Diseño y Programación Orientados a Objetos
1er. Semestre 2022

Tarea 1: Simulando Ventanas y Lámparas Domóticas

Lea detenidamente la tarea. **Si algo no lo entiende, consulte. Si es preciso, se incorporarán aclaraciones al final**. Esta interacción se asemeja a la interacción entre desarrolladores y clientes cuando algo no está del todo especificado.

Objetivos de la tarea

- Modelar objetos reales como objetos de software.
- Ejercitar la creación y extensión de clases dadas para satisfacer nuevos requerimientos.
- Reconocer clases y relaciones entre ellas en códigos fuentes Java.
- Ejercitar la compilación y ejecución de programas en lenguaje Java desde una consola de comandos.
- Ejercitar la configuración de un ambiente de trabajo para desarrollar aplicaciones en lenguaje Java, se pide trabajar con un IDE. Si bien IntelliJ es el sugerido, su grupo puede trabajar con otro IDE.
- Ejercitar la entrada y salida de datos.
- Manejar proyectos vía GIT (voluntario para esta tarea).
- Conocer el formato .csv y su importación hacia una planilla electrónica.
- Ejercitar la preparación y entrega de resultados de software (creación de makefiles, readme, documentación).
- Familiarización con desarrollos "iterativos" e "incrementales".

1.- Descripción General

Esta tarea busca practicar la orientación a objeto en un sistema domótico. La domótica busca la automatización de una vivienda o edificio.
En esta tarea usted modelará y programará un tipo de cortina motorizada (aquí un ejemplo) y un tipo de lámpara.

Modelo de una cortina “roller” motorizada

Aquí, el modelo de la cortina es simple, incluye un motor capaz de girar en ambas direcciones y una tela, Figura 1. Cada cortina está asociada a un canal de trabajo. Para fijar la apertura de la cortina se dispone de un control con botones para enviar mensajes “up”, “down” y “stop”. Cada control remoto de cortina tiene asociado un canal y a través de éste se define las cortinas que puede controlar. Al ser instaladas (al “nacer”) cada cortina parte talmente enrollada; es decir, 100% abierta.

Figura 1: Modelo simple de cortina motorizada y su control remoto

Cuando se acciona una vez el botón de subir cortina (comando “up” en corto sólo “U”), el motor comienza a girar en sentido correspondiente y como resultado de esto la tela extendida se reduce. Para detener el giro del motor se debe presionar el botón central (comando “stop”, en corto sólo “S”).
La rapidez angular del motor es un parámetro de éste (alfa). Su radio es fijo e igual a 4 [cm] para todos los motores de cortinas.

El largo total de la tela (“Length” o “L”) es un parámetro y puede variar de cortina a cortina.
Este tipo de cortinas tiene sensores “de fin de carrera” que detienen el motor cuando la tela está completamente enrollada y cuando está completamente extendida. Para no complicar el modelo, parte del estado de la tela será su nivel de apertura y será la tela la que detenga el giro del motor según cuando no puede seguir abriéndose o cerrándose. Por simplicidad cada giro del motor extenderá o reducirá la tela un mismo valor. Notar que éste debería depender del radio efectivo de la tela enrollada.

Modelo de una lámpara controlada
La lámpara de esta tarea permite ser encendida y cambiar el color de la luz a través de un control remoto, Figura 2. La luz se compone de tres colores básicos rojo, verde y azul, los cuales pueden ser ajustados para generar la percepción de otros colores. Cada color básico es representado por un valor que varia de 0 a 255. La intensidad es lograda variando el nivel de cada uno de los colores. Cuando una lámpara es instalada (“nace”), parte apagada y con intensidades 255 para cada color.

Figura 2: Modelo de una lámpara domótica y su control remoto

Cada lámpara tiene un canal asociado el cual no se mezcla con los canales de las cortinas. Un control de lámpara asociado a un canal controlará todas las lámparas de ese canal.
Al encender una lámpara (botón “power” o sólo “P”)[1], ésta ilumina con la intensidad y color que tenía previo a ser apagada. Se apaga con mismo botón (“power” o sólo “P”).
Cada vez que se presiona el botón asociado a un color (comandos “R U” por “red up”, “R D” por ”red down”, “G U”, “G D”, “B U”, “B D”), las lámparas de ese canal aumentan o disminuyen en un valor fijo e igual a 10 (o hasta saturar superior o inferiormente)[2] la intensidad de ese color. El incremento o decremento efectuado es un parámetro común para todas las lámparas. No es existen intensidades inferiores a 0 o superiores a 255 para cada color.

“La Nube”
No corresponde a un objeto físico sino a una aplicación en la nube donde cada aparato controlable demóticamente se registra y puede ser controlado por su correspondiente control. Así, en este modelo los controles envían mensajes a la nube y ésta los envía a cada dispositivo domótico.

Formato de archivos de Entrada y Salida
Para probar su tarea desde el main su programa crea varias cortinas y lámparas que luego va controlando según las acciones definidas en un archivo de texto de entrada. Para observar el correcto funcionamiento de todos los elementos, su programa debe enviar a la salida estándar el estado de cada objeto siguiendo un formato dado.

El formato para el archivo de entrada es rígido y el nombre del archivo es pasado como un argumento de la ejecución del programa. En sus primeras líneas se define el número de cortinas, lámparas y controles. Luego se ingresan parámetros que caracterizan cada cortina y lámpara. Posteriormente, hay un listado de comandos
Su formato es:
<#_de_cortinas> <#_de_lámparas> <#_controles_cortinas> <#_controles_lámparas>

[1] Cuando los controles tienen botón único de encendido y apagado, éste funciona conmutando el estado actual.
[2] Esta constante no fue definida en primera versión.

<alfa0> <length0> <canal0> … <alfaN_1> <lengthN_1> <canalN_1> // rapidez angular [rad/s], largo de la tela y canal de la primera hasta n-ésima cortina
<canal0>…..<canalL_1> // canales de la primera a la L-ésima lámpara
<canal0>. … <canalM_1> // canales del primer hasta el m-ésimo control de cortina.
<canal0>…. <canalK_1> // canales del primer hasta el k-ésimo control de lámpara.
<T [s] > <C | L> <#channel> <Command> // Tiempo entero y en segundos, Cortina o Lámpara, comando

Si no hay cortinas o lámparas, no hay líneas correspondientes a sus parámetros.
No se preocupe por sobreponerse a posibles errores del formato del archivo de entrada. Sí se espera que usted advierta por la salida estándar dónde se produjo un error. Como separador se sugiere usar "tabs" para mejor lectura y edición del archivo.

Por ejemplo, este archivo puede tener valores como:

Contenido del archivo	Explicación del ejemplo
2 1 1 1 2.1 1 1 2.5 1.5 1 0 1 0 0 C 1 D 1 L 0 P 3 C 1 S 7 L 0 R D 8 L 0 B U :	2 cortinas una lámpara y un control de c/u Ambas cortinas en mismo canal, 1 La única lámpara está en canal 0 Control de cortina en canal 1 (es consiste) Control de lámpara único y en canal 0. Partiendo de tiempo 0 [s] siguen las líneas que definen los comandos de cortinas y lámparas Completar con las líneas que quiera. Luego de ejecutar el último comando, el programa concluye.

Como salida, interesa conocer la evolución del porcentaje de apertura de cada cortina y el nivel de intensidad para cada color de las lámparas. Conforme se envían comandos a cortinas y lámparas, se pide enviar a pantalla el estado resultante para éstas. El formato pedido se conoce como .csv (comma-separated values). Si bien su nombre insinúa uso de comas como separadores, es posible usar otros caracteres también. Aquí usted usará "tabs". El tiempo mostrado por pantalla avanza en unidades cercanas a 1 segundo hasta que el archivo de entrada haya sido procesado completamente.
La primera línea es de encabezado para describir cada columna.

Time	RS0	…	RSN_1	L0R	L0G	L0B	…	LL_1R	LL_1G	LL_1B
<t>	<%>	…	<%>	<nº>	<nº>	<nº>	…	<nº>	<nº>	<nº>

Llamando a su programa DomoticaTest.java, la ejecución de su tarea sería:
$ java DomoticaTest "archivo de entrada.txt"

Para enviar la salida de su programa a un archivo, usar:
$ java DomoticaTest "archivo de entrada.txt" > salida.csv

2.- Desarrollo en Etapas
Para llegar al resultado final de esta tarea usted debe aplicar una metodología de tipo "Iterativa e Incremental" para desarrollo de software. Usted y su equipo irán desarrollando etapas donde los requerimientos del sistema final son abordados gradualmente. En cada etapa usted y su equipo obtendrá una solución que funciona para un subconjunto de los requerimientos finales o bien es un avance hacia ellos. En AULA se dispondrá el recurso para subir la solución correspondiente a cada etapa del desarrollo. **Su equipo deberá entregar una solución para cada una de las etapas** aun cuando la última integre las primeras. **El readme y archivo de documentación deben ser preparados**

solo para la última etapa. Prepare y entregue un makefile para cada una de las etapas. Esto tiene por finalidad, educar en la metodología iterativa e incremental.

2.1.- Primera Etapa: manejo de entrada y salida para una lámpara On/Off
En esta etapa se pide leer el formato completo del archivo de entrada el cual contendrá cero cortinas y una lámpara. La funcionalidad de la lámpara sólo permite encenderla y apagarla. Sus atributos red, green y blue variarán entre 0 y 255. Se generará y verificará la generación correcta de la salida correspondiente.
Use y/o complete las clases Lamp, LampControl, Cloud y Stage1 disponibles aquí. Usted no está obligado(a) a usar estos códigos. Están para su conveniencia y en caso de que le resulten útiles. Otras formas de estructurar el resultado hasta llegar a la etapa 4 también son válidos.

La clase Stage1 contiene el método main, en esta está la lógica para recorrer el archivo de entrada y crear instancias de Lamp, LampControl y Cloud como objetos con funcionalidades básicas. Esta clase es la encargada de simular el avance del tiempo y pedir a la instancia de Lamp generar la salida correspondiente.

Como salida esta etapa pide generar un archivo con formato pedido en 1.
Entregue las clases de esta etapa, incluya su makefile, el archivo de entrada usado y el de salida generado.

2.2.- Segunda Etapa: Motor Controlado de Cortina sin Tela
Para esta etapa se identifica la clase Motor la cual recibe los comandos desde la clase Cloud que a su vez son solicitados desde el control remoto. El motor gira conforme avanza el tiempo definido en main de la clase Stage2. El método main se encarga de informar a cada Motor el avance del tiempo.
Una diferencia importante entre una lámpara y una cortina es que, al presionar una vez, por ejemplo, “down” en el control de la cortina, su motor comienza a girar hasta que se presione “stop”. Aquí no hay tela aún y por ello no hay fin de carrera detectado.
Usted puede usar y/o completar las clases RollerShade, Motor, ShadeControl y Stage2 disponibles aquí.
Como salida se pide generar un archivo con el formato pedido en 1 excepto que en lugar de indicar el % de apertura de la cortina muestre 100 para giro en un sentido 0 detenido y -100 para giro en sentido contrario. Considere sólo cortinas como dispositivos domóticos posibles.
Entregue las clases de esta etapa, incluya su makefile, el archivo de entrada usado y el de salida generado.

2.3.- Tercera Etapa: Lámparas y Cortinas operando en conjunto y gráfica salida
A partir de la etapa previa incluya las modificaciones necesarias para completar la funcionalidad de la clase Lamp. En este caso el archivo de entrada puede contener lámparas (100% funcionales) y cortinas con sólo su motor.
Entregue las clases de esta etapa, incluya su makefile, el archivo de entrada usado y el de salida generado.

2.4.- Cuarta Etapa: Dispositivos domóticos con funcionalidad total.
En esta última etapa se espera alcanzar la funcionalidad descrita en la sección 1. Utilice un archivo de entrada con el formato señalado allí y genere un archivo de salida como el indicado en sección 1.
Para un archivo de entrada que incluya al menoso dos cortinas y una lámpara, envíe la salida a un archivo y usando una planilla de cálculo genere dos gráficas. En una se muestra la variación del porcentaje de apertura en el tiempo para cada cortina. En el otro se muestra la variación en el tiempo de las componentes de color de la lámpara.
Incluya el archivo de entrada usado, el archivo de salida obtenido y dos archivos (imágenes o pdf) con las gráficas obtenidas.

2.5.- Extra-crédito: Cierre de cortinas ante encendido de lámparas
Cree la clase LightSensor. Una instancia de LightSensor ordena cerrar las cortinas de un mismo canal cuando la suma de las componentes de color de lámparas de ese canal supera 512. Para agregar un LightSensor, en la línea de comando agregue el parámetro <channel>. Éste indica el canal a observar por el sensor.
Esta parte es voluntaria, su desarrollo otorga 10 puntos adicionales (la nota final se satura en 100).
Si desarrolla esta parte, indíquelo en su documentación y, además de la cuarta etapa, agregue un escenario donde se aprecie el efecto de este sensor.

3.- Elementos a considerar en su documentación
Entregue todo lo indicado en “Normas de Entrega de Tareas”.
Prepare un archivo makefile para compilar y ejecutar su tarea en aragorn con rótulo “run”. Además, incluya rótulos "clean" para borrar todos los .class generados. Los comandos a usar en cada caso son:
$ make /* *para compilar* */
$ make run /* *para ejecutar la tarea* */
$ make clean /* *para borrar archivos .class* */

En su archivo de documentación (pdf o html) incorpore el diagrama de clases de la aplicación (etapa 4).

4.- Nociones de simulación de fenómenos continuos
La idea básica usada en esta tarea es discretizar el tiempo. Cuando una acción se desarrolla en el tiempo, como el caso del giro de un motor, para cada instante discreto, congelamos el tiempo y pedimos a cada objeto de la simulación calcular cuál será su estado futuro (delta t más tarde) a partir del estado actual y las condiciones a que está sometido. Es importante congelar el tiempo y hacer este cálculo para cada objeto. Luego suponemos que el tiempo avanzó, actualizamos el tiempo (avanza en delta) y actualizamos el estado correspondiente a ese avance de tiempo; por ejemplo, actualizamos el nuevo largo para la tela de una cortina. Se debe operar así para representar adecuadamente interacciones entre objetos que cambian. Para eventos de cambio “instantáneos” como detectar un fin de carrera, éstos reflejan su efecto de manera inmediata.
Esta simple idea trabaja bien aquí y en muchas situaciones. La calidad de la simulación aumenta al usar un Δt lo suficientemente pequeño; sin embargo, si en muy pequeño, la simulación requerirá mucho cálculo y tomará más tiempo de ejecución.

Se puede jugar cambiando los parámetros de entrada o ver qué pasa si se permitiera cambiar de canal a un dispositivo domótico. Como este ramo no es el único donde usted debe rendir bien, se sugiere concentrarse sólo en lo pedido.