APRENDERAPROGRAMAR.COM

LA CLASE VECTOR DEL API JAVA. MÉTODOS TRIMTOSIZE Y ENSURECAPACITY EJEMPLO Y EJERCICIOS RESUELTOS. (CU00922C)

Sección: Cursos
Categoría: Lenguaje de programación Java nivel avanzado I

Fecha revisión: 2039

Resumen: Entrega nº22 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: Manuel Sierra y José Luis Cuenca

CLASE VECTOR

Vamos a continuar el estudio de la interface List del api de Java, pero esta vez nos centraremos en la implementación de esta interface a través de la clase Vector. También veremos las características más importantes de esta nueva implementación y haremos un ejemplo a modo de ejercicio.

VECTOR

La clase Vector, al igual que ArrayList o LinkedList, también implementa a List, pero de un modo especial. Este modo especial es sincronizado, lo que permite que se pueda usar en entornos concurrentes (es decir, en varios procesos que se ejecutan al mismo tiempo y hacen uso posiblemente de los mismos recursos). Esta además es la principal característica que la diferencia de otras clases estudiadas anteriormente como ArrayList.

Se recomienda que si no es necesario se use ArrayList en vez de Vector. Por tanto, solo utilizaremos la clase Vector si tenemos previstas circunstancias especiales como procesos concurrentes.

Vamos a ver las principales características de esta clase, que además es muy parecida a ArrayList.

Un objeto de tipo Vector contiene elementos que pueden ser accedidos por un índice y puede aumentar o disminuir su tamaño dinámicamente en tiempo de ejecución.

EJEMPLO USO CLASE VECTOR

Vamos a realizar un ejemplo de uso sobre la clase Vector, donde añadiremos elementos, eliminaremos elementos y consultaremos la capacidad del vector. Para ello también usaremos la clase Persona que hemos venido utilizando en ocasiones anteriores durante el curso:

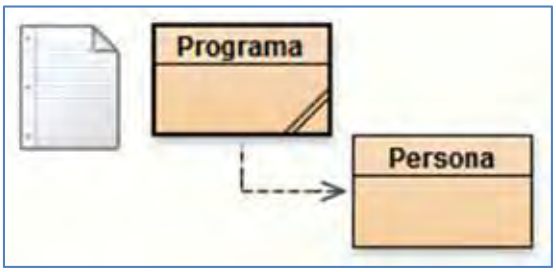
```
/* Ejemplo Interface List, clase Vector aprenderaprogramar.com */
public class Persona{

   private int idPersona;    private String nombre;    private int altura;

   public Persona(int idPersona, String nombre, int altura) {
      this.idPersona = idPersona;       this.nombre = nombre;       this.altura = altura;}

   @Override
   public String toString() {
      return "Persona-> ID: "+idPersona+" Nombre: "+nombre+" Altura: "+altura+"\n";
   }
}
```

Vamos a escribir ahora el siguiente código con el que vamos a trabajar:

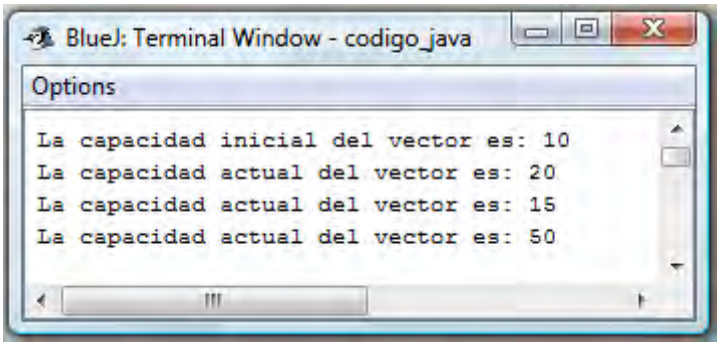
```
/* Ejemplo Interface List, clase Vector aprenderaprogramar.com */
import java.util.Vector;
public class Programa {
   public static void main(String arg[]) {
        Vector<Persona> v = new Vector<Persona>();
        System.out.println("La capacidad inicial del vector es: "+v.capacity());
        int i = 0;
        while (i<15)
        {
           v.add(new Persona(i,"Persona"+i,i));
           i++;
        }
        System.out.println("La capacidad actual del vector es: "+v.capacity());
        v.trimToSize();
        System.out.println("La capacidad actual del vector es: "+v.capacity());
        v.ensureCapacity(50);
        System.out.println("La capacidad actual del vector es: "+v.capacity());
    }
}
```

Como podemos observar en el código, hemos creado una variable v de la clase Vector<Persona> sobre la que consultamos sus capacidades en distintos momentos (antes de insertar elementos en el vector, y tras realizar inserciones o cambios). Hemos utilizado algunos métodos que comentaremos ahora.

El diagrama de clases que podemos ver en BlueJ será similar al siguiente:



La salida que obtendremos por consola será similar a esta:



Como hemos visto la capacidad inicial cuando creamos un nuevo vector es 10 (capacidad que asigna Java por defecto si no se especifica otra), y el incremento definido en caso de ampliación del vector es 0. El incremento en caso de ampliación establece cómo se aumentará el tamaño del vector en caso de que su capacidad vaya a ser excedida al incrementarse el número de elementos en él y podemos establecerlo en 10, 20, 50, etc. elementos (los que nosotros queramos). Tanto la capacidad inicial como el incremento definido pueden establecerse usando constructores especiales que incluyen estos parámetros. En caso de no usar estos constructores, Java utiliza valores por defecto. El valor por defecto 0 para el incremento significa que si se necesita ampliar la capacidad lo que se hace por defecto es duplicar la capacidad del vector.

En nuestro ejemplo el vector se crea con capacidad inicial por defecto 10. Después añadimos 15 elementos dentro del bucle while. Una vez se completa la capacidad inicial, para añadir el elemento 11 el vector necesita ampliar su capacidad y como el incremento de capacidad está fijado por defecto como 0, entonces se duplica su capacidad a 20.

Posteriormente hacemos uso del método de la clase trimToSize(); este método lo que hace es ajustar la capacidad del vector al tamaño real que tiene en ese momento. Por tanto su capacidad nuevamente se ve alterada hasta el valor 15.

Finalmente utilizamos el método ensureCapacity(50); este método permite ampliar la capacidad hasta un valor determinado para asegurarnos de que el vector tendrá capacidad suficiente y hacer reserva de memoria hasta un número dado de elementos, en nuestro ejemplo 50 elementos.

Recordar que si quisiéramos una determinada capacidad inicial y un determinado incremento por defecto nos bastaría con crear el vector pasando los parámetros de forma adecuada en el constructor. Puedes consultar los constructores disponibles en la documentación de la clase.

EJERCICIO

Crea una clase denominada Paquete con los atributos idPaquete (int) y pesoPaquete (int), donde pesoPaquete supondremos que es un dato en kg que podrá tomar valores entre 80 y 150.

Crea una clase con el método main donde se cree una lista de tipo Vector que tendrá una capacidad inicial de 5 y un incremento de 1. Esta lista será “el contenedor” donde guardaremos los paquetes. Este contenedor podrá cargar un peso máximo de 100 kilos multiplicado por su capacidad (capacity de Vector). Es decir, el contenedor inicialmente podrá cargar 500 kilos. Este dato lo calcularemos y lo denominaremos cargaMaximaContenedor.

Introduciremos 50 paquetes en la lista. El atributo pesoPaquete debe establecerse para cada objeto de forma aleatoria. Al introducir los paquetes la capacidad (capacity) del contenedor se ampliará a 50, por lo que calcularemos de nuevo el valor de cargaMaximaContenedor. Ahora deberemos obtener que se podrá cargar hasta 5000 kilos, ya que la nueva capacidad (capacity) será 50. Muestra un mensaje por consola informando de la capacidad del contenedor, de su carga máxima y del número de paquetes que contiene y el peso total de estos paquetes.

Si el peso total de los paquetes resulta superior al peso máximo que puede cargar el contenedor deberemos aumentar la capacidad (capacity) del contenedor en tantos elementos como fuera necesario para que el peso total de los paquetes sea inferior al peso máximo que puede cargar el contenedor y mostrar un mensaje informando de ello.

Ejemplo de tres posibles ejecuciones del programa:

1) Situación: un Contenedor con capacidad para 50 paquetes podría cargar 5000 kilos y tenemos 50 paquetes con un peso total de: 5129
Se ha aumentado la capacidad a 52 paquetes lo que permite cargar hasta 5200 kilos

2) Situación: un Contenedor con capacidad para 50 paquetes podría cargar 5000 kilos y tenemos 50 paquetes con un peso total de: 5724
Se ha aumentado la capacidad a 58 paquetes lo que permite cargar hasta 5800 kilos

3) Situación: un Contenedor con capacidad para 50 paquetes podría cargar 5000 kilos y tenemos 50 paquetes con un peso total de: 4713
No ha sido necesario aumentar la capacidad.

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00923C

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=58&Itemid=180