Resúmenes 11, 12 y 13

Data Structures and Algorithms in Java

Manuel Arturo Chinchilla Sánchez
2013009344

Esta nueva edición de Java describe estructuras de datos, métodos de organización de grandes cantidades de datos, y el análisis de algoritmos, la estimación del tiempo de ejecución de los algoritmos. Medida que las computadoras se vuelven más y más rápido, la necesidad de programas que pueden manejar grandes cantidades de entrada se hace más aguda. Paradójicamente, esto requiere más cuidadosa atención a la eficiencia, ya que las ineficiencias en los programas se hacen más evidentes cuando los tamaños de entrada son grandes. Mediante el análisis de un algoritmo antes de que sea realmente codificado, los estudiantes pueden decidir si una solución particular será factible. Por ejemplo, en este texto los estudiantes miran a problemas específicos y ver cómo implementaciones cuidadosas pueden reducir las limitaciones de tiempo para grandes cantidades de datos desde siglos en menos de un segundo. Por lo tanto, ningún algoritmo o datos de estructura se presentan sin una explicación de su tiempo de ejecución. En algunos casos, los detalles minuciosos que afectan el tiempo de ejecución de la aplicación se exploran. Una vez que se determina un método de solución, un programa aún debe ser escrito. Como afeitado equipo ser más poderosa, los problemas que deben resolver se han convertido en más grande y más complejo, que requiere el desarrollo de los programas más complejos. El objetivo de este texto es enseñar a los estudiantes buenos conocimientos de programación y análisis de algoritmo de forma simultánea para que puedan desarrollar este tipo de programas con el máximo de eficiencia. Este libro es adecuado, ya sea para un (CS7) Curso de estructuras de datos avanzadas o un curso de postgrado de primer año en el análisis de algoritmos. Los estudiantes deben tener algún conocimiento de programación intermedia, incluyendo temas como la programación y la recursividad basada en objetos, y algunos antecedentes en matemática discreta.

La idea básica en Java es que podemos implementar una clase genérica mediante el uso de una superclase apropiada, tal como Object. Un ejemplo es la clase de memoria de celda. Hay dos detalles que deben ser considerados cuando utilizamos esta estrategia. La primera es la de representar a un hombre que escribe un "37" para un objeto de memoria de celda y luego lee desde el objeto de memoria de celda. Para acceder a un método específico del objeto, debemos abatido al tipo

correcto. (Por supuesto, en este ejemplo, no necesitamos los abatidos, ya que simplemente estamos invocando el método de cadena en la línea 9, y esto se puede hacer para cualquier objeto.) Un segundo detalle importante es que los tipos primitivos no se pueden utilizar. Sólo los tipos de referencia son compatibles con objetos. Una solución estándar para este problema se discute momentáneamente.

Cuando implementamos algoritmos, a menudo nos encontramos con un problema escribiendo idioma: Tenemos un objeto de un tipo, pero la sintaxis del lenguaje requiere un objeto de un tipo diferente. Esta técnica ilustra el tema básico de una clase contenedora. Un uso típico es para almacenar un tipo primitivo, y añadir las operaciones que el tipo primitivo o bien no admite o no admite correctamente. En Java, ya hemos visto que a pesar de todos los tipos de referencia es compatible con el objeto, los ocho tipos primitivos no lo son. Como resultado, Java proporciona una clase de contenedor para cada uno de los ocho tipos primitivos. Por ejemplo, la envoltura para el tipo int es Integer. Cada objeto envoltorio es inmutable (que significa su estado nunca puede cambiar), tiendas de un valor primitivo que se establece cuando se construye el objeto, y proporciona un método para recuperar el valor. Las clases contenedoras también contienen una gran cantidad de métodos de utilidad estáticas.

El uso de objetos como un tipo genérico sólo funciona si las operaciones que se están realizando canbe expresan utilizando sólo los métodos disponibles en la clase de objetos. Considere, por ejemplo, el problema de encontrar el elemento máximo en una gran variedad de artículos. El código básico es de tipo independiente, pero sí requiere la capacidad de comparar dos objetos y decidir cuál es más grande y que es más pequeño. Por lo tanto no podemos simplemente encontrar el máximo de un conjunto de objetos que necesitamos más información. La idea más simple sería encontrar el máximo de un conjunto de Comparable. Para determinar el orden, podemos utilizar la comparamos con el método que sabemos debe estar disponible para todos los Comparables. El código para hacer esto se muestra en la Figura 1.8, que ofrece a un hombre que encuentra el máximo en un conjunto de cuerdas o de forma. Es importante mencionar algunas advertencias. En primer lugar, sólo los

objetos que implementan la interfaz Comparable se pueden pasar como elementos de la matriz comparable. Los objetos que tienen un comparan con método, pero no declaran que no implementan Comparable no son comparables, y no tienen el requisito de la relación IS-A. Por lo tanto, se presume que los implementos de forma