Programación Interactiva
Utilidades y arreglos Dinámicos

Escuela de Ingeniería de Sistemas y Computación
Facultad de Ingeniería
Universidad del Valle

Utilidades Gráficas

- Se entiende por utilidades o funcionalidades gráficas, las opciones de presentar en un programa de Java, dibujos planos, en 2D o 3D.
- Las utilidades gráficas se encuentran en el paquete java.awt.*; principalmente en las clases Graphics y Graphics 2D.
- Existe un API que se llama Java 3D con un sinnúmero de utilidades gráficas de alto nivel.

Consideraciones

- Las funcionalidades gráficas operan sobre contenedores de propósito general como JPanel, Canvas, JApplet, Applet y sobre el área de Trabajo del Jframe (panel Contenedor)
- Se realizan a través de un objeto gráfico (objeto de la clase Graphics) y su método **paint(),** propio de éstos contenedores.

Clase canvas

- Es un contenedor de propósito general, parecido al panel, pero que sólo cumple funciones de dibujo. Este componente es utilizado como área para mostrar operaciones de dibujo.
- Constructor:
  - **public Canvas();**

Método Paint

- El método paint() es el encargado de realizar funciones gráficas como dibujar figuras geométricas, textos, imágenes, figuras en 2D y 3D, en lo contenedores de propósito general.
- Sintaxis:
  - **public void paint(Graphics g)**
- Donde g es un objeto de la clase Graphics de awt, que actúa como lienzo (área donde se dibuja y que es invisible)

Método Paint

- Este método se ejecuta de primero en el objeto, osea que no se requiere invocarlo.
- Sin embargo, si se requiere invocar a éste método para realizar una operación de “repintar” o “redibujar” o “actualizar” algún cambio, se debe utilizar cualquiera de los siguientes métodos:
  - repaint(); // encargado de incovar a paint()
  - update(); // encargado de proveer memoria de “lo pintado”

Clase Graphics

- Pertenece al paquete java.awt
- Los objetos de ésta clase, sirven como parámetros de los métodos paint de los contenedores de propósito general.
- Los objetos de este tipo, actuán como “lienzos” o áreas de trabajo, en las cuales se realizan todas las operaciones de dibujo que se quieran mostrar.
- La clase Graphics no se instancia, en su defecto el método paint() al ejecutarse captura el objeto con el cual se trabaja.

Clase Graphics - Métodos

- drawOval
- fillOval
- drawRect
- fillRect
- drawPolygon
- fillPolygon
- drawLine
- drawString
- getColor
- repaint
- update

Clase Graphics - Métodos

setColor:

- Este método asinga color al lienzo o área de trabajo del contenedor. Con este color se realizan cualquiera de las operaciones gráficas.
- *Sintaxis:*
  - g.setColor(color);
- Donde color es la variable que contiene el color a aplicar al lienzo.
- Ejm: Color.red, Color.blue,etc.

Clase Graphics - Métodos

drawOval

- Dibuja el contorno de un círculo o elipse, donde coordx y coordy forman el punto de inicio para pintar el círculo, ancho es la longitud de la base y alto la longitud de la altura del rectángulo donde esta inscrito el círculo.Todos son de tipo entero.
- *Sintaxis:*
  - g.drawOval(coordx,coordy,ancho,alto);

Clase Graphics - Métodos

fillOval

- Dibuja un círculo o elipse relleno del color definido por setColor, donde coordx y coordy forman el punto de inicio para pintar el círculo, ancho es la longitud de la base y alto la longitud de la altura del rectángulo donde esta inscrito el círculo.Todos son de tipo entero.
- *Sintaxis:*
  - g.fillOval(coordx,coordy,ancho,alto);

Clase Graphics - Métodos

drawRect

- Dibuja el contorno de un cuadrado o rectángulo, donde coordx y coordy forman el punto de inicio para pintar la figura, ancho es la longitud de la base y alto la longitud de la altura del rectángulo donde esta inscrito el círculo. *coordx, coordy, ancho y alto son de tipo int*
- *Sintaxis:*
  - *drawRect(coordx,coordy,ancho, alto)*

Clase Graphics - Métodos

fillRect

- Dibuja *un cuadrado o rectángulo relleno del color definido por setColor(), donde coordx y coordy forman el punto de inicio para pintar la figura, ancho es la longitud de la base y alto la longitud de la altura del rectángulo donde esta inscrito el círculo.*
- *Sintaxis:*
  - *fillRect(coordx,coordy,ancho, alto)*

Clase Graphics - Métodos

drawLine

- *Este método permite dibujar una línea, en donde coordx1 y coordy1 forman el punto inicial de la recta y coordx2 y coordy2 forman el punto final de ella.*
- La línea se dibuja del color definido por setColor al contexto gráfico o lienzo.
- *Sintaxis:*
  - *drawLine(coordx1,coordy1,coordx2,coordy2)*

Utilidades Gráficas

- Aquí voy

Clase Graphics – Métodos

drawString

- Este método escribe una cadena en el contexto gráfico (lienzo), la cual está representada por ***cadenaAPintar*** y ***coordx*** y ***coordy*** forman el punto inicial a partir del cual se pinta la cadena.

Sintaxis:

drawString(cadenaAPintar,coordx,coordy)

Clase Graphics – Métodos

drawPolygon

- Este método dibuja el contorno de un polígono de numLados, y cuyas coordenadas X de sus vértices están almacenados en el arreglo de enteros arregloCoordX y las coordenayas Y de sus vértices están en el arregloCoordY de tipo int.

Sintaxis:

drawPolygon(arregloCoordX,arregloCoordY, numLados);

Clase Graphics – Métodos

fillPolygon

- Este método dibuja un polígono relleno del color definido por setColor, de numLados, y cuyas coordenadas X de sus vértices están almacenados en el arreglo de enteros arregloCoordX y las coordenayas Y de sus vértices están en el arregloCoordY de tipo int.

Sintaxis:

fillPolygon(arregloCoordX,arregloCoordY, numLados)

Clase Graphics – Métodos

update

- Este método hace un llamado al componente para su actualización. Esto quiere decir que éste método en algunos casos sustituye a repaint y como él, tiene la función de llamar a paint(). A diferencia de repaint(), update conserva las operaciones que se han hecho en el lienzo o contexto gráfico.

Sintaxis:

update(Graphics g)

Estructuras de Datos Dinámicas en Java: La Clase Vector

Las Estructuras de Datos

- Las estructuras de datos son mecanismos que permiten agrupar en una sola variable u objeto, un grupo de variables u objetos que tienen ciertas características.
- Las estructuras de datos permiten almacenar en un solo espacio valores (de manera global) que permiten que el programa tenga conocimiento o “memoria” de los datos que se están utilizando en cualquier parte del código.

Las Estructuras de Datos

- Se trabjan con archivos, cuando se quiere hacer que los datos sean persistentes.
- En Java, también se les denomina estructura de colecciones
- Las clases que implementan las diversas estructuras de datos que existen y sus variaciones, se encuentran en el paquete java.util

Las Estructuras de Datos

Ejemplos:

- Vector
- Properties
- Enumeration
- HashTable
- Stack
- Set
- List
- Array
- Dictionary
- Collection
- SortedSet
- LinkedList

La Clase Vector

- La clase Vector permite crear un arreglo dinámico que almacena objetos.
- En un objeto de tipo vector se almacenan objetos únicamente (es decir, variables de tipos de datos no primitivos o de clases bases).
- Para almacenar valores de tipo de datos primitivos, se debe utilizar un arreglo estático o hacer casting de los valores a su correspondiente representación de objetos.

La Clase Vector

- Esta clase tiene muchos métodos implementados que facilitan el trabajo con arreglos: búsquedas, eliminación, inserción, expansión entre otras.
- Un objeto de tipo vector, se puede redimensionar en tiempo de ejecución, si las celdas que definen se han ocupado.
- Para ello, existe un atributo o propiedad que le indica al compilar, en cuántas posiciones
- se puede redimensionar el vector, cuando ya haya ocupado su capacidad máxima.

La Clase Vector: Contructores

Vector objeto = new Vector();

- Se crea un objeto de tipo vector con capacidad por defecto de 10 posiciones.

Vector objeto = new Vector(tam);

- Se crea un objeto de tipo vector con capacidad definida por la variable tamaño, que debe ser int.

La Clase Vector: Contructores

Vector objeto = new Vector(int tam,int incr);

Se crea un objeto de tipo vector con capacidad definida por la variable *tam* y con posibilidad de incrementarse el número de posiciones dadas por la variable *incr* cuando se hayan ocupado todas sus posiciones.

En primer y segundo constructor, cuando se llene el vector, Java automáticamente lo redimensiona aumentando el número de posiciones de su tamaño original, así no se haya hecho explícita la orden de incremento.

La Clase Vector: Adición de Objetos

- Todos los elementos que se almacenan en un vector son de tipo Object.
- Esto implica que hay que hacer uso del proceso de casting, para guardar y/o recuperar los objetos.
- Recordar que, la superclase de todas las clases es Object, por lo tanto cualquier objeto de una clase, ya sea una definida por Java o una clase base, es por defecto un objeto de la clase Object.

La Clase Vector: Atributos

- int capacityIncrement: este atributo permite conocer el número de posiciones en las cuales se puede incrementar el vector cuando se haya ocupado su capacidad inicial.
- int elementCount: permite devolver el número de elementos almacenados actualmente en el vector.

La Clase Vector

addElement

- Adiciona el elemento obj al final del vector, incrementando el número de elementos en 1. Se incrementa la capacidad del vector, en caso que éste esté totalmente ocupado.
- Parámetros: obj -el elemento a adicionarse

Sintaxis:

public void addElement(Object obj)

La Clase Vector

elementAt

- Devuelve el elemento en la posición dada por el entero index.
- Parámetros: index -una posición dentro del vector
- Retorna: el componente en dicha posición

Sintaxis:

public Object elementAt(int index)

La Clase Vector

firstElement

- Devuelve el primer elemento del vector (es decir, el elemento en la posición 0).
- Devuleve: el primer elemento del vector

Sintaxis:

```
public Object firstElement()
```

La Clase Vector

indexOf

- Busca la primera aparición del elemento elem en el vector.
- Parámetros: elem - un elemento
- Retorna: la posición de la primera aparición de dicho elemento en el vector. Si el elemento no está en el vector, devuelve -1.

Sintaxis:

```
public int indexOf(Object elem)
```

La Clase Vector

indexOf

- Busca la primera aparición del elemento elem en el vector, a partir de la posición pos.
- Parámetros: elem - un elemento y pos, un entero que indica a partir de cuál posición se empieza a buscar en el vector.
- Retorna: la posición de la primera aparición de dicho elemento en el vector. Si el elemento no está en el vector, devuelve -1.

Sintaxis:

```
public int indexOf(Object elem, int pos)
```

La Clase Vector

insertElementAt

- Inserta el objeto obj en la posición pos del vector. La posición debe ser un valor mayor o igual a 0 o menor que el tamaño actual del vector.
- Parámetros: obj -el elemento a adicionarse pos - la posición en el vector donde se insertará el elemento.

Sintaxis:

```
public void insertElementAt(Object obj, int
                                        pos)
```

La Clase Vector

lastElement

- Devuleve el último elemento del vector
- Retorna: el objeto que ocupa la última posición del vector.

Sintaxis:

```
public Object lastElement()
```

La Clase Vector

lastIndexOf

- Busca la última aparición del elemento elem.
- Parámetros: elem - un elemento
- Retorna: la posición de la última aparición de dicho elemento en el vector. Si el elemento no está en el vector, devuelve -1.

Sintaxis:

```
public int lastIndexOf(Object elem)
```

La Clase Vector

lastIndexOf()

- Busca la última aparición del elemento elem, a partir de la posición pos.
- Parámetros: elem - un elemento pos - la posición a partir de la cual empieza a buscar
- Retorna: la posición de la última aparición de dicho elemento en el vector. Si el elemento no está en el vector, devuelve -1.

Sintaxis:

```
public int lastIndexOf(Object elem, int pos)
```

La Clase Vector

removeAllElements

- Elimina todos los elementos del vector y asigna a éste tamaño 0.

Sintaxis:

```
public void removeAllElements()
```

La Clase Vector

removeElementAt

- Elimina el elemento en la posición pos. El tamaño del vector se reduce en 1. La variable pos debe ser mayor o igual a 0 y menor que el actual tamaño del vector.
- Parámetros: pos - la posición del elemento a eliminar

Sintaxis:

```
public void removeElementAt(int pos)
```

La Clase Vector

setElementAt

- Asigna el elemento obj a la posición pos del vector. Si había algún elemento en dicha posición, éste es eliminado. La posición debe ser un valor mayor o igual que 0 y menor que el tamaño actual del vector.
- Parámetros: obj - el elemento a ser adicionado, pos - la posición a ser modificada.

Sintaxis:

```
public void setElementAt(Object obj, int pos)
```

La Clase Vector

setSize

- Asigna un tamaño a este vector, dado por nuevoTam. Si nuevoTam es mayor que el tamaño actual del vector, éste de agranda y pone elementos nulos en las posiciones nuevas. Si el nuevo tamaño es menor, las posiciones siguientes a nuevoTam son eliminadas.
- Parámetros: nuevoTam- el nuevo tamaño del vector

Sintaxis:

```
public void setSize(int nuevoTam)
```

La Clase Vector

size

- Devuelve el número de elementos actualmente almacenados en el vector.
- Retorna: un entero con el número de elementos actualmente almacenados en el vector.

Sintaxis:

```
public int size()
```