TECNOLOGÍA DE PROGRAMACIÓN **Curso 2015-2016**

Práctica 2

1.- Descripción general de la práctica

El objetivo de esta práctica es tener una primera experiencia con el polimorfismo tanto paramétrico (basado en programación genérica) como por herencia. Como objetivo secundario, compararemos la gestión de memoria de C++ y de Java.

2.- Agrupación dinámica

En la práctica anterior desarrollaste un tipo abstracto de dato en forma de clase, denominado `agrupacion`, tanto en Java como en C++. Dicha colección de elementos tiene una serie de operaciones asociadas representadas mediante métodos que son independientes de su implementación interna.

La implementación interna de la práctica anterior, no obstante, era puramente estática. Esto tiene ciertas ventajas (contigüidad en memoria, que favorece a la cache) pero tiene la desventaja de tener un límite de tamaño establecido en compilación. Como contrapartida a esta estructura, vamos a diseñar una agrupación dinámica (tanto en Java como en C++) en el que, al añadir cada elemento, se añada un nodo a la estructura de forma dinámica, sin límite de tamaño (salvo el establecido por la propia memoria de la máquina).

Renombra las clases / archivos de la práctica anterior:

- Para **C++**, coge el tipo de dato `agrupacion` que generaste en la práctica anterior y renómbralo como `agrupacion_estatica` (renombra el archivo desde `agrupacion-class.h` a `agrupacion-estatica.h`).
- Para **Java**, coge el tipo de dato `Agrupacion` de la práctica anterior y renómbralo como `AgrupacionEstatica` (renombra el archivo desde `Agrupacion.java` a `AgrupacionEstatica.java`).

Después, para Java y C++, genera dos clases nuevas que representen el mismo TAD cambiando los pero con la implementación dinámica. Mantén todos los métodos públicos (de clase e iterador) cambiando la implementación de los métodos y los atributos:

- Para **C++**, llama a la clase `agrupacion_dinamica` y ponlo en un fichero `agrupacion-dinamica.h`.
- Para **Java**, llama a la clase `AgrupacionDinamica` en el fichero `AgrupacionDinamica.java`.

Deberás encargarte de gestionar la memoria dinámica (la creación y destrucción, en su caso, de los nodos). Para ello la clase (tanto en C++ como en Java) deberá tener una clase interna (al estilo del iterador) que represente un nodo. Estos nodos se crearán al añadir un elemento y se destruirán cuando se borre el último nodo (y cuando el programa acabe).

En la liberación / destrucción de la memoria dinámica es donde notarás la mayor diferencia en ambos lenguajes. En Java no es necesario preocuparse de eso (la propia máquina virtual se encarga). Sin embargo, en C++, deberás definir un destructor tanto para la clase `agrupacion_dinamica` como, si lo ves necesario, para la clase interna que represente un nodo. Además, deberás liberar específicamente la memoria al borrar un elemento de la agrupación (método `borrarUltimo()`).

3.- Polimorfismo

En este punto tienes declaradas dos clases (tanto en Java como en C++) que representan el mismo tipo abstracto de datos con diferentes implementaciones. Ahora deberás conseguir generar código que funcione de forma equivalente con ambos dos tipos de datos.

Para ello te proporcionamos archivos que contienen un programa principal (`main.cc` y `Main.java`, en cada caso, junto con un `Makefile` en el caso de C++), cada uno de ellos con dos “TODO:” que deberás resolver.

Te proporcionamos (en cada caso) la implementación de dos funciones (métodos estáticos, en Java), pero no te proporcionamos la cabecera de dichas funciones (métodos). Deberás crear la cabecera correcta que permita que la función (método) funcione con las dos clases que has desarrollado.

Aquí es, de nuevo, donde la filosofía de Java y de C++ difiere:

- En **Java**, el mecanismo para lograr este tipo de polimorfismo es la herencia. Deberás diseñar un interfaz común a ambas implementaciones de la agrupación en Java que permita el polimorfismo que buscas (llama a esta nueva clase como quieras).
- En **C++**, el mecanismo será la programación genérica. Deberás conseguir que las funciones correspondientes sean genéricas y acepten ambos dos tipos de datos.

En ningún caso toques el resto del código (programa principal, implementación de métodos y funciones) para facilitarte la tarea. Sólo deberás tocar las cabeceras de los correspondientes métodos y funciones.

4.- Entrega

Todos los archivos deben estar contenidos en un directorio de nombre **`practica2_<NIP>`** si la realización es individual, o **`practica2_<NIP1>_<NIP2>`** si es por parejas. La estructura de archivos debe ser la siguiente:

```
practica2_XXXXXX
      \---- java
            \--- AgrupacionEstatica.java
            \--- AgrupacionDinamica.java
            \--- Main.java
            \--- ...
      \--- c++
            \--- agrupacion-estatica.h
            \--- agrupacion-dinamica.h
            \--- main.cc
            \--- Makefile
            \--- ...
```

El programa en C++ debe ser compilable y ejecutable con los archivos que has entregado mediante:

```
make
main
```

El programa en Java debe ser compilable y ejecutable con los archivos que has entregado mediante:

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

En caso de no compilar siguiendo estas instrucciones la evaluación de la práctica será de 0.

El directorio se comprimirá en un archivo con el mismo nombre y extensión `.zip`. El archivo comprimido a entregar no debe contener ningún fichero `.class`, `.o` (objeto) o ejecutable.

No se deben utilizar paquetes ni librerías ni ninguna infraestructura adicional fuera de las librerías estándar de los propios lenguajes.

El fichero `.zip` se entregará a través de Moodle. Si la realización es por parejas, sólo uno de los dos miembros de la pareja deberá entregar la práctica.