algoritmo a ser redefinidos son *cambiar()* y *actuar*. El diagrama de secuencia se muestra en la Figura 2

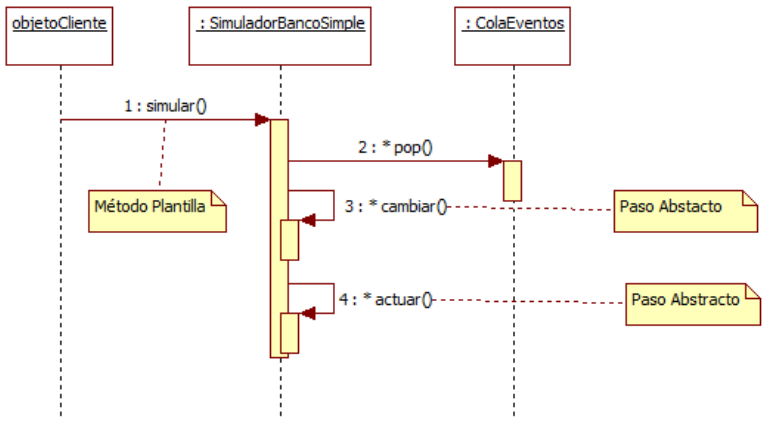
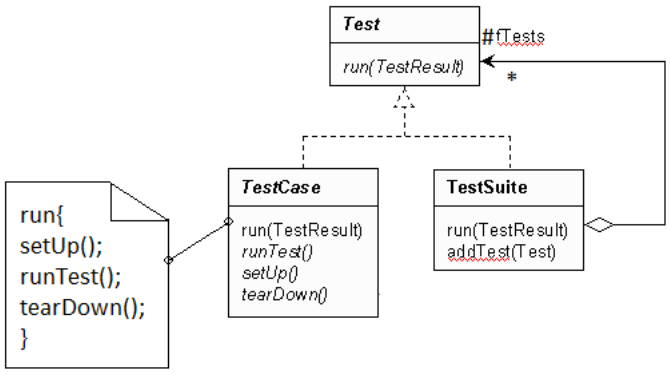



Figura 2: Solución a parte del Punto 1b

# 2. Reconocimiento de Patrones de Diseño Valor 1.0

JUnit es un framework para realizar y automatizar pruebas de aplicaciones Java. Es decir, JUnit soporta la disciplina de pruebas dentro del ciclo de desarrollo. La siguiente figura presenta la versión original creada por Erich Gamma (el mismo de los patrones GoF) y Kent Beck(El mismo de XP).

Una prueba es programada en una clase derivada de la clase *TestCase*, específicamente en el método *runTest()*. Antes de correr la prueba el framework está programado (método *setUP()*) para fijar unas condiciones iniciales de ejecución (inicializar algunos objetos y reservar recursos) y después de correr la prueba, está programado (método *tearDown()*) para fijar unas condiciones finales de ejecución, útil por ejemplo para liberar recursos. Como la idea en procesos como XP, dónde existe la propiedad colectiva de código, es facilitar el cambiar contínuamente el código, pero garantizar




la integridad, toda clase implementada requiere de un conjunto de casos de prueba, sumado a la cantidad de clases, se hace necesario manejar conjuntos de prueba y tratarlas en forma homogénea como si fuese una prueba más, en JUnit, se llama *TestSuite*. Una *TestSuite* permite la ejecución de varias pruebas.

a) Identifique dos patrones que está usando el framework JUnit. ¿Cuáles son los participantes? Solución: (1)El patrón Composité o Composición la Clase *TestCase* es un Hoja, la Clase *TestSuite* es el Compuesto y la clase *Test* es el ComponenteAbstracto que homogeiniza el trabajo con casos de prueba y suites de prueba respecto a la operación *run(TestResult)*. (2) El patrón Plantilla la clase *TestCase* es la clase Plantilla, el método *run(TestResult)* es el método plantilla y los métodos *runTest()*, *setUp()*,*tearDown()* son los pasos abstractos del algoritmo plantilla.

b) Implemente el método *run()* de la clase *TestSuite*. Solución minimalista con un arreglo:

```
public void run(TestResult t){
 for(i=0; i<fTest.size(); i++){
                    fTest[i].run(t);
 }
}
```

c) Explique el papel que cumple el principio de Hollywood y proponga la definición de un método Hook. Solución: el principio de Hollywood se cumple gracias al método template y el manejo homogéneo de las pruebas por el patrón composición, así es posible definir clases derivadas tanto de *TestCase*, cómo de *TestSuite*

y la lógica del framework los va a llamar gracias a las implementaciones concretas que se hagan de los métodos definidos en abstracto. Es decir, al crear nuevos componentes éstos no llaman al framework, sino al contrario el framework está preparado para soportar más elementos mientras no se viole el principio de Liskov(Se cumplan los contratos). Los métodos *setUP()* y *tearDown()* podrían implementarse haciendo nada, es decir, cómo métodos en blano pero concretos, ésto con el fin de facilitar pruebas simples que no requieren sino sólo la especificación en *runTest()*.

# 3. Caso de Aplicación de Patrones de Diseño - Valor 1.0

Hemos adquirido un sistema de gestión de bases de datos con una licencia para soportar 5 conexiones simultáneas. Con el fin de permitir muchas más conexiones simultáneas, aprovechando los tiempos muertos en los que las aplicaciones no requieren retener una conexión, se ha propuesto el desarrollo de una clase PoolConnections. ¿Qué patrón usaría para lograr esta solución? ¿Qué modificaciones se realizaron al patrón original para éste problema? Haga un diagrama de clases e implemente las clases principales por completo. La lógica de conexión real a la base de datos déjela indicada mediante un comentario. Solución: el patrón a utilizarse sería el singleton, dado que éste controla la existencia de una única instancia, éste será modificado para soportar la creación y administración de un conjunto finito de instancias. En éste caso la aplicación sería de 5 conexiones a un sistema de gestión de bases de datos. El diagrama de clases solución y el código que la programa se encuentra en la Figura 3.

# 4. Notación UML - Valor 1.0

Revise todos los modelos de ésta prueba, cada uso inadecuado de UML les restará 1 décima hasta un máximo de 10 décimas. Uso inadecuado se refiere a la equivocada utilización de clases(abstractas y concretas), interfaces, relaciones como herencia, realización, composición, agregación, asociación y dependencia. También incluye el uso adecuado de la navegación, los roles, la multiplicidad. Igualmente los diagramas de secuencia o colaboración usados deben tener en cuenta condicionales, repetición de mensaje, paso de parámetros. En otras palabras, el UML usado en los puntos anteriores debe estar impecable para ganarse este punto, así que por favor revisite

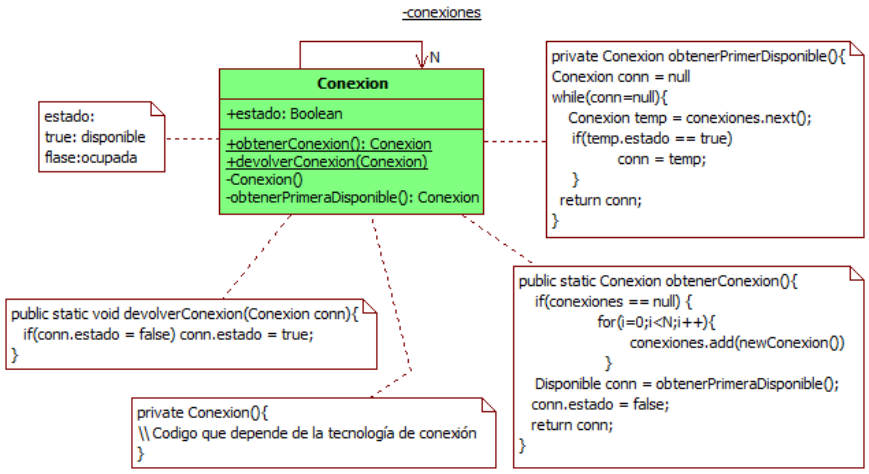



Figura 3: Solución al Punto 3

sus modelos y mejorelos lo que más pueda.

# 5. Arquitecturas de Software - Valor 1.0

El MVC es considerado un patrón arquitectónico, de acuerdo a la definición de arquitectura de software vista en clase determine cuales serían las estructuras, las componentes, los conectores, las propiedades externamente visibles que éste patrón expresa. Haga un gráfico a su gusto del patrón que concuerde con lo descrito y haga una descripción de la notación usada. Utilice este caso para mostrar la relación de la arquitectura de software con el diseño detallado, rescate el valor que tienen algunos principios de diseño orientado a objetos y los patrones de diseño. Solución, la Figura 4 vista en clase representa la estructura de comportamiento del patrón arquitectónico mvc junto con una notación informal. Los componentes son Modelo, Vista y Controlador. Los conectores son invocaciones explícitas(llamados a métodos), e implícitas (llamados no visibles programados a nivel abstracto siguiendo el principio de Hollywood). Las propiedades externamente visibles son: para el Modelo son consultar estado y cambiar estado, para el Controlador atender solicitudes de usuario y para la Vista dejarse seleccionar y notificar de los cambios del modelo. Aquí es posible observar el principio fundamental de separación de preocupaciones, a nivel orientado a objetos la implementación del patrón requiere principalmente de los principios

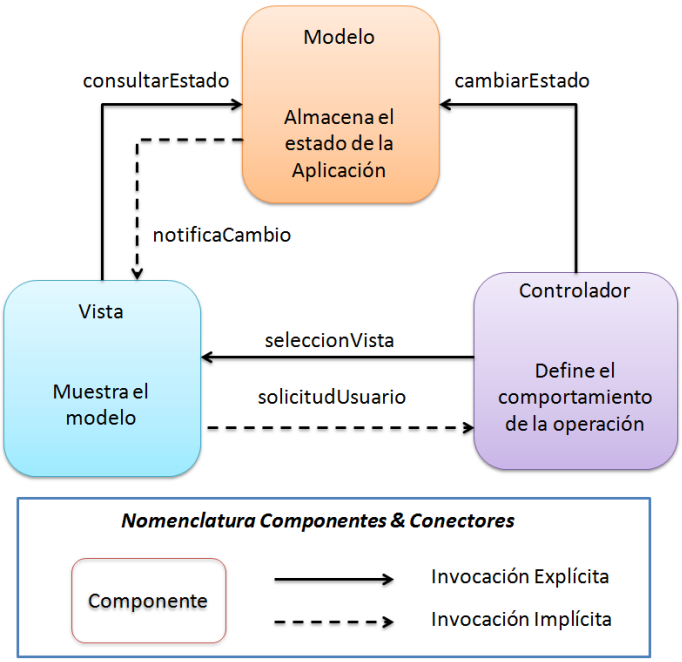



Figura 4: Patrón Arquitectónico MVC

de inversión de dependencias y de una sola responsabilidad. Los patrones permiten reutilizar conocimiento y permitir la implementación de soluciones arquitectónicas sofisticadas, en este caso los patrones Observador, Commando, Plantilla, Intermediario y de Fabricación Abstracta permiten construir una implementacíón completa y simple de utilizar(para los programadores)del patrón MVC.

*Espero haberlos motivado lo suficiente*
*para seguir aprendiendo más de la ingeniería de software*
Julio Ariel