Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

Práctica 3

Tecnología de Programación

Grado en Ingeniería Informática

Objetivos

- 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.

1 Productos, contenedores, 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 tiene un peso determinado (en kilogramos, un número real). Ambos dos datos son fundamentales para poder transportarlos.

La empresa dispone también 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). El peso del contenedor es igual a la suma del peso de todos los elementos contenidos, y los contenedores no tienen ningún límite de peso (en el contexto de este ejercicio). Dentro de un contenedor se puede introducir otro contenedor, siempre y cuando no supere el límite de volumen (tiene que ser, evidentemente más pequeño).

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 o igual a su capacidad. Dentro de un camión también se puede introducir un contenedor (siempre y cuando quepa en términos de volumen). 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 (puedes comprobar la correcta compilación con el programa principal que proporcionamos):

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

Todas las clases tendrán los siguientes métodos:

- `double peso()` (`const` si es en C++) que devuelva el peso correspondiente.
- `double volumen()` (`const` si es en C++) que devuelva el volumen correspondiente.
- `String nombre()` (`std::string` y `const` si es en C++) que devuelva el nombre del objeto (directamente "camion" y "contenedor" si se trata de un camión o de un contenedor, respectivamente).
- `String toString()` (`std::string to_string() const` si es en C++) que devuelva una representación textual del nombre, peso y volumen del objeto. En el caso de un contenedor o camión, mostrará también el contenido (un elemento por cada línea, indentado con tres espacios hacia la derecha).

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

- `bool 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*. No es necesario que hagas ciertas comprobaciones sofisticadas, como elementos repetidos en el mismo o diferentes contenedores o un contenedor dentro de si mismo.

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.

2 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`, `Toxico` y `Generico` representaran productos de tipo ser vivo, tóxico o genérico (no especializado), con un constructor idéntico al de la clase `Producto` diseñada en el apartado anterior.
- `Producto` pasará a representar un producto genérico cualquiera (no especial, ni tóxico ni ser vivo).
- Deberás modificar la clase `Contenedor` para que permita definir contenedores específicos para seres vivos y productos tóxicos, así como contenedores de tipo genérico. Su método `nombre()` tendrá que devolver no sólo que es un contenedor sino de qué tipo es (por ejemplo "contenedor de seres vivos").
- También deberás modificar el método `bool 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) aprovechando el sistema de tipos del lenguaje. Por ejemplo, introducir un contenedor cualquiera dentro de otro contenedor de seres vivos debería dar un error de compilación. 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.

3 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. Para hacerlo, tendrás que poner todas tus cabeceras incluidas desde un único archivo *practica3.h* y lidiar con el *Makefile* para que lo compile todo. Además, te recomendamos que generes tus propios archivos de prueba y, si lo consideras necesario, añade métodos que te faciliten la depuración del código, sobre todo para probar todos los posibles casos en los que tu biblioteca debería dar un fallo de compilación.

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 deben incluir ficheros objeto (*.o) o ejecutables en el caso de C++, ni ficheros de clases compiladas (*.class) de Java.

Si la solución está hecha en C++, deberá ser compilable y ejecutable con los archivos que has entregado y con el *main.cc* que te proporcionamos mediante los siguientes comandos:

```
make
./main
```

Si la solución está hecha en Java, deberá ser compilable y ejecutable con los archivos que has entregado y con el *Main.java* que te proporcionamos mediante los siguientes comandos:

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

En caso de no compilar siguiendo estas instrucciones, el resultado de la evaluación de la práctica será de 0. No se deben utilizar paquetes ni librerías ni ninguna infraestructura adicional fuera de las librerías estándar de los propios lenguajes.

Todos los archivos de código fuente solicitados en este guión deberán ser comprimidos en un único archivo `zip` con el siguiente nombre:

- `practica3_<nip1>_<nip2>.zip` (donde `<nip1>` y `<nip2>` son los NIPs de 6 dígitos los estudiantes involucrados) si el trabajo ha sido realizado por parejas. En este caso sólo uno de los dos estudiantes deberá hacer la entrega.

- `practica3_<nip>.zip` (donde `<nip>` es el NIP de 6 dígitos del estudiante involucrado) si el trabajo ha sido realizado de forma individual.

El archivo comprimido a entregar no debe contener ningún fichero aparte de los fuentes que te pedimos: ningún fichero ejecutable o de objeto, ni ningún otro fichero adicional.

La entrega se hará en la tarea correspondiente a través de la plataforma Moodle: `http://moodle.unizar.es`.