sep04-2v.pdf

jvc93

Programación Orientada a Objetos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

Programación Orientada a Objetos

Examen final (2.ª vuelta)

Curso 2003–04 **Viernes 17 de septiembre 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 fragmentos de ADN. Esta clase se denominará *ADN* y su representación interna estará basada en dos atributos, un puntero, c, y un tamaño, n. El atributo c 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 la secuencia de nucleótidos de una de las hélices. El atributo n representará la longitud de la secuencia y se permitirá la existencia de secuencias vacías, que tendrán longitud 0. **2,5 p**

La clase *ADN* contendrá un tipo enumerado, *Nucleotido*, accesible públicamente, que estará formado por las constantes enumeradas A (adenina), C (citosina), G (guanina) y T (timina).

a) Defina dos constructores: uno predeterminado que cree un objeto *ADN* vacío y otro que cree un objeto *ADN* a partir de una secuencia de nucleótidos representada por dos parámetros (un puntero y un tamaño). **0,25 p**

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

c) Sobrecargue el operador de índice para poder acceder a nucleótidos individuales de un objeto *ADN*, 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**

d) Sobrecargue el operador de suma para devolver la concatenación de dos objetos *ADN*. Suponga que ha sido declarado como amigo. **0,50 p**

e) Sobrecargue el operador de inserción para escribir un objeto *ADN* en un flujo de salida de manera que aparezca la secuencia de iniciales de sus nucleótidos. Suponga que ha sido declarado como amigo. **0,50 p**

f) ¿Qué efecto tendría sobrecargar el operador de inserción internamente? **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++ diversas funciones genéricas relacionadas con conjuntos. **2,5 p**

a) Escriba una función, *disjuntos*(), que reciba dos conjuntos paramétricos y decida si son disjuntos. Pueden serle útiles las siguientes funciones de la STL: **1,0 p**

- *set_intersection*(), que recibe dos rangos de iteradores de entrada $[a, b)$ y $[c, d)$ y un iterador de salida s y coloca los n elementos resultantes de intersectar los elementos de $[a, b)$ y $[c, d)$ en $[s, s + n)$. Los rangos de entrada deben estar ordenados y no puede existir solapamiento entre el rango de salida y los de entrada.
- *front_inserter*(), que recibe un contenedor c y devuelve un insertor delantero para c.

b) Con *disjuntos*(), escriba otra función, *comparten*(), que reciba una lista de conjuntos paramétricos y decida si todas sus parejas comparten algún elemento. **1,0 p**

c) Sobrecargue el operador de inserción para escribir los elementos de un conjunto paramétrico en un flujo de salida. Si está vacío escribirá sólo el carácter `'O'`, en caso contrario, los elementos se escribirán separados por comas y la secuencia completa entre llaves. **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**