jun04-1v.pdf

jvc93

Programación Orientada a Objetos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

Programación Orientada a Objetos

Examen final (1.ª vuelta)

Curso 2003–04 **Lunes 7 de junio de 2004**

1. Defina los principios de encapsulado y ocultación de datos de la programación orientada a objetos. ¿Cómo influyen en la cohesión y el acoplamiento de los componentes de un sistema? **1,0 p**

2. Se desea implementar en C++ una clase para almacenar cadenas de caracteres. Esta clase se denominará *Cadena* y su representación interna estará basada en dos atributos, un puntero a carácter, *pc*, y un tamaño, *n*. El atributo *pc* apuntará a una zona de memoria dinámica creada con **new** que será destruida con **delete** tan pronto como deje de ser necesaria. Dicha zona contendrá los caracteres de la cadena y un carácter adicional terminador (`'\0'`). Por lo tanto, *n* representará la longitud de la cadena, excluido el carácter terminador. La cadena vacía se representará mediante un único carácter terminador. **2,5 p**

   a) Defina un constructor de conversión para crear un objeto *Cadena* a partir un tamaño. El objeto contendrá tantos caracteres de espacio (`' '`) como indique el tamaño. **0,25 p**

   b) ¿Qué hay que hacer para que dicho constructor sirva además de constructor predeterminado (para la cadena vacía) y para que no se realicen conversiones implícitas con él? **0,25 p**

   c) Defina un constructor de conversión para crear un objeto *Cadena* a partir una cadena de caracteres de bajo nivel. **0,25 p**

   d) ¿Es necesario definir un operador de asignación si queremos asignar objetos *Cadena* con la semántica habitual que poseen los tipos primitivos? Si lo es, hágalo. **0,50 p**

   e) Sobrecargue el operador de índice para poder acceder a caracteres individuales de un objeto *Cadena* como si de una cadena de bajo nivel se tratase, tanto en lectura como en escritura (si no es constante), pero siempre con comprobación de rango. Si el índice no está en el rango correcto se lanzará una excepción de tipo *AccesoIndebido* (declarada como una clase vacía). Asegúrese de especificar que puede aparecer dicha excepción. **0,50 p**

   f) Sobrecargue el operador de suma para devolver la concatenación de dos objetos *Cadena*. **0,25 p**

   g) Sobrecargue el operador de inserción para escribir un objeto *Cadena* en un flujo de salida. **0,25 p**

   h) Discuta la conveniencia de sobrecargar los operadores de suma e inserción interna o externamente a la clase. **0,25 p**

3. Las Matemáticas enseñan que una circunferencia es una especie de elipse «degenerada», donde sus dos radios, r_x y r_y, son iguales. Consideremos este principio para desarrollar mediante orientación a objetos una hipotética aplicación en C++. Hay que encontrar una forma conveniente de representar estas figuras geométricas en un modelo de objetos. **2,5 p**

   Tras un primer análisis se identifican dos clases, *Elipse* y *Circunferencia*. La primera dispondrá de dos métodos observadores, *radio_x*() y *radio_y*(), mientras que la segunda dispondrá de un método observador, *radio*(). En ambas clases existirá un método *escala*() que recibirá un factor de escala y modificará los radios en dicho factor. Por último, la clase *Elipse*() poseerá una sobrecarga de *escala*() que recibirá dos factores de escala independientes: uno para r_x y otro para r_y.

   a) ¿Qué relación podemos establecer entre elipses y circunferencias para poder implementar las segundas a partir de las primeras siguiendo el análisis anterior? Escriba todo lo necesario para implementar dicha relación. **1,0 p**

   b) Tras un análisis más profundo consideramos que las figuras planas son una abstracción útil de las elipses, circunferencias y otras figuras geométricas. Toda figura plana constaría de una operación de escalado para incrementar o disminuir su tamaño sin alterar su aspecto. ¿Qué relación podemos establecer entre figuras planas, elipses y circunferencias? Escriba todo lo necesario para implementar dicha relación. **0,5 p**

c) Tras completar las figuras planas con una operación de dibujo, nos damos cuenta de que una serie de figuras planas pueden agruparse en una escena que poseería sus propias operaciones de dibujo y escalado, consistentes en aplicar la operación correspondiente a cada una de las figuras que la integran. ¿Qué relación podemos establecer entre escenas y figuras planas? Escriba todo lo necesario para implementar dicha relación. **1,0 p**

4. Se trata de implementar en C++ una función genérica, *ordenado*(), que decida si los elementos de un rango de iteradores aparecen en un cierto orden. Para lograr la máxima generalidad, los únicos requisitos que la función impondrá es que los iteradores sean de entrada y que la relación de orden tenga las propiedades de un orden estricto débil. También hay que tratar adecuadamente los rangos vacíos, que se suponen ordenados, ya que no contienen elementos. **2,5 p**

a) Escriba una versión que compruebe el orden respecto del **operator** < del tipo en cuestión. **1,0 p**

b) Escriba una versión que compruebe el orden respecto de una función de comparación arbitraria. **0,5 p**

c) Escriba un fragmento de código que compruebe si un vector de bajo nivel de enteros está en orden ascendente de valores absolutos. Emplee un objeto función. **0,5 p**

d) Escriba un fragmento de código que compruebe si un fichero de texto que contiene enteros separados por espacio blanco está en orden ascendente. **0,5 p**

5. Conteste a las siguientes preguntas acerca de las diferencias existentes entre C++ y JAVA. **1,5 p**

a) ¿Qué código JAVA corresponde al siguiente fragmento de código C++? Realice una traducción lo más fiel posible, mantenga idénticos los identificadores y justifique, más allá de la sintaxis, las diferencias que surjan entre ambos códigos. **0,5 p**

```
class Trabajador {
public:
  Trabajador(string n, string a): nombre(n), apellidos(a) {}
  virtual ~Trabajador() {}
  // ...
  virtual void actualizaSueldo(double ipc) { sueldo *= 1.0 + ipc; }
private:
  string nombre, apellidos;
  double sueldo;
};

class Funcionario: public Trabajador {
public:
  Funcionario(string n, string a, string nrp): Trabajador(n, a), nrp(nrp) {}
  // ...
  void actualizaTrienios() { /* ... */ }
  void actualizaSueldo(double ipc) {
    Trabajador::actualizaSueldo(ipc);
    actualizaTrienios();
  }
private:
  string nrp;
};
```

b) ¿Cómo se consigue genericidad en los contenedores de JAVA? Compárelo con los mecanismos presentes en C++ en términos de flexibilidad y eficiencia. **0,5 p**

c) ¿En qué contextos en los que en C++ se emplea herencia múltiple se puede paliar su ausencia en JAVA? ¿Cómo? **0,5 p**