Ejercicio 1

Añadir las siguientes funcionalidades al metamodelo del Ejercicio 1 de la Práctica 1A (Fábrica de Caramelos):

1. Implementar una restricción que avise cuando las cintas transportadoras superen su capacidad
2. Implementar restricciones que limiten el tipo de componentes que tienen las cintas transportadoras, dependiendo de la máquina que tienen a su salida y/o entrada, de acuerdo con los siguientes criterios:
    - A la salida de las máquinas CandyProducer debe haber únicamente componentes CandyBall
    - A la salida de las máquinas StickProducer debe haber únicamente componentes Stick
    - A la salida de las máquinas Flattener debe haber únicamente componentes CandyFlat
    - A la salida de máquinas Assembler debe haber únicamente componentes LolliPop
    - A la salida de máquinas Wrapper debe haber únicamente componentes LolliPop
3. Crear una propiedad derivada en la clase Factory que devuelva el número total de componentes producidos
4. Crear una propiedad derivada en la clase Factory que cuente el número total de componentes existentes en este momento en la fábrica

El metamodelo enriquecido deberá almacenarse en un fichero llamado **candy_factory_v2.ecore**

Para poner a prueba las restricciones impuestas, se pide además crear las siguientes 3 nuevas instancias del metamodelo:

- Un modelo de la fábrica con componentes en las cintas transportadoras, y que cumpla con todas las restricciones (**candy_factory_inst3.xmi).**
- Un modelo de la fábrica con componentes en las cintas transportadoras, en el que alguna cinta transportadora tenga más componentes de los permitidos (**candy_factory_inst4.xmi).**
- Un modelo de la fábrica con componentes en las cintas transportadoras, en el que alguna cinta transportadora no tenga un componente correcto (**candy_factory_inst5.xmi).**

NOTA: para crear las instancias de la fábrica, se puede partir de alguno de los modelos creados en el Ejercicio 1 de la Práctica 1A.

Ejercicio 2

Añadir las siguientes funcionalidades al metamodelo del Ejercicio 2 de la Práctica 1A (Programación de Vuelos):

1. Añadir las siguientes restricciones:
    - Nunca debe programarse un avión que no esté disponible para volar
    - La hora programada de despegue de un vuelo debe ser anterior a la hora programada de aterrizaje
    - Debe haber al menos 1 hora entre un aterrizaje y el siguiente despegue del mismo avión.
    - Ningún avión puede volar más de 16 horas en un mismo día.
    - Todos los vuelos deben volar hacia o desde una base
    - El origen y el destino de cualquier vuelo debe ser distinto
2. Implementar las siguientes operaciones:
    - Una operación de avión que retorne el número de horas que va a volar.
    - Una operación que retorne una lista de aviones ordenada según el número de horas que va a volar cada uno, en orden descendente.
    - Una operación de avión que retorne la lista de sus vuelos programados, ordenada por hora de despegue.
3. Crear un modelo de planificador de vuelos con al menos 3 aviones, 5 destinos, y 10 vuelos. El modelo debe cumplir todas las restricciones.

Operaciones OCL relevantes para esta práctica

Para una documentación más completa sobre las operaciones OCL disponibles, se puede consultar este link: https://wiki.eclipse.org/Acceleo/OCL_Operations_Reference

`Sequence{a..b}:`

Crea una colección ordenada (llamada Sequence en OCL) de valores numéricos entre a y b

Ejemplo: `Sequence{1..5} // Crea la secuencia {1,2,3,4,5}`

`forAll (expr : OclExpression) : Boolean`

Aplica la expresión en cada uno de los elementos de una colección. Retorna true si la expresión devolvió true en todas las evaluaciones

Ejemplo: `Sequence{1..5} -> forAll(self>0) // retorna true, porque todos los valores de la secuencia son mayores de 0`

`at (index : Integer) : T`

Devuelve el elemento en la posición index

Ejemplo: `Sequence{3..6} -> at(1) // Devuelve 3. En OCL el primer elemento tiene índice 1`

`sortedBy (expr : OclExpression) : Sequence(T)`

Retorna una nueva secuencia. Esta secuencia tendrá los valores de la colección de entrada, ordenados según el criterio especificado por la expresión expr.

Ejemplo: Supongamos que tenemos una clase Persona con un atributo "edad" de tipo entero. Supongamos que estamos aplicando el código OCL sobre un objeto que tiene una propiedad "personas", que es una lista de objetos Persona

```
self.personas -> sortedBy(edad) // retorna una secuencia de personas, ordenadas por sus edades (de menor a mayor)
```

`select (expr : OclExpression) : Collection(T)`

Retorna una colección que contiene los valores de la colección de entrada que cumplen con la condición expr

Ejemplo: `Sequence{1..5} -> select(i | i > 3) // devuelve {4,5}`

La expresión `i | i > 3` se puede entender como:

- La expresión define un parámetro de entrada, que llama `i`
- El cuerpo de la expresión es `i>3`.
- La barra vertical `|` se utiliza para separar el parámetro de entrada y el cuerpo

collect (expr : OclExpression) : Collection(T2)

Retorna una nueva colección que resulta de aplicar una expresión expr sobre cada uno de los elementos de la colección de entrada. La expresión expr debe tener un parámetro de entrada, y retornar un valor

Ejemplo: `Sequence{1..5} -> collect(i | i*2) // devuelve {2,4,6,8,10}`

oclIsTypeOf(typespec : Classifier) : Boolean

Retorna true si el objeto sobre el que se aplica la operación es exactamente del tipo typespec

Ejemplo: Supongamos que tenemos una propiedad llamada persona de tipo Persona

```
persona.oclIsTypeOf(Persona) // devuelve true
```

```
persona.oclIsTypeOf(Animal) // devuelve false, porque persona no es del tipo Animal
```

oclIsKindOf(typespec : Classifier) : Boolean

Retorna true si el objeto sobre el que se aplica la operación es compatible con el tipo typespec

Ejemplo: Supongamos que tenemos una clase Persona, y otra clase Empleado que extiende Persona

```
empleado.oclIsKindOf(Persona) // devuelve true
```

```
empleado.oclIsKindOf(Empleado) // devuelve true
```

```
empleado.oclIsTypeOf(Persona) // devuelve false, un empleado no es exactamente de tipo Persona
```

isUnique (expr : OclExpression) : Boolean

Retorna true si todos los elementos de la colección evalúan a un valor distinto con la expresión expr

Ejemplo:

```
Sequence{2,3,4,5} -> isUnique(i | i>1)  // devuelve false, porque todas las expresiones devuelven lo mismo (true)
```

```
Sequence{2,3,4,5} -> isUnique(i | i)    // devueve true, porque cada expresión devuelve un valor distinto (el propio valor en la secuencia)
```

excludes (object : T) : Boolean

Retorna true si el objeto no aparece en la lista

Ejemplo:

```
Sequence{2,3,4,5} -> excludes(6)    // devueve true, “2” no aparece en la lista
```