Gregorio Vidoy Fajardo

Ejercicio 1 P1

Tabla de contenidos:

DESCRIPCIÓN DEL PROBLEMA:

Programar utilizando hebras la simulación de 2 carreras simultáneas con el mismo número inicial (N) de bicicletas. N no se conoce hasta que comienza la carrera. De las carreras de montaña y carretera se retirarán el 20% y el 10% de las bicicletas, respectivamente, antes de terminar. Ambas carreras duran exactamente 60 s. y todas las bicicletas se retiran a la misma vez.

Supondremos que Carrera, una clase interfaz de Java, declara el método de fabricación: crearCarrera(), que implementarán una clases factoría para la creación de 2 objetos ArrayList de Java que van a incluir las bicicletas de cada tipo (CARRETERA, MONTANA), que participan en cada una de las dos modalidades de carrera.

Asumimos dos clases factoría: FactoriaCarreraMontaña y FactoriaCarreraCarretera, que implementarán el método de fabricación anteriormente aludido que, a su vez, creará uno de los objetos: ArrayList<Bicicleta>[N] llenándolo de bicicletas de la clase compatible con la modalidad de carrera que nos interese.

La clase Bicicleta se debería definir como una clase abstracta y en su constructor aparecería un parámetro del tipo: public enum TC{

```
    MONTANA, CARRETERA

  }
```

Y el resto de sus métodos podrían ser: TC getTipo(), void setTipo(), redefiniendo además el método toString() para que devuelva la cadena: "bicicleta - " + tipo.

En Java no existe una forma estándar de liberar memoria (como delete() de C++), pero necesitaremos eliminar bicicletas durante la ejecución del programa. Supondremos que el asignar explícitamente referencias a null se puede considerar una técnica legítima de liberar memoria en Java y dejaremos que su recolector de basura haga el trabajo automáticamente.

Para programar la simulación de las 2 carreras simultáneas de bicicletas utilizaremos creación de hebras de Java.

Por supuesto, tendremos que añadir métodos de fabricación adicionales para que las bicicletas sean unos objetos compuestos de manillar, 2 ruedas y marco. Una carrera se compondrá de N bicicletas. Por consiguiente, se han de programar métodos de fabricación específicos para dichos objetos "básicos": cuadroCarretera(), manillarCarretera(), ruedasCarretera(), cuadroMontaña(), manillarMontaña(), ruedasMontaña(). CuadroMontaña, ManillarMontaña, RuedaMontaña son clases específicas que “versionarán” las clases genéricas: Cuadro, Manillar, Ruedas.

RELEASE NOTE:

-Compilado con Java 1.8
-Clase Main Prueba.

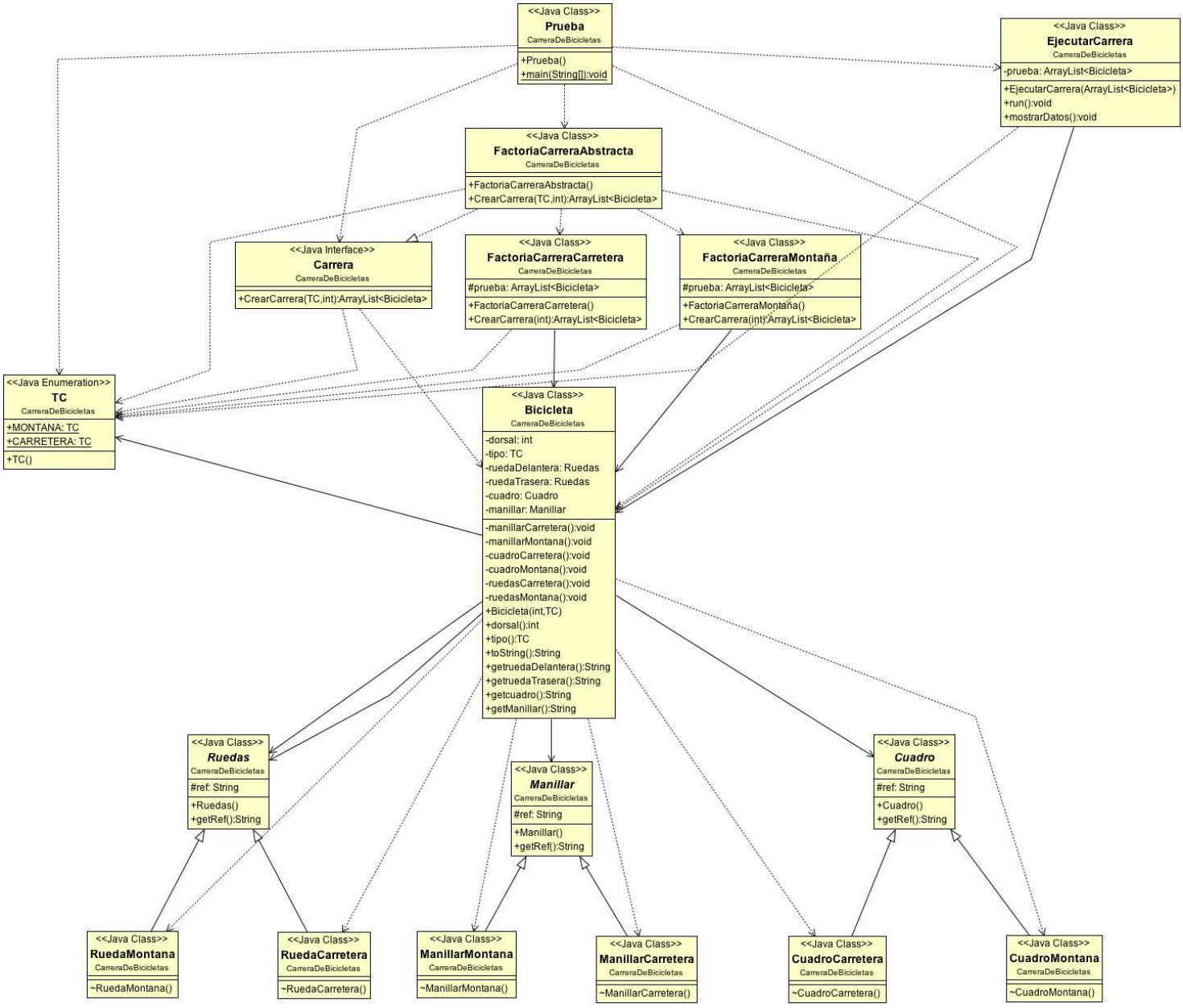
SOLUCIÓN AL PROBLEMA:

-Para la creación de las bicicletas compuestas, uso el patrón de factoría. Creando las clases Cuadro, Manillar, Ruedas, en las que especializa según sea carretera o montaña.
-Para la creación de la parrilla de salida de bicicletas, uso el patrón factoría abstracta, genera un arraylist de bicicletas (La parrilla de salida), según sean de carretera o de montaña.
-Para la realización de la carrera, uso la clase EjecutarCarrera, que hereda de Thread y ejecuta una hebra por carrera mostrando los resultados de la carrera al terminar con un paro del sistema de (60s) descontando la duración de la carrera, exactamente.

EXPLICACIÓN DEL PROGRAMA:

El programa inicia y crea dos parrillas de bicicletas, una de 20 participantes de bicicletas de montaña y otra de 30 participantes de bicicletas de carretera.
Se almacenan en 2 arraylist.
Se le pasa cada arraylist a ejecutar carrera, posteriormente se muestran los datos de la parrilla de carrera.
Después se inician las hebras, una para cada carrera, y a continuación se para el sistema para calcular los 60 s de parada, posteriormente se muestran los resultados por pantalla, las bicicletas se eliminan aleatoriamente en función del tipo de bicicleta.

UML:


<<Java Class>>
Prueba
CarreraDeBicicletas
+Prueba()
+main(String[]):void
<<Java Class>>
EjecutarCarrera
CarreraDeBicicletas
-prueba: ArrayList<Bicicleta>
+EjecutarCarrera(ArrayList<Bicicleta>)
+run():void
+mostrarDatos():void
<<Java Class>>
FactoriaCarreraAbstracta
CarreraDeBicicletas
+FactoriaCarreraAbstracta()
+CrearCarrera(TC,int):ArrayList<Bicicleta>
<<Java Interface>>
Carrera
CarreraDeBicicletas
+CrearCarrera(TC,int):ArrayList<Bicicleta>
<<Java Class>>
FactoriaCarreraCarretera
CarreraDeBicicletas
#prueba: ArrayList<Bicicleta>
+FactoriaCarreraCarretera()
+CrearCarrera(int):ArrayList<Bicicleta>
<<Java Class>>
FactoriaCarreraMontaña
CarreraDeBicicletas
#prueba: ArrayList<Bicicleta>
+FactoriaCarreraMontaña()
+CrearCarrera(int):ArrayList<Bicicleta>
<<Java Enumeration>>
TC
CarreraDeBicicletas
+MONTANA: TC
+CARRETERA: TC
+TC()
<<Java Class>>
Bicicleta
CarreraDeBicicletas
-dorsal: int
-tipo: TC
-ruedaDelantera: Ruedas
-ruedaTrasera: Ruedas
-cuadro: Cuadro
-manillar: Manillar
-manillarCarretera():void
-manillarMontana():void
-cuadroCarretera():void
-cuadroMontana():void
-ruedasCarretera():void
-ruedasMontana():void
+Bicicleta(int,TC)
+dorsal():int
+tipo():TC
+toString():String
+getruedaDelantera():String
+getruedaTrasera():String
+getcuadro():String
+getManillar():String
<<Java Class>>
Ruedas
CarreraDeBicicletas
#ref: String
+Ruedas()
+getRef():String
<<Java Class>>
Manillar
CarreraDeBicicletas
#ref: String
+Manillar()
+getRef():String
<<Java Class>>
Cuadro
CarreraDeBicicletas
#ref: String
+Cuadro()
+getRef():String
<<Java Class>>
RuedaMontana
CarreraDeBicicletas
~RuedaMontana()
<<Java Class>>
RuedaCarretera
CarreraDeBicicletas
~RuedaCarretera()
<<Java Class>>
ManillarMontana
CarreraDeBicicletas
~ManillarMontana()
<<Java Class>>
ManillarCarretera
CarreraDeBicicletas
~ManillarCarretera()
<<Java Class>>
CuadroCarretera
CarreraDeBicicletas
~CuadroCarretera()
<<Java Class>>
CuadroMontana
CarreraDeBicicletas
~CuadroMontana()