Unidad 1 – UML

Almacenamiento

Realice el diagrama de clases para representar las relaciones entre los elementos de un sistema de archivos: los archivos, los directorios y las particiones del disco rígido.

Familia

Realice el diagrama de clases para representar las relaciones entre un marido, su mujer y sus 3 hijos dentro de un sistema de árboles genealógicos.

Musicoteca

Un famoso músico nos contrata para desarrollar un sistema que sea capaz de organizar toda su música. El sistema debe manejar los álbumes que el artista posee. De los álbumes, se precisa su nombre, su artista, su año de edición, y su listado de canciones. De cada canción interesa su título, su duración y su posición dentro de un álbum. Cada artista debe poder dar una lista de las canciones que hizo, o los discos que tiene. Cada disco debe poder mostrar qué duración tiene y cuántas canciones tiene.

Banco

Realice el diagrama de clases para representar el sistema que utiliza el cajero de un banco donde suceden las siguientes interacciones:

Un cliente de un banco quiere extraer dinero de su cuenta de ahorro, se acerca a la ventanilla y le solicita al cajero la extracción de $100. El cajero, solicita al cliente su DNI y su número de cuenta. Luego, el cajero imprime el comprobante y pide al cliente que lo firme. Paso después, le entrega la suma de dinero solicitada.

Antes de irse, el cliente pide al cajero que le informe el saldo actual de su cuenta, y los últimos tres movimientos de la misma (dinero extraído, depositado, transferido, debitádo).

Farmacia

En una farmacia se calcula el precio de los remedios de forma diferente si se paga en efectivo o con tarjeta(10% de descuento efectivo, 5% recargo por tarjeta) y si se tiene o no obra social. Los remedios tienen nombre, precio de lista y necesitan o no receta.

Los clientes pueden o no tener obra social. Las obras sociales pueden decir el precio de un remedio que se les indique que será del 60% del precio de lista del remedio si este necesita receta, y de un 90% si no necesita. Tener en cuenta que exista una obra social vacía que devuelva siempre el 100% del precio de lista del remedio.

Mascota virtual

Se pide modelar una mascota virtual que sepa comer, jugar y decir si puede salir a jugar o no.

Se requiere además poder conocer qué tan feliz está una mascota, nivel representado con un número entero mayor o igual que 0, donde a mayor nivel, más feliz se siente la mascota. Una mascota puede estar aburrida, hambrienta o contenta, y su comportamiento depende de en qué estado esté.

Cuando una mascota come, pasa lo siguiente:

- Si está hambrienta, se pone contenta.
- Si está contenta, su nivel se incrementa en una unidad.
- Si está aburrida, entonces no le pasa nada, no cambia nada.

Cuando una mascota juega, pasa lo siguiente:

- Si está contenta, su nivel se incrementa en dos unidades.
- Si está aburrida, se pone contenta.
- No produce ningún efecto jugar con la mascota si esta hambrienta.

Una mascota puede jugar si está contenta o aburrida, si está hambrienta no.

Aclaración: no se puede consultar de ninguna manera el estado actual de la mascota.

Inscripción a materias

Los alumnos de una Universidad se inscriben a las materias que desean cursar a través de un sistema online que el establecimiento ofrece. El alumno ingresa al sistema utilizando un usuario y una contraseña, siendo el usuario el número de documento.

El sistema ofrece al alumno una lista de materias a las que él puede inscribirse en el cuatrimestre actual (no se puede anotar a materias de otro cuatrimestre).

El alumno elige las materias a las cuales inscribirse, y el sistema realiza las validaciones necesarias en cuanto a correlatividades.

Un alumno puede inscribirse a una materia si tiene aprobadas TODAS las materias correlativas. En caso necesario, el sistema informará al alumno que no pueda inscribirse a una materia, debido a que no tiene aprobadas ciertas materias

Facturación

Para realizar la facturación de los productos de un comercio adquiridos por sus clientes, se tienen en cuenta dos factores: la condición frente al IVA del cliente y la naturaleza de los productos.

Existen tres condiciones frente al IVA distintas a tener en cuenta:

- responsable inscripto, que abonará un IVA del 10,5%
- sujeto exento, que no abonará IVA
- consumidor final, que abonará un IVA del 21%

En cuanto a los productos, existen los productos nacionales y los importados, los cuales sufren un recargo del 10% sobre el precio de lista por tasas de aduana.

Diseñar un sistema que permita por cada venta, generar la factura correspondiente. La factura deberá contener información del cliente junto con el listado de productos comprados, y poder calcular el total a abonar.

Restaurante

Un conocido restaurante nos ha convocado para sistematizar parte de su funcionamiento, para lo cual ha aportado la siguiente información:

Cada uno de los mozos del local, atiende un grupo de mesas predeterminado. La información que se registra de los mozos es su nombre, CUIL, dirección y teléfono.

Los productos que se ofrece el local, son menús fijos, comida a la carta y bebidas, cada uno de los cuales se referencia por un código numérico, su nombre y precio.

Los menús fijos no incluyen la bebida y sobre su precio se aplica un porcentaje de descuento que el local fija diariamente. Los productos a la carta y las bebidas no tienen descuentos. Además de cada bebida se registra el stock mínimo y se va actualizando la existencia con cada pedido.Sin limitar la venta, el sistema debe informar cuando ésta sea menor que el stock mínimo.

Al ocuparse una mesa se habilita la misma registrándose el número, la fecha y el primer consumo. Los consumos siguientes en la misma mesa se registran a medida que se producen los pedidos, detallándose cada producto y la cantidad solicitada. En el momento de cerrar una mesa atendida por un mozo se registra la forma de pago, efectivo o tarjeta, y el importe a abonar.

Micros

Para que la gente pueda llegar a la planta, la empresa ACME S.A. tiene varios micros contratados. En cada micro entran n pasajeros sentados y m parados, donde el n y el m son particulares de cada micro (no son todos los micros iguales). Todos los pasajeros suben al inicio del recorrido y sólo se puede bajar al finalizar el mismo.

La gente no es toda igual, entonces para subirse a un micro se fija en distintas cosas:

- los apurados se suben siempre
- los claustrofóbicos se suben sólo si el micro tiene más de 120 m 3 de volumen (se conoce el volumen de cada micro)
- los fiacas se suben sólo si entran sentados
- los moderados se suben sólo si quedan al menos x lugares libres (no importa si sentados o parados), donde el x es particular de cada persona moderada.
- los obsecuentes toman la misma decisión que tomaría su jefe (de cada empleado se sabe quién es su jefe, que es otro empleado).

Modelar a los micros y las personas de forma tal de:

- poder preguntarle a un micro si se puede subir a una persona, para lo cual tienen que darse dos condiciones: que haya lugar en el micro, y que la persona acepte ir en el micro.
- hacer que se suba una persona a un micro, si no puede, que comunique el error de manera adecuada
- hacer que se baje una persona de un micro, si no se puede (porque está vacío), que comunique el error de manera adecuada
- poder preguntarle a un micro quién fue el primero que se subió

Unidad 2 – Introducción a Java

Hola mundo

Crear un programa en Java que muestre en pantalla "`Hola mundo`"

Múltiples constructores

Crear una clase `Persona` con los atributos nombre y edad. La clase persona debe tener 2 constructores: uno que reciba el nombre y la edad y otro que reciba el nombre y la fecha de nacimiento y que calcule la edad.

Jerarquía de constructores

Crear la clase abstracta `Animal` con un atributo privado `nombre` de tipo `String`. La clase debe tener un único constructor que reciba por parámetro el nombre del animal.

Crear la clase `Perro` que herede de `Animal`. La clase perro debe tener un atributo `raza` y debe tener solo un constructor que reciba como parámetro el nombre y la raza del animal.

Reutilización de constructores

Crear la clase `Perro` con el atributo `raza`. La clase debe tener 2 constructores. El primer constructor debe recibir como parámetro la raza del perro y modificarla para que la primera letra de cada palabra comience con mayúscula.

Crear un constructor que no reciba parámetros que invoque al constructor anterior con el parámetro `callejero`.

Sobreescritura de métodos

Exixten distintos tipos de formas: círculos, cuadrados, rectángulos y triángulos. Crear la clase `Forma` con las subclases correspondientes, e implementar la sobreescritura del método `double calcularArea()`.

Sobreescritura de toString()

Crear la clase `Persona` con los atributos `nombre` y `apellido`. Sobreescribir el método `toString()` heredado de `Object`, para que devuelva el apellido y nombre, separado por coma.

Unidad 3 – Objetos y clases en Java

Piopio

Codificar a Piopio, con estos patrones de modificación de energía:

- cuando vuela, consume un joule por cada kilómetro que vuela, más 10 joules de "costo fijo" en cada vuelo.
- cuando come, adquiere 4 joules por cada gramo que come.

Piopio mensajero

Piopio aprendió a volar sobre la ruta 2, pudiendo hacer paradas en:

- Buenos Aires, km 0
- Chascomús, km 122
- Lezama, km 156
- Dolores, km 210
- Las Armas, km 300
- Mar del Plata, km 400

Codificar estos lugares y agregar lo que haga falta para que:

- Piopio sepa dónde está (vale indicarle un lugar inicial al inicialazarlo).
- le pueda decir a Piopio que vaya a un lugar: eso cambia el lugar y lo hace volar la distancia correspondiente.
- pueda preguntar si Piopio puede o no ir a un lugar: puede ir si le da la energía para hacer la distancia entre donde está y donde le piden ir.

El coyote y el correcaminos

Codificar los objetos coyote y correcaminos e implementar los métodos correspondientes para poder hacer lo siguiente:

- conocer la velocidad del coyote, que es 5 + (energia / 10).
- conocer la velocidad del correcaminos, que es 10 - su peso.
- saber si el coyote puede atrapar al correcaminos; para poder atraparlo, el coyote tiene que ser más veloz que el correcaminos.
- hacer que el coyote corra al correcaminos. Cuando el coyote corre al correcaminos pierde energía en base al tiempo que tarda en alcanzarlo. Se debe restar 0.5 * su velocidad * distancia entre ambos. Además su posición actual pasa a ser igual a la del correcaminos.
- hacer que el coyote se coma al correcaminos. Para que el coyote coma al correcaminos primero debe correrlo. Al comer su energía aumenta en 12 + el peso la presa.

La velocidad del coyote es constante durante la corrida, no se cansa en el medio.

Las posiciones pueden considerarse como números enteros, asumiendo que sólo corren en línea recta.

Frodo, el lobo bobo

Frodo quiere un sistema para simular sus actividades diarias y poder saber si va a dejar de estar gordo (o "rellenito" como piensa él cada vez que se mira al espejo). Se desea:

- conocer la cantidad actual de grasa de Frodo.
- que el lobo se pueda comer a Aurelio Roja (el hermano de Caperucita), que aporta tanta grasa como su peso dividido 10.
- saber si el lobo está gordo (o sea, si tiene más de 200 gramos de grasa).
- saber si el lobo está saludable (o sea, tiene entre 20 y 150 gramos de grasa).
- hacer que el lobo corra un determinado tiempo (quema 2 gramos de grasa por minuto).

La dieta de Piopio

Piopio se cansó de comer siempre alpiste, por lo que ha decidido expandir su dieta. Ahora puede comer también al pez Nemo o un postrecito light.

- cuando come alpiste, la fórmula de variación de la energía es igual a la usada anteriormente (4 joules * cantidad de gramos). La cantidad de joules que aporta Nemo es la raíz cuadrada de su edad, mientras que la energía que aporta un postrecito light es 0.
- saber cuándo Piopio tiene hambre. Piopio tiene hambre si no está empachado ni satisfecho. (está empachado si su energía es mayor a 100, y está satisfecho si su energía es un número entre 50 y 100).

El lobo Frodo y Caperucita Roja

No satisfecho con Aurelio, el lobo se encuentra con Caperucita Roja y quiere comérsela. Además, el lobo quiere comerse a la abuelita. Caperucita aporta sus propias calorías más lo que aporte la canastita, en la que tiene una cierta cantidad de manzanas (cada manzana tiene 2 calorías). La abuela no le aporta nada. Cada caloría equivale a un gramo de grasa. Se pide:

- hacer que el lobo vaya corriendo hasta un lugar, asumiendo que el tiempo que demora en hacerlo depende únicamente del lugar a dónde vaya.
- representar la historia de Frodo y Caperucita, de acuerdo a la siguiente versión:

  El lobo Frodo va corriendo hasta el bosque que queda a diez minutos de su cueva y se encuentra con Caperucita, que al defenderse le tira una manzana. Luego, Frodo corre a la casa de la abuelita (a cinco minutos de allí) y la devora. Apenas llega la nena y le comienza a hacer preguntas molestas, el lobo también se come a Caperucita Roja. Del cazador, ni noticias. ¿Queda en un estado saludable el lobo? ¿Está gordo?

El lobo Frodo y los tres chanchitos

En su raid de destrucción:

- hacer que el lobo sople la casa de uno de los tres chanchitos. Al hacerlo, pierde tanta grasa como la resistencia de la casa más el peso de los ocupantes. La casa de paja no resiste nada, la de madera tiene resistencia 5 y la de ladrillos resiste 2 por cada ladrillo.
- hacer que el lobo vaya corriendo hasta un lugar, asumiendo que el tiempo que demora en hacerlo depende únicamente del lugar donde vaya.

- representar la historia de los 3 chanchitos:

  El lobo sopla primero la casa de paja, la hace caer y el chanchito huye a la casa de madera donde estaba el otro chanchito. El lobo lo persigue hasta la casa de madera, la sopla y también la tumba, por lo que ambos chanchitos se van a la casa de ladrillos del tercer chanchito. El lobo los alcanza e intenta tumbarla ¿lo logra?

Sueldo de Pepe

Implementar los objetos necesarios para calcular el sueldo de Pepe. Éste se calcula de la siguiente manera: sueldo = neto + bono x presentismo + bono x resultados.

El neto depende de la categoría; hay varias categorías. Por ejemplo, los gerentes ganan $1000 de neto, y cadetes ganan $1500, aunque puede haber más.

Hay dos bonos por presentismo:

- uno es $100 pesos si la persona a quien se aplica no faltó nunca, $50 si faltó un día, $0 en cualquier otro caso.
- otro siempre es $0, independientemente de las faltas.

Hay tres posibilidades para el bono por resultados:

- 10% sobre el neto.
- $80 fijos.
- o nada.

Probar cambiándole a Pepe la categoría, la cantidad de días que falta y sus bonos por presentismo y por resultados; y preguntarle en cada caso su sueldo.

Unidad 4: Excepciones

Lanzar excepción

Crear la clase `Calculadora` que tenga el método `double dividir(double dividendo, double divisor)`. En caso de que el divisor sea cero, lanzar `java.lang.Exception`, con el mensaje "`No se puede dividir por cero`"

Excepciones propias

Crear la clase `ParseadorDeFecha`, que reciba una fecha en el formato DD/MM/YYYY y que lance la excepción `FormatoInvalidoDeFecha` si la fecha no es válida. Para parsear la fecha, usar el método split de la clase `String`.

Lanzar runtime exceptions

Crear la clase `Circulo` con el atributo `radio`. La clase debe tener 2 constructores, uno vacío y otro que reciba como parámetro el radio. Además debe tener un método `void setRadio(double radio)` para cambiar el radio del círculo. Tanto en el contructor como en el método, en caso de recibir un radio menor o igual a cero, lanzar la excepción `IllegalArgumentException`.

Además crear un método `double calcularArea()`. En caso de que el radio sea 0, lanzar la excepción `IllegalStateException`.

Manejo de fechas

Para este ejercicio, primero crear la clase `Fecha` con tres atributos enteros: `dìa`, `mes` y `año`. A continuación, crear la clase `ManejadorDeFechas` que tenga 3 métodos:

- `String fomatearFecha(Fecha unaFecha)`: que dada una fecha devuelva la fecha en formato String dia(2)/mes(2)/año(4). Ej: 22/12/2012
- `Fecha parsearFecha(String unaFecha) throws Exception`: dada una fecha en el formato anterior, que la convierta a Fecha
- `int cantidadDeAnios(Fecha unaFecha)`: calcular la cantidad de años transcurridos desde el día de hoy hasta la fecha del parámetro

Todavía sirve

Crear las clases `Alimento`, que representará a un alimento en particular, y `VerificadorVencimiento`, que será la encargada de verificar si un alimento se encuentra vencido o no. De cada alimento se conoce su `nombre` y cuántos `días` restan hasta su fecha de vencimiento. Implementar en la clase `VerificadorVencimiento` el método `public void todaviaSirve(Alimento unAlimento) throws AlimentoEnMalEstadoException`, que dado un alimento, lanzará la excepción `AlimentoEnMalEstadoException` en caso de detectar que el alimento se encuentra vencido. Un alimento se considera vencido si la cantidad de días para su vencimiento es menor o igual a cero.

Unidad 5 – Manejo de colecciones

Manejo de listas

Crear la clase `Bebida` con el atributo `marca`. Sobreescribir los métodos `equals()` y `hashcode()` para que se correspondan con este atributo. Crear una lista e iterar sobre los elementos de la misma.

Crear una nueva instancia de `Bebida` que tenga la misma marca que una del listado y pasarla como parámetro al método `remove(Objecto o)` de la lista. Volver a recorrer la lista para verificar que ya no esté el objeto.

Manejo de Set

Repetir el ejercicio anterior pero en lugar de utilizar una lista, utilizar un `Set` con la implementación `HashSet`.

Luego crear una instancia de `Bebida` con la misma marca que alguna de las que está dentro del set y agregarla al listado. Verificar que el set no haya agregado el elemento.

Manejo de SortedSet

Crear la clase `Bebida` con el atributo `marca`. Sobreescribir los métodos `equals()` y `hashcode()` para que se correspondan con este atributo. Implementar la interfaz `Comparable` para que las bebidas tengan como orden natural el orden alfabético de la marca.

Crear un `SortedSet` en base a un `TreeSet` y agregar 4 bebidas. Mostrar la colección y verificar que se vean en orden alfabético.

Crear otro `TreeSet` pero esta vez pasar como argumento una instancia de `OrdenadorBebidas`. Esta clase debe implementar `Comparator<Bebida>` y ordenar las bebidas por el orden alfabético inverso. Agregar 4 bebidas a este nuevo sortedSet y verificar que se muestren en el orden inverso.

Pizzería

Una pizzería quiere tener un sistema para registrar todos los pedidos a domicilio que realiza. Los productos que vende son empanadas, pizzas y promociones, que son una suma de los otros productos a un precio total de 25% menos. Por ejemplo una promoción es una pizza grande de muzarrella, 3 empanadas de carne y 3 empanadas de pollo.

Se desea que el sistema, pueda:

- registrar un pedido (que se pidió y a donde hay que llevarlo), así como que se calcule su precio automáticamente.
- decir cual es el producto más vendido y su sabor. Tener en cuenta que las promociones como tales no se cuentan, sino que se toma en cuenta las pizzas y empanadas que efectivamente se venden. Por ejemplo: empanada de carne.
- calcular el total vendido.

Ordenamiento de listados

Crear la clase `Persona` con los atributos `nombre`, `apellido`, `edad` y `DNI`. Agregar 5 elementos a un List. Reordenar el list a través de `java.util.Collections.sort(List l, Comparator c)` según los siguientes criterios:

- por apellido y nombre en orden alfabético
- por nombre y apellido en orden alfabético inverso
- por edad de menor a mayor
- por DNI de mayor a menor

En todos los casos, luego de ordenar el listado, imprimirlo en pantalla para verificar el orden correcto.

Depurar colecciones

Crear un método que dado una colección de objetos `Persona`, elimine de dicha colección todas las personas menores de 21 años de edad.

Estadísticas de alumnos

A final de año, en un sistema de un colegio secundario están registrados los alumnos de cada curso, las fechas de las distintas ausencias y las notas de cada materia en cada uno de los 3 trimestres de este año. En cada falta se registra si está justificada o no.

Se desea crear un programa que muestre genere un `Map` de los cursos y la cantidad de alumnos que no van a repetir de año.

Si un alumno se queda libre, repite de grado. Esto se da cuando:

- tiene más de 15 faltas no justificadas
- tiene más de 25 faltas totales

Si un alumno desaprueba más de 2 materias, puede repetir (ya que le pueden quedar más de 2 materias libres). Un alumno no aprueba la materia cuando:

- tiene menos de 6 en el 3er trimestre
- tiene un promedio menor a 6 entre los 3 trimestres

Unidad 6 – Entrada y salida

Cuenta letras

Crear un programa que lea un archivo de texto y muestre en pantalla cuantas veces se repite cada letra (de la A-Z) en el archivo.

Separando por comas

Dada la clase `Empleado` que tiene `legajo`, `nombre`, `apellido` y `telefono`, crear un programa que guarde los datos de 3 empleados en un archivo de texto CSV. Crear otro programa que abra el archivo y muestre los datos en pantalla.

Repitiendo en binario

Realizar el ejercicio del punto anterior, utilizando archivos binarios mediante `DataInputStream` y `DataOutputStream`.

Figuras de archivo

Dada la clase `Figura` y sus clases hijas `Circulo`, `Rectangulo` y `Triangulo`, implementar la clase `FigurasDeArchivo` con el método `Figura crearFiguraDeArchivo() throws NoHayMasFigurasException`, que devuelve una figura leída del archivo. Existe un archivo para cada tipo de figura. El archivo debe ser binario y ser leído mediante `ObjectInputStream`.

Figuras textuales

Implementar el ejercicio del punto anterior, leyendo cada figura desde un archivo de texto CSV. Respetar la definición del método `Figura crearFiguraDeArchivo() throws NoHayMasFigurasException`, que por cada línea leída creará y devolverá una `Figura`. Utilizar el método split de la clase `String` para obtener cada "token" de la línea leída.

Unidad 7 – Concurrencia

Lanzar "Hola mundo Thread"

Crear:

- una clase que herede de `Thread` que escriba en pantalla "`Hola mundo Thread`"
- una clase que implemente `Runnable` que escriba en pantalla "`Hola mundo Runnable`"

El main, una vez que haya ambos threads, debe escribir en pantalla "`Fin de main`". Verificar que en múltiples ejecuciones del main, se escriban los mensajes en diferente orden.

Yield

Repetir el ejercicio anterior pero invocando al metodo yield() en el thread y en el runnable antes de escribir en pantalla. Verificar que la mayoría de las veces se escribe "`Fin de main`".

Join

Lanzar un thread que escriba el mensaje "`Thread 1`" 5 veces. Utilizar el método `Thread.sleep(1000)` para la pausa. Lanzar simultaneamente otro thread similar que escriba "`Thread 2`". Hacer que el main escriba "`Fin del main`" luego de que ambos threads hayan terminado (utilizar el método j`oin()`).

Interrupciones

Lanzar un thread que escriba el valor de un contador (y lo incremente). Hacer que el main interrumpa al thread luego de 5 segundos y escriba el mensaje "`Fin del main`" luego de que el thread haya terminado. Verificar que el thread no siga escribiendo en pantalla.

CSVs y Threads

Lanzar un thread que dé de alta 20 alumnos en un archivo CSV (separado por comas) de alumnos, con el siguiente formato: NOMBRE,APELLIDO,DNI. Hacer una clase `Ordenador`, que tome por parámetro la lista de alumnos a ordenar y otro que indique el tipo de ordenamiento (orden por nombre, por dni, por apellido) y en base a ese parámetro decida con que `Comparator` debe ordenar. Lanzar otro thread que lea todo el archivo, ordene los alumnos por NOMBRE, y los escriba en pantalla. Dos minutos después, deberá leer de vuelta el archivo y ordenar por APELLIDO, y en otros dos minutos, ordenar por DNI.

Carrera de caballos

Se desea crear un simulador en tiempo real de una carrera de caballos. Para esto es necesario crear un programa java que simule una carrera. Al iniciar la carrera deben comenzar a "correr" 5 caballos en simultáneo. El primero que llega a los 1000 metros es el ganador.

Al finalizar la carrera el programa tiene que mostrar 2 listados:

- el listado de caballos ordenado por nombre y mostrando su posición de llegada
- el listado de caballos ordenado por orden de llegada a la meta

Para simular la velocidad del caballo, hay que hacer que el caballo avance a intervalos regulares (sugerido 50ms) una cantidad de metros aleatoria entre su velocidad mínima y máxima (expresada en metros) dada en el siguiente cuadro:

Caballo	*Velocidad mínima*	*Velocidad máxima*
Centella	11	14
Tornado	6	24
Suspiro	12	13
Relámpago	10	21
Pegaso	7	22

Sincronización - Sumar y restar

Crear la clase `Contador` con los métodos `void incrementar()` que incrementa en uno al contador, `void decrementar()` que lo decrementa en uno y `int dameEstadoActual()` quede devuelva su estado actual. Lanzar concurrentemente 6 threads, 3 que incrementen 1000 veces a un contador y otros 3 que lo decrementen 1000 veces al mismo contador. Tienen que hacerlo de la forma más concurrente posible. Luego de que finalicen los 6 threads, el main debe mostrar el estado actual del contador. Verificar que el resultado sea 0.

Unidad 8: Patrones de diseño

Ejemplos de patrones de diseño

Describa un caso práctico donde aplicaría cada uno de los siguientes patrones de diseño:

- Singleton
- Factory Method
- State
- Observer
- Strategy
- Composite
- Decorator

Singleton Pi

Implemente la clase `Pi` que sea un singleton y que sea la encargada de brindar el valor de PI en el sistema.

Factory de figuras random

Dada la clase `Figura` y sus clases hijas `Circulo`, `Rectangulo` y `Triangulo`, implementar los *factories* concretos de la clase `FigurasRandomFactory` con el método abstracto `Figura crearFiguraRandom()`, que crea una figura con sus atributos generados aleatoriamente.

Para generar números aleatorios utilizar `java.util.Random`

Composite - Tamaño de archivos

Utilizar el patron composite para las clases `Carpeta` y `Archivo` con el fin de resolver el método `double tamañoTotal()` que informa cuál es el tamaño de un archivo o de una carpeta y sus subcarpetas y archivos.

Composite - Empleados a cargo

En una empresa existe la siguiente jerarquía de puestos: gerente general, gerente de área, jefe de sección, supervisor y empleado raso (quien no tiene gente a cargo). Utilizar el patrón composite para poder preguntarle a cualquiera de los distintos puestos de la jerarquía, a cuantas personas tiene a su cargo.

Decorator - Cafetería

En una cafetería se vende café, al cual puede agregarse distintos ingredientes a gusto del consumidor. Actualmente, se cuenta con leche, crema, canela, chocolate y leche condensada como posibles agregados. Tanto del café como de cada ingrediente se conoce su nombre y su precio. Aplicando el patrón decorator, implementar una solución que permita averiguar el costo de un café, teniendo en cuenta que se pueden agregar varios ingredientes a un mismo café, incluso repetirlos.

State - Cuenta bancaria

En un sistema bancario, la cuenta corriente permite las operaciones de depósito y extracción. Estas operaciones pueden estar permitidas o no según estado de la cuenta. Los posibles estados son:

- cuenta normal: este estado se da para cualquier cuenta con saldo mayor o igual a cero. Permite todas las operaciones.
- cuenta en rojo: se da cuando la cuenta tiene un saldo menor a cero. En este caso se permiten los depósitos, pero no se permiten extracciones
- cuenta cerrada: esta se da cuando el cliente avisa que quiere cerrar la cuenta. No permite depósitos y sólo permite extracciones que dejen a la cuenta en saldo mayor o igual a cero.

State - Semáforo

Crear la clase `Semaforo` con el método `String cambiarColor()`. Implementar el patrón state para resolver el pasaje de color a color del semáforo de manera que no se utilice ningún “if”.

Strategy - Cuenta del restaurante

En un restaurante, cuando le generan la cuenta para un cliente, se tiene que sumarizar el total de lo consumido. Pero según el día de la semana, la forma de cálculo es distinta:

- miercoles: 20% de descuento sobre el total
- domingos: 10% de descuento si paga en efectivo
- otros días: importe normal

Aplicar el patrón Strategy para implementar los distintos tipos de cálculo de total.