TECNOLOGÍA DE PROGRAMACIÓN

Guión de la práctica 4

1.- Objetivo de la práctica

- Aprender a combinar la herencia y la programación genérica.
- Maximizar el polimorfismo y la reutilización de código, eligiendo el mecanismo óptimo de orientación a objetos.

Tarea:

Elige un lenguaje orientado a objetos (C++ o Java). Realizarás toda la práctica en dicho lenguaje.

2.- Productos, contenedores y camiones

Una empresa de transportes se encarga de transportar una serie de **productos** entre diferentes ciudades el mundo. Cada producto ocupa un **volumen** determinado (en metros cúbicos, un número real), y es fundamental conocerlo para poder transportarlo.

A su vez, la empresa dispone de contenedores. Cada **contenedor** también ocupa un volumen determinado, que coincide con su capacidad. Un contenedor puede contener tantos productos como quepan en dicho contenedor, es decir, el volumen total de todos los productos de dentro de dicho contenedor tiene que ser igual o menor que su **capacidad** (que coincide con el volumen que ocupa).

Además, la empresa dispone de una flota de **camiones** de diferentes capacidades, en los que de nuevo sólo caben productos que en total ocupen un volumen total menor a su capacidad. Dentro de un camión también se puede introducir un contenedor (siempre y cuando quepa en términos de volumen). De hecho, dentro de un contenedor también se puede introducir otro contenedor, siempre y cuando su volumen lo permita. Un camión, sin embargo, no es un tipo de carga, y por tanto no puede guardarse dentro de un contenedor ni dentro de otro camión.

Tarea:

Diseña (en en el lenguaje elegido) las clases necesarias para representar todos los elementos necesarios requeridos, incluyendo productos, contenedores y camiones. Los nombres de las clases (o interfaces) implicadas serán obligatoriamente los siguientes :

- **`Camion`** representará un camión con su correspondiente capacidad, información que se le pasará en su constructor.
- **`Producto`** representará un producto con su correspondiente volumen y un nombre identificativo (una cadena de texto). El constructor será **`Producto(double volumen, String nombre)`** en Java y **`Producto(double volumen, const std::string& nombre)`** en C++.
- **`Contenedor`** representará un contenedor, también con su correspondiente volumen (que equivale a su capacidad). Dicho valor se le pasará también en el constructor.

Tanto la clase que representa a un contenedor como la que representa a un camión deberán tener el siguiente método:

- **`boolean guardar(? elemento)`**, que guardará el elemento que se le pasa por parámetro (sea un producto o un contenedor) si cabe según su capacidad. Si el elemento es guardado, el método devolverá **`true`**, mientras que si no hay capacidad libre suficiente, el elemento no se guardara y el método devolverá **`false`**. Además de esto, el método no hará más comprobaciones (no se comprobarán elementos repetidos, un un contenedor dentro de sí mismo ni otro tipo de situaciones equivalentes)

Sobre esta estructura se podrán añadir los métodos y clases que se consideren necesarios, pero ninguno de los métodos deberá lanzar excepciones. No se permite el uso de excepciones en esta práctica (con el objetivo de no complicarla innecesariamente).

Se valorará negativamente el que exista código duplicado entre diferentes clases. Se valorará positivamente la reutilización de código mediante polimorfismo.

3.- Productos especiales

La empresa puede transportar diferentes categorías de productos especiales (**seres vivos** y **productos tóxicos**) que sólo pueden ser transportadas en un contenedor específico para dichas categorías (y nunca directamente en un camión o en un contenedor de otro tipo). El resto de productos se consideran **genéricos** y no necesitan de contenedores especiales y pueden transportarse en un contenedor genérico o directamente en un camión.

Los contenedores seguirán pudiendo introducirse dentro de otros contenedores, siempre y cuando dichos contenedores no sean de productos especiales (o sea, sólo en contenedores de tipo genérico). También siguen pudiendo introducirse en camiones.

Tarea:

Diseña (en el lenguaje elegido) las clases necesarias para representar esta nueva información, modificando lo diseñado en el apartado anterior como sea necesario. Los nombres de las clases (o interfaces) implicadas serán obligatoriamente los siguientes:

- **`SerVivo`** representará producto de tipo ser vivo, con un constructor idéntico al de la clase **`Producto`** diseñada en el apartado anterior.
- **`Toxico`** representará producto tóxico, con un constructor idéntico al de la clase **`Producto`** diseñada en el apartado anterior.
- **`Generico`** representará un elemento genérico, es decir, cualquier cosa que puede guardarse en un contenedor genérico (no especializado) o en un camión.
- La clase **`Producto`** definida en el apartado anterior se considerará que representa un producto genérico (no especial).
- Deberás modificar la clase **`Contenedor`** para que permita definir contenedores específicos para seres vivos y productos tóxicos, así como contenedores de tipo genérico.
- También deberás modificar el método **`boolean guardar(? elemento)`**, que deberá estar disponible para todos los tipos de contenedores (y para el camión), y además deberá dar un **error de compilación** si el elemento introducido no es del tipo adecuado al contenedor correspondiente (o al camión). Dicho método no deberá lanzar ninguna excepción.

Se valorará negativamente el que exista código duplicado entre diferentes clases. Se valorará positivamente la reutilización de código mediante polimorfismo.

4.- Pruebas

Toda biblioteca de clases, aunque no debería de ser necesario recordarlo, debería ser probada exhaustivamente, para asegurar de que cumple con la funcionalidad requerida, incluyendo los casos en los que debería de dar error de compilación.

Tarea:

Prueba exhaustivamente las clases que has generado. Como ayuda te proporcionamos un archivo (**`Main.java`** o **`main.cc`**) que contiene código que te permitirá probar tu biblioteca. Además, te recomendamos que generes tus propios archivos de prueba y, si lo consideras necesario, añade métodos que te faciliten la depuración del código.

Nota importante:

Tu biblioteca de clases deberá compilar (sin ningún error ni aviso de compilación) con el archivo que te proporcionamos, sin necesidad de modificar dicho archivo. Si no lo hace la evaluación de esta práctica resultará en un 0. Esto te obligará a que el nombre de las clases y los correspondientes métodos coincida con lo que te pedimos en este guión.

Entrega

Deberás entregar todos los archivos de código fuente que hayas necesitado para resolver el problema, excepto el archivo de pruebas que te proporcionamos nosotros (todos menos **`Main.java`** para Java o **`main.cc`** para C++). Adicionalmente, para C++, deberás incluir un archivo **`Makefile`** que se encargue de compilar todos tus archivos `.cc` y generar el ejecutable. No deberás incluir ningún archivo de objeto, ni ejecutable, ni ningún archivo **`.class`**. Deberás comprimir todos los archivos en un archivo comprimido `.zip`, con el siguiente nombre:

- **`practica4_<nip1>_<nip2>.zip`** (incluyendo los dos nips de la pareja) si el trabajo se ha hecho por parejas.
- **`practica4_<nip>.zip`** si el trabajo se ha hecho de forma individual.

Si trabajas en Java asegúrate también que todos los archivos **`.java`** están en la raíz del archivo comprimido (sin uso de paquetes, que no son necesarios en un proyecto tan pequeño). Al descomprimir el archivo y añadir el archivo de pruebas **`Main.java`**, deberá poder compilarse y ejecutarse desde su directorio raíz con las siguientes líneas de comandos:

```
javac Main.java
java Main
```

Si no lo hace la evaluación de esta práctica resultará en un 0.

Si trabajas en C++ deberás proporcionar un **`Makefile`** que se encargue de la compilación. El programa, después de añadir el archivo de pruebas **`main.cc`**, deberá ejecutarse desde su directorio raíz con las siguientes líneas de comandos:

```
make
./main
```

Si no lo hace la evaluación de esta práctica resultará en un 0.