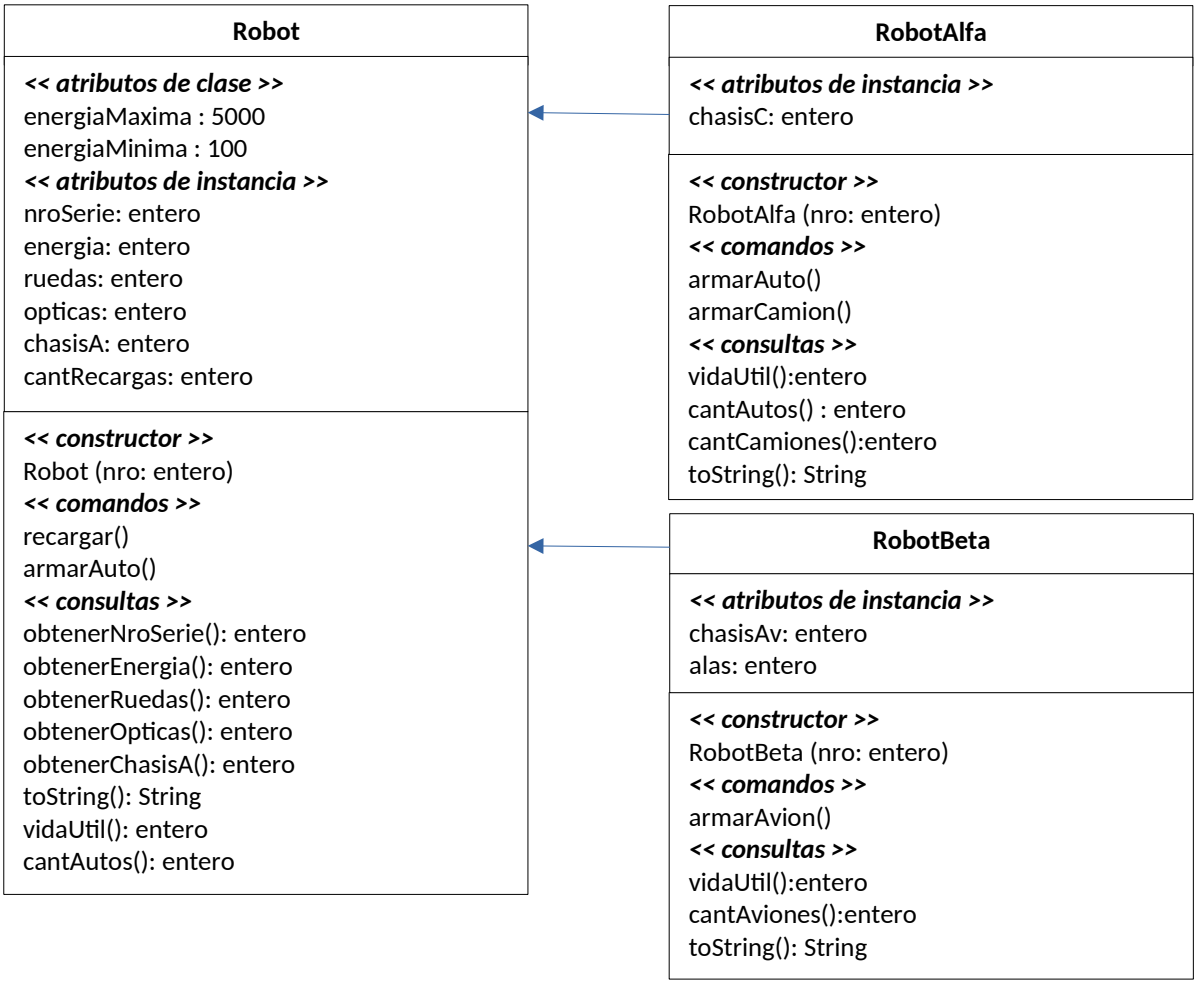
Introducción a la Programación Orientada a Objetos

DCIC - UNS
2019

LABORATORIO NRO 09
FÁBRICA DE JUGUETES

En el presente laboratorio trabajaremos sobre el modelo de las clases **Robot** totalmente implementada en Java y **RobotAlfa**, parcialmene implementadas en Java, y **RobotBeta**, capaz de construir aviones.

Dado la siguiente definición de clases:



Teniendo en cuenta que:

- Cada robot construye un auto usando 1 chasis, 4 ópticas y 4 ruedas. Cada robot recarga su energía hasta el máximo cuando queda por debajo del mínimo. La vida útil es 1000 menos la cantidad de recargas realizadas.
- Los robots del modelo Alfa arman un auto usando 1 óptica y una rueda adicional (que colocan en el interior) y arman también camiones, con 6 ruedas, 6 ópticas y 1 chasis de camión. La vida útil es 5000 menos la cantidad de recargas realizadas.
- Los robots del modelo Beta pueden armar autos y aviones.
- Para armar un avión utilizan 3 ruedas, 2 alas, y 1 chasis de avión.
- El desgaste de esta última clase de robots es mucho mayor que los otros modelos, por lo tanto, la vida útil se calcula como 1000 menos dos veces la cantidad de recargas.

Introducción a la Programación Orientada a Objetos

DCIC - UNS
2019

Los robots trabajan en una fábrica, que se modela con una *colección* de las siguientes características:

- No es importante la ubicación en la que trabaja cada robot. Sin embargo, se puede consultar qué robot ocupa una determinada posición. En ese caso, es el cliente el responsable de validar que la posición sea válida (*0 <= pos < capacidadOcupada()*).
- Cada robot que trabaja en la fábrica aparece una sola vez en la *colección*.
- La *colección* no se mantiene ordenada por ningún criterio más que el orden en que los robots fueron agregados.
- Es significativo saber cuántos robots están trabajando en la fábrica, y si hay lugar disponible para nuevos robots.
- La *colección* se implementa mediante un arreglo, con las componentes ocupadas al principio, y las no ocupadas al final con valor null.
- Al asignar un Robot a la Fábrica, si hay lugar, se agrega a continuación del último Robot. Es responsabilidad del cliente verificar que haya lugar antes de asignar.
- Al desasignar un Robot, la colección debe mantenerse comprimida.

Dada la siguiente definición de la clase **Fabrica**, que implementa una *colección* de Robots en Java:

Fabrica
<< atributos de instancia >> C [] Robot cant: entero
<< constructor >> Fabrica (max : entero) ***<< comandos >>*** asignar(r: Robot) **desasignar(r: Robot)** ***<< consultas >>*** capacidadTotal(): entero capacidadOcupada(): entero estaRobot(Robot r): boolean robotEnPosicion(pos: int): Robot **existenNRobotsVU(n: int, vu: int): boolean**

Ejercicios:

1. Implementar la clase **RobotBeta** de manera completa y el método armarAuto de la clase **RobotAlfa**.

2. Implementar el comando **desasignar(r: Robot),** que localiza al Robot en la colección, lo elimina y comprime la estructura. Recordar reducir en uno la cantidad de robots en la fábrica. Puede ocurrir que el Robot recibido no pertenezca a la colección, en ese caso no debe hacer nada. Para ser eficiente, debe ser implementado en un solo recorrido.

3. Implementar la consulta **existenNRobotsVU(n: int, vu: int): boolean**, que decide si existen al menos **n** robots con una vida útil mayor a **vu**.

4. Analice como cambia la estrategia y proponga una implementación para **desasignar(r:Robot)** y para **existenNRobotsVU(n: int, vu: int): boolean** esta vez considerando que la Fábrica se modela como una tabla de sectores donde cada sector puede o no tener un robot asignado.

5. Completar el Tester con casos de prueba significativos.

Introducción a la Programación Orientada a Objetos

DCIC - UNS
2019

Ejercicio Adicional

Considere ahora que la clase **FábricaSucursales** modela una empresa de múltiples sucursales, donde cada sucursal tiene la misma cantidad de sectores, y en cada sector de cada sucursal puede haber o no asignado un Robot. Para mantener la información sobre cada robot en cada sector de cada sucursal se utiliza un matriz donde cada fila representa una sucursal, y cada columna representa el sector. Por ejemplo, en la fila 0, columna 2 de la matriz se almacena un **RobotAlfa** que se corresponde con la sucursal 1 sector 3.

	Sector 1	Sector 2	Sector 3	…	Sector k
Sucursal 1	Robot	*No asignado*	Robot Alfa	…	Robot
Sucursal 2	Robot Alfa	Robot Alfa	*No asignado*	…	*No asignado*
…	…	…	…	…	…
Sucursal n	Robot	Robot	Robot	…	Robot alfa

FabricaSucursales
C: [][]Robot
<<Constructores>> FabricaSucursales(cantSuc, cantSec: entero) <<Comandos>> establecerRobot(r:Robot, sucursal, sector: entero) <<Consultas>> obtenerRobot(sector,sucursal:entero):Robot robotsSinVU():FabricaSucursales cantAutos(sucursal:entero):entero cantAutos():entero

Dada la especificación de la clase **FábricaSucursales**:

a) Implemente en java la consulta **robotsSinVU():FabricaSucursales** que retorna una nueva FabricaSucursales con aquellos robots sin vida útil.

b) Proponga un planteo recursivo y una implementación en java, que se corresponda con el planteo, para el método **cantAutos(suc:entero):entero** que determina la cantidad de autos que se pueden armar con todos los robots de la sucursal suc. *Nota: recordar que el cliente ve el número de sucursal desde 1 hasta la cantidad de sucursales, no desde cero.*

c) Utilizando el método previamente desarrollado implemente en java el método **cantAutos():entero** que retorna la cantidad de autos que se pueden construir actualmente usando todos los robots de todas las sucursales de la fábrica.

d) Pruebe su solución utilizando el Tester provisto y amplíe los casos de prueba de ser necesario.