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 la reutilización de código, eligiendo el mecanismo óptimo en cada caso, entre la herencia y la programación genérica.

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

1. Productos, contenedores, camiones

1.1. Planteamiento

Una empresa de paquetería se encarga de transportar una serie de productos entre diferentes ciudades del mundo. Cada **producto** ocupa un volumen determinado (en metros cúbicos, un número real) y tiene un peso determinado (en kilogramos, un número real). Ambos datos son fundamentales para poder transportarlos.

La empresa dispone también de **contenedores**. Cada contenedor tiene una capacidad (en metros cúbicos), que define también el volumen que ocupa. Un contenedor puede contener tantos productos como quepan en dicho contenedor, es decir, el volumen total de todos los productos almacenados dentro de dicho contenedor tiene que ser igual o menor que su capacidad. El peso del contenedor es igual a la suma del peso de todos los elementos contenidos (los contenedores no tienen ningún límite de peso). Dentro de un contenedor se puede introducir otro contenedor, siempre y cuando no se supere su capacidad (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.

1.2. Tareas

Diseña las clases necesarias para representar todos los elementos 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 definido mediante un nombre identificativo (una cadena de texto), su volumen y su peso (valores pedeterminados, definidos al crearlo).
  Su constructor debe seguir la siguiente especificación:
  - En C++:
    `Producto(const string& nombre, double volumen, double peso)`
  - En Java:
    `Producto(String nombre, double volumen, double peso)`

- `Contenedor`: representará un contenedor, con su correspondiente capacidad (que equivale a su volumen). Dicho valor se le pasará en su constructor.
- `Carga`: representará cualquier carga (producto o contenedor) que se puede transportar dentro de un almacén (contenedor o camión).
- `Camion`: representará un camión con su correspondiente capacidad, información que se le pasará en su constructor.

Todas las clases tendrán los siguientes métodos (adapta su sintaxis al lenguaje corespondiente, con el uso de `const` y referencias donde proceda en el caso de C++):

- `string nombre()`
  Devuelve el nombre del ítem (”Camion” y ”Contenedor” si se trata de un camión o de un contenedor, respectivamente).
- `double volumen()`
  Devuelve el volumen del ítem correspondiente.
- `double peso()`
  Devuelve el peso del ítem correspondiente. Para los camiones y contenedores, su peso no es un valor predeterminado, sino la suma de los pesos de su contenido.

Todos los elementos deben poder mostrarse por pantalla con los métodos estándar de cada lenguaje:

- En C++ mediante el operador `'<<'` de inserción en un stream.
  Para ello necesitas implementar la función

  ```
  ostream& operator<<(ostream& os, ???? item)
  ```

  para tus clases.
- En Java mediante el método `System.out.print(...)` de la biblioteca de sistema.
  Para ello debes implementar el método estándar auxiliar

  ```
  String toString()
  ```

  que devuelve una cadena con la información a mostrar para el objeto, y que es usado por el método `print()`, en tus clases.

En ambos casos se mostrará 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 dos espacios hacia la derecha). Puedes ver un ejemplo de salida del programa en los programas principales de cada lenguaje que te suministramos.

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 como argumento (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 guardará y el método devolverá *false*.
  No es necesario que hagas otras comprobaciones sofisticadas, como elementos repetidos en el mismo o diferentes contenedores o un contenedor dentro de sí mismo.

Sobre esta estructura se podrán añadir los métodos y clases que se consideren necesarios.
Ninguno de los métodos deberá lanzar excepciones. No se permite el uso de excepciones en esta práctica (con el objetivo de no complicarla innecesariamente). La estructura del código debe detectar las operaciones no legales (por ejemplo, intentar guardar un camión dentro de un contenedor) mediante la definición adecuada de

los tipos de los parámetros en los distintos métodos, dando un error en tiempo de compilación.
Se valorará negativamente el que exista código duplicado entre diferentes clases.
Se valorará positivamente la reutilización de código mediante polimorfismo (usando la herencia y/o la programación genérica).

2. Productos especiales

2.1. Planteamiento

La empresa quiere transportar también diferentes categorías de **cargas especiales**: **seres vivos** y **productos tóxicos**. Para ello es necesario poder tener contendores específicos para cada uno de esos tipos de productos (además de los normales para **cargas estándar**), de forma que sólo puedan ser transportadas en un contenedor específico para dicha categoría (y nunca directamente en un camión o en un contenedor general o de otro tipo).

El resto de productos se consideran **cargas** estándar, pueden transportarse en un contenedor para cargas estándar o directamente en un camión, y no pueden introducirse en un contenedor para seres vivos o productos tóxicos.

Los contenedores especiales se pueden introducir dentro de los contenedores estándar o de un camión, como cualquier otra carga.

2.2. Tareas

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

- `Producto` pasará a representar un producto estándar (no especial, ni tóxico ni ser vivo), que puede guardarse en almacenes para carga estándar (contenedores normales o camiones).
- `SerVivo` y `Toxico` representaran productos especiales, de tipo ser vivo o tóxico, con un constructor idéntico al de la clase `Producto`.
- `Carga` representará cualquier carga estándar (productos normales u otros contenedores) que se puede transportar dentro de un almacén para carga estándar.
- Deberás modificar la clase `Contenedor` para que permita definir contenedores específicos para seres vivos y productos tóxicos (que sólo permitan guardar productos de dichos tipos), así como contenedores para carga estándar (en los que no se pueden guardar cargas especiales –vivos o tóxicos–). 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 `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 a guardar no es del tipo adecuado al contenedor correspondiente (o al camión) aprovechando el sistema de tipos del lenguaje. Por ejemplo, intentar introducir un contenedor cualquiera dentro de otro contenedor de seres vivos debería dar un error de compilación. Dicho método no deberá lanzar ninguna excepción.

La estructura del código debe detectar las operaciones no legales (por ejemplo, intentar guardar un ser vivo en un contenedor estándar, o un camion dentro de un contenedor) dando un error en tiempo de compilación.
Se valorará negativamente el que exista código duplicado entre diferentes clases.
Se valorará positivamente la reutilización de código mediante polimorfismo (usando la herencia y/o la programación genérica).

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.

3.1. Tarea

Prueba exhaustivamente las clases que has generado. Como ayuda te proporcionamos un archivo (*main.cc* o *Main.java*) que contiene código que te permitirá probar tu biblioteca. En el caso de C++ tendrás que incluir todas tus cabeceras desde un único archivo *practica3.h* y lidiar con el *Makefile* para que lo compile todo. No debes escribir todo tu código en *practica3.h*, define tus propios ficheros de cabecera e incúyelos desde *practica3.h*. 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 programa principal 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. 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 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 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`