Universidad de Panamá
Facultad de Informática, Electrónica y Comunicación
Centro Regional Universitario de Veraguas

Parcial No. 3

排	列
0	1

Autor
Faustino Aguilar Quirós
2-732-727

Facilitador
Prof. Abundio Mendoza

II° Semestre de 2015

DEFINICIÓN DEL PROBLEMA

Se trata de investigar, experimentar y presentar los resultados de un estudio comparativo de las *Clases ArrayList, Vector y Array*. Para apoyar su investigación, resuelva el siguiente problema con cada una de las estructuras de datos antes mencionadas:

Un vector contiene los códigos de empleados y en las mismas posiciones de otro vector están los índices o lugares donde se encuentra los nombres correspondientes a los empleados. Los nombres están almacenados en un tercer vector. Escriba un programa que capture n código de empleados y despliegue los nombres correspondientes, ordenados alfabéticamente.

COMPARACIÓN

A continuación presentamos un análisis que nos aclara las principales características de estos tipos de datos.

El Vector es mejor según las circunstancias, otras veces el ArrayList es mejor, algunas veces no conviene usar ninguno de los dos y lo mejor es usar Arrays sencillos.

Hay tres puntos principales a remarcar cuando tenemos este dilema:

Primero, el API

En el "The Java Programming Language "se describe que el Vector es analogo al ArrayList. Asi que el Api en perspectiva, las dos clases son muy similares, pero hay algunas pequeñas diferencias entre ellas, como el consumo de memoria y la eficiencia, muchos autores comentas que ArrayList es el sucesor natural de Vector.

Sincronización

El Vector es sincronizado. Lo que implica que cada metodo tocado de un vector es threads safe. El ArrayList, por otro lado es de-sincronizado, haciendo no threads safe. La principal diferencia que podemos llegar a notar es cuando necesitamos mayor rendimiento.

Crecimiento de información

Tanto ArrayList<E> como Vector<E> utilizan un objeto Array<E> para almacenar los elementos internamente. Las colecciones de tipo Array<E>, sin embargo, tienen un tamaño fijo que se determina de antemano al llamar al constructor.

Por defecto ArrayList<E> y Vector<E> utilizan arrays con capacidad para 10 elementos. Cuando el número de elementos sobrepasa la capacidad disponibleVector<E> dobla el tamaño del array interno, mientras que ArrayList<E> utiliza la fórmula (capacidad * 3) / 2 + 1.

Para indicar valores que se ajusten lo máximo posible al uso que vamos a hacer de nuestra lista tanto Vector<E> como ArrayList<E> cuentan, además de con un constructor vacío y un constructor al que pasar una colección con los elementos con los que inicializar el vector o la lista, con un constructor al que pasar un entero con la capacidad inicial de la colección.

Vector<E> cuenta también con un constructor extra mediante el que indicar el incremento en capacidad a utilizar cuando el número de elementos rebase la capacidad del vector, lo cuál puede ser muy interesante.

OBTENCIÓN DE LA INFORMACIÓN

A continuación presentamos las referencias principales que se utilizó para realizar el trabajo.

- Savitch, W., & Carrano, F. M. (2008). Java: Introduction to Problem Solving and Programming. Prentice Hall Press. **Utilizado para extración de código y ejemplos.**
- Boisvert, R. F., Dongarra, J. J., Pozo, R., Remington, K., & Stewart, G. W. (1998). Developing numerical libraries in Java. arXiv preprint cs/9809009. **Utilizado para extración de código y ejemplos.**
- Gaddis, T. (2009). Starting Out with Java: From Control Structures Through Objects. Addison-Wesley Publishing Company. **Utilizado para extración de código y ejemplos.**
- Kurzyniec, D., & Sunderam, V. (2001, June). Efficient cooperation between Java and native codes–JNI performance benchmark. In The 2001 international conference on parallel and distributed processing techniques and applications. **Utilizado para comprender mejor ciertos puntos de eficiencia en Java.**
- Odersky, M., & Wadler, P. (1997, January). Pizza into Java: Translating theory into practice. In Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (pp. 146-159). ACM. **Utilizado para extración de código y ejemplos.**
- Pratt, T. W., Zelkowitz, M. V., & Gopal, T. V. (1984). Programming languages: design and implementation (No. 04; QA76. 7, P8 1984.). Englewood Cliffs: Prentice-Hall.**Utilizado para extración de código y ejemplos.**
- Gundersen, G., & Steihaug, T. (2004). Data structures in Java for matrix computations. Concurrency and computation: Practice and Experience, 16(8), 799-815. **Utilizado para comprender mejor la relación entre Java y sus estructuras de datos.**
- Artigas, P. V., Gupta, M., Midkiff, S. P., & Moreira, J. E. (2000). High performance numerical computing in Java: Language and compiler issues (pp. 1-17). Springer Berlin Heidelberg. **Utilizado para extración de código y ejemplos.**
- Havelund, K., & Pressburger, T. (2000). Model checking java programs using java pathfinder. International Journal on Software Tools for Technology Transfer, 2(4), 366-381. **Utilizado para extración de código y ejemplos.**
- Dietel, P. (2009). Java how to program. PHI. **Utilizado para análizar los métodos disponibles en diferentes clases y utilidades de Java.**

Código Fuente Comentado

↓ Vector.java

```
import java.util.*;

class VectorTest {

  /* Los atributos princioales son
  los ectores del problema */
  private Vector<String> employeeCode = new Vector<String>();
  private Vector<Integer> employeeNameIndex = new Vector<Integer>();
  private Vector<String> employeeName = new Vector<String>();

  VectorTest() {
    /*
    En este caso se crean los vectores de códigos,
    índices y nombres de los empleados.
    */
    this.employeeCode.add("B011");
    this.employeeCode.add("A001");
    this.employeeCode.add("A000");
    this.employeeCode.add("B010");
    this.employeeCode.add("C100");
    this.employeeNameIndex.add(3);
    this.employeeNameIndex.add(1);
    this.employeeNameIndex.add(0);
    this.employeeNameIndex.add(2);
    this.employeeNameIndex.add(4);
    this.employeeName.add("Faustino Aguilar");
    this.employeeName.add("Carlos Álvarez");
    this.employeeName.add("Alcides Zurita");
    this.employeeName.add("Efraín Aguilar");
    this.employeeName.add("Cristian Siu");
  }

  private static void main(String[] args) {
    /*EN la funcipn  principal de crea la
    instancia de la clase VectorTest y se
    llama al método que hace funcional el
    programa y que enlaza a los demás métodos*/
    VectorTest employee = new VectorTest();
    try {
      employee.getCodes();
    } catch (Exception e) {
      System.out.println("Ocurrió un error: " + e);
    }
  }

  private void getCodes() {
    // Obtención de códigos
    Integer i, n;
```

```
    String code;
    Scanner input = new Scanner(System.in);
    Vector<String> codeStack = new Vector<String>();
    Vector<String> nameStack = new Vector<String>();
    System.out.println("\nPROGRAMA PARA GESTIÓN DE EMPLEADOS");
    System.out.print("Escriba la cantidad de códigos para analizar: ");
    n = input.nextInt();
    input.nextLine();
    for(i=1; i<=n; i++) {
      System.out.print("Escriba el código del " + i + "° empleado XXXX: ");
      code = input.nextLine();
      codeStack.add(code);
      nameStack.add(filterCodes(code));
    }
    printCodes(codeStack, nameStack);
  }

  private String filterCodes(String code) {
    // Filtra los códigos en los vectores
    int index, element;
    index = this.employeeCode.indexOf(code);
    if (index >= 0) {
      element = this.employeeNameIndex.get(index);
      return this.employeeName.get(index);
    } else {
      return "¡Código Inválido!";
    }
  }

  private void printCodes(Vector<String> nameStack, Vector<String> codeStack) {
    // Imprime en pantalla el resultado
    // Hashtable solo es un ebellecedor
    Hashtable<String, String> nameCodes = new Hashtable<String, String>();
    int i;
    for (i=0; i < codeStack.size(); i++) {
      nameCodes.put(nameStack.get(i), codeStack.get(i));
    }
    Collections.sort(nameStack);
    System.out.println("\nCódigo | Empleado");
    System.out.println("------ | --------");
    for (String name : nameStack) {
      System.out.println(" " + nameCodes.get(name) + "  | " + name);
    }
    System.out.println();
  }
}
```

↓ Array.java

```
import java.util.*;

// Desarrollo del problema utilizando Arrays

class ArrayTest {

  // Atributos del la clase
  // Son simples arrays en comparación con los objetos Vector y List
  private String[] employeeCode = new String[5];
  private Integer[] employeeNameIndex = new Integer[5];
  private String[] employeeName = new String[5];

  ArrayTest() {
    this.employeeCode[0] = "B011";
    this.employeeCode[1] = "A001";
    this.employeeCode[2] = "A000";
    this.employeeCode[3] = "B010";
    this.employeeCode[4] = "C100";
    this.employeeNameIndex[0] = 3;
    this.employeeNameIndex[1] = 1;
    this.employeeNameIndex[2] = 0;
    this.employeeNameIndex[3] = 2;
    this.employeeNameIndex[4] = 4;
    this.employeeName[0] = "Faustino Aguilar";
    this.employeeName[1] = "Carlos Álvarez";
    this.employeeName[2] = "Alcides Zurita";
    this.employeeName[3] = "Efraín Aguilar";
    this.employeeName[4] = "Cristian Siu";
  }

  public static void main(String[] args) {
    ArrayTest employee = new ArrayTest();
    try {
      employee.getCodes();
    } catch (Exception e) {
      System.out.println("Ocurrió un error: " + e);
    }
  }

  private void getCodes() {
    // Obtiene los códigos y los almacena en un array a comparación con
    // las otras clases, utilizando un índice.
    Integer i, n;
    String code;
    Scanner input = new Scanner(System.in);
    System.out.println("\nPROGRAMA PARA GESTIÓN DE EMPLEADOS");
    System.out.print("Escriba la cantidad de códigos para analizar: ");
    n = input.nextInt();
    input.nextLine();
    String[] codeStack = new String[n];
    String[] nameStack = new String[n];
```

```
    for(i=1; i<=n; i++) {
      System.out.print("Escriba el código del " + i + "° empleado XXXX: ");
      code = input.nextLine();
      codeStack[i-1] = code;
      nameStack[i-1] = filterCodes(code);
    }
    printCodes(codeStack, nameStack);
  }

  private String filterCodes(String code) {
    // Foltra el código y restorna su resultado
    int index, element;
    index = Arrays.asList(this.employeeCode).indexOf(code);
    if (index >= 0) {
      element = this.employeeNameIndex[index];
      return this.employeeName[index];
    } else {
      return "¡Código Inválido!";
    }
  }

  private void printCodes(String[] nameStack, String[] codeStack) {
    // Imprime una serie de códigos con su respectivo
    // resultado en pantalla
    Hashtable<String, String> nameCodes = new Hashtable<String, String>();
    int i;
    for (i=0; i < codeStack.length; i++) {
      nameCodes.put(nameStack[i], codeStack[i]);
    }
    Arrays.sort(nameStack);
    System.out.println("\nCódigo | Empleado");
    System.out.println("------ | --------");
    for (String name : nameStack) {
      System.out.println(" " + nameCodes.get(name) + "  | " + name);
    }
    System.out.println();
  }

}
```

↓ ArrayList.java

```
import java.util.*;

// Este programa usa ArrayList para resolver el problema

class ArrayListTest {

  /* Los atributos princioales son
  los ectores del problema */
  private ArrayList<String> employeeCode = new ArrayList<String>();
  private ArrayList<Integer> employeeNameIndex = new ArrayList<Integer>();
  private ArrayList<String> employeeName = new ArrayList<String>();
```

```
ArrayListTest() {
  /*
  En este caso se crean los ArrayList de códigos,
  índices y nombres de los empleados.
  */
  this.employeeCode.add("B011");
  this.employeeCode.add("A001");
  this.employeeCode.add("A000");
  this.employeeCode.add("B010");
  this.employeeCode.add("C100");
  this.employeeNameIndex.add(3);
  this.employeeNameIndex.add(1);
  this.employeeNameIndex.add(0);
  this.employeeNameIndex.add(2);
  this.employeeNameIndex.add(4);
  this.employeeName.add("Faustino Aguilar");
  this.employeeName.add("Carlos Álvarez");
  this.employeeName.add("Alcides Zurita");
  this.employeeName.add("Efraín Aguilar");
  this.employeeName.add("Cristian Siu");
}

private static void main(String[] args) {
  /*En la función  principal de crea la
  instancia de la clase ArrayListTest y se
  llama al método que hace funcional el
  programa y que enlaza a los demás métodos*/
  ArrayListTest employee = new ArrayListTest();
  try {
    employee.getCodes();
  } catch (Exception e) {
    System.out.println("Ocurrió un error: " + e);
  }
}

private void getCodes() {
  // Función para obtener los códigos a analizar
  Integer i, n;
  String code;
  Scanner input = new Scanner(System.in);
  ArrayList<String> codeStack = new ArrayList<String>();
  ArrayList<String> nameStack = new ArrayList<String>();
  System.out.println("\nPROGRAMA PARA GESTIÓN DE EMPLEADOS");
  System.out.print("Escriba la cantidad de códigos para analizar: ");
  n = input.nextInt();
  input.nextLine();
  for(i=1; i<=n; i++) {
    System.out.print("Escriba el código del " + i + "° empleado XXXX: ");
    code = input.nextLine();
    codeStack.add(code);
    nameStack.add(filterCodes(code));
  }
  printCodes(codeStack, nameStack);
```

```
  }

  private String filterCodes(String code) {
    // Esta función permite que los códigos sean parseados en los demas arreglos
    int index, element;
    index = this.employeeCode.indexOf(code);
    if (index >= 0) {
      element = this.employeeNameIndex.get(index);
      return this.employeeName.get(index);
    } else {
      return "¡Código Inválido!";
    }
  }

  private void printCodes(ArrayList<String> nameStack, ArrayList<String>
codeStack) {
    // Esta función imprime los códigos de manéra agradable.
    // Se usó HashTable con el motivo de tener una mejor visualización
    // y no forma parte de la lógica del problema.
    Hashtable<String, String> nameCodes = new Hashtable<String, String>();
    int i;
    for (i=0; i < codeStack.size(); i++) {
      nameCodes.put(nameStack.get(i), codeStack.get(i));
    }
    Collections.sort(nameStack);
    System.out.println("\nCódigo | Empleado");
    System.out.println("------ | --------");
    for (String name : nameStack) {
      System.out.println(" " + nameCodes.get(name) + "  | " + name);
    }
    System.out.println();
  }
}

// Este programa usa ArrayList para resolver el problema

class ArrayListTest {

  /* Los atributos princioales son
  los ectores del problema */
  private ArrayList<String> employeeCode = new ArrayList<String>();
  private ArrayList<Integer> employeeNameIndex = new ArrayList<Integer>();
  private ArrayList<String> employeeName = new ArrayList<String>();

  ArrayListTest() {
    /*
    En este caso se crean los ArrayList de códigos,
    índices y nombres de los empleados.
    */
    this.employeeCode.add("B011");
    this.employeeCode.add("A001");
    this.employeeCode.add("A000");
    this.employeeCode.add("B010");
```

```
    this.employeeCode.add("C100");
    this.employeeNameIndex.add(3);
    this.employeeNameIndex.add(1);
    this.employeeNameIndex.add(0);
    this.employeeNameIndex.add(2);
    this.employeeNameIndex.add(4);
    this.employeeName.add("Faustino Aguilar");
    this.employeeName.add("Carlos Álvarez");
    this.employeeName.add("Alcides Zurita");
    this.employeeName.add("Efraín Aguilar");
    this.employeeName.add("Cristian Siu");
  }

  private static void main(String[] args) {
    /*En la función  principal de crea la
    instancia de la clase ArrayListTest y se
    llama al método que hace funcional el
    programa y que enlaza a los demás métodos*/
    ArrayListTest employee = new ArrayListTest();
    try {
      employee.getCodes();
    } catch (Exception e) {
      System.out.println("Ocurrió un error: " + e);
    }
  }

  private void getCodes() {
    // Función para obtener los códigos a analizar
    Integer i, n;
    String code;
    Scanner input = new Scanner(System.in);
    ArrayList<String> codeStack = new ArrayList<String>();
    ArrayList<String> nameStack = new ArrayList<String>();
    System.out.println("\nPROGRAMA PARA GESTIÓN DE EMPLEADOS");
    System.out.print("Escriba la cantidad de códigos para analizar: ");
    n = input.nextInt();
    input.nextLine();
    for(i=1; i<=n; i++) {
      System.out.print("Escriba el código del " + i + "° empleado XXXX: ");
      code = input.nextLine();
      codeStack.add(code);
      nameStack.add(filterCodes(code));
    }
    printCodes(codeStack, nameStack);
  }

  private String filterCodes(String code) {
    // Esta función permite que los códigos sean parseados en los demas arreglos
    int index, element;
    index = this.employeeCode.indexOf(code);
    if (index >= 0) {
      element = this.employeeNameIndex.get(index);
      return this.employeeName.get(index);
```

```
    } else {
      return "¡Código Inválido!";
    }
  }

  private void printCodes(ArrayList<String> nameStack,
                          ArrayList<String> codeStack) {
    // Esta función imprime los códigos de manéra agradable.
    // Se usó HashTable con el motivo de tener una mejor visualización
    // y no forma parte de la lógica del problema.
    Hashtable<String, String> nameCodes = new Hashtable<String, String>();
    int i;
    for (i=0; i < codeStack.size(); i++) {
      nameCodes.put(nameStack.get(i), codeStack.get(i));
    }
    Collections.sort(nameStack);
    System.out.println("\nCódigo | Empleado");
    System.out.println("------ | --------");
    for (String name : nameStack) {
      System.out.println(" " + nameCodes.get(name) + "  | " + name);
    }
    System.out.println();
  }
}
```

EVIDENCIA

```
PROGRAMA PARA GESTIÓN DE EMPLEADOS

Escriba la cantidad de códigos para analizar: 2
Escriba el cód del 1° empleado XXXX: A000
Escriba el cód del 2° empleado XXXX: B010


Código | Empleado
------ | --------
 B010  | Alcides Zurita

 A000  | Faustino Aguilar
```

COMENTARIOS Y REFLEXIONES

Java es un lenguaje muy versátil que nos ha permitido a los desarrolladores crear cientos de aplicaciones en diferentes ámbitos. Esta es la ventaja que más valor tiene para los nosotros hoy en día, pues esa independencia hace que ya no sea necesario que, a la hora de realizar cambios, sea la misma persona y desde el mismo equipo la que deba acceder al programa, sino que cualquier programador podrá acceder desde otro ordenador.

Java es un lenguaje que se adapta a la perfección a todo tipo de dispositivos móviles como las incipientes tablets o smartphones, una ventaja que permite que cualquier desarrollo de software creado a través de Java sea visible desde cualquier lugar y en cualquier momento.

Una de las mayores ventajas que le he visto a Java es que con este lenguaje es posible hacer casi cualquier elemento o aplicación, además de las atractivas páginas web dinámicas que, mediante XML, ofrecen un diseño mucho más atractivo que una página estática. Además permite incluir sonido y objetos multimedia así como bases de datos y otras funcionalidades.