Programación de Computadores: Java - Arreglos uni-dimensionales, multi-dimensionales y listas (y un intro a Git)

Juan F. Pérez

Departamento MACC
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

juanferna.perez@urosario.edu.co

Segundo Semestre de 2017

Contenidos

Un intro a Git

Git

- Sistema de control de versiones

Git

- Sistema de control de versiones
- Controlar cambios en código (archivos)

Git

- Sistema de control de versiones
- Controlar cambios en código (archivos)
- ¿Quién hizo qué cambio en qué línea cuándo?

Git

- Sistema de control de versiones
- Controlar cambios en código (archivos)
- ¿Quién hizo qué cambio en qué línea cuándo?
- Se mantiene historial y se pueden reversar cambios de ser necesarios

Git

- Sistema de control de versiones
- Controlar cambios en código (archivos)
- ¿Quién hizo qué cambio en qué línea cuándo?
- Se mantiene historial y se pueden reversar cambios de ser necesarios
- Necesario en cualquier proyecto que involucre desarrollar software

Git

- Sistema de control de versiones
- Controlar cambios en código (archivos)
- ¿Quién hizo qué cambio en qué línea cuándo?
- Se mantiene historial y se pueden reversar cambios de ser necesarios
- Necesario en cualquier proyecto que involucre desarrollar software
- Especialmente en equipos

Git

- Sistema de control de versiones
- Controlar cambios en código (archivos)
- ¿Quién hizo qué cambio en qué línea cuándo?
- Se mantiene historial y se pueden reversar cambios de ser necesarios
- Necesario en cualquier proyecto que involucre desarrollar software
- Especialmente en equipos
- Historial en repositorio

GitHub

- `https://github.com/`

GitHub

- `https://github.com/`
- Servicio para mantener repositorios Git

GitHub

- `https://github.com/`
- Servicio para mantener repositorios Git
- Creemos una cuenta

GitHub

- `https://github.com/`
- Servicio para mantener repositorios Git
- Creemos una cuenta
- Creemos un repositorio

SourceTree

- `https://www.sourcetreeapp.com/`

SourceTree

- `https://www.sourcetreeapp.com/`
- Aplicación cliente para sincronizar repositorios remotos (GitHub) con repositorios locales

SourceTree

- `https://www.sourcetreeapp.com/`
- Aplicación cliente para sincronizar repositorios remotos (GitHub) con repositorios locales
- Abramos SourceTree

SourceTree

- `https://www.sourcetreeapp.com/`
- Aplicación cliente para sincronizar repositorios remotos (GitHub) con repositorios locales
- Abramos SourceTree
- Clonemos el repositorio que acabamos de crear en GitHub

Operación

- Clonar repositorio (1 vez)

Operación

- Clonar repositorio (1 vez)
- Halar cambios en el repositorio (pull)

Operación

- Clonar repositorio (1 vez)
- Halar cambios en el repositorio (pull)
- Hacer modificaciones localmente

Operación

- Clonar repositorio (1 vez)
- Halar cambios en el repositorio (pull)
- Hacer modificaciones localmente
- Enviar los cambios al repositorio local (commit)

Operación

- Clonar repositorio (1 vez)
- Halar cambios en el repositorio (pull)
- Hacer modificaciones localmente
- Enviar los cambios al repositorio local (commit)
  - Comentarios claros!

Operación

- Clonar repositorio (1 vez)
- Halar cambios en el repositorio (pull)
- Hacer modificaciones localmente
- Enviar los cambios al repositorio local (commit)
  - Comentarios claros!
- Enviar los cambios al repositorio remoto (push)

Operación

- Clonar repositorio (1 vez)
- Halar cambios en el repositorio (pull)
- Hacer modificaciones localmente
- Enviar los cambios al repositorio local (commit)
  - Comentarios claros!
- Enviar los cambios al repositorio remoto (push)
- Repite

Operación

- Usemos el repositorio creado para subir un proyecto sencillo

Operación

- Usemos el repositorio creado para subir un proyecto sencillo
- Cambios + commit + push

Operación

- Usemos el repositorio creado para subir un proyecto sencillo
- Cambios + commit + push
- Permitamos que alguien más se una al repositorio y hale los cambios (pull)

Operación

- Usemos el repositorio creado para subir un proyecto sencillo
- Cambios + commit + push
- Permitamos que alguien más se una al repositorio y hale los cambios (pull)
- Ahora el otro hace los cambios (commit + push)

Operación

- Usemos el repositorio creado para subir un proyecto sencillo
- Cambios + commit + push
- Permitamos que alguien más se una al repositorio y hale los cambios (pull)
- Ahora el otro hace los cambios (commit + push)
- Halamos los cambios localmente (pull)

Arreglos

Arreglos

- Almacenar muchos elementos de información uniforme

Arreglos

- Almacenar muchos elementos de información uniforme
- Información ordenada

Arreglos

- Almacenar muchos elementos de información uniforme
- Información ordenada
- Homogénea

Arreglos

- Almacenar muchos elementos de información uniforme
- Información ordenada
- Homogénea
- Arreglos de datos primitivos (int, double, char, boolean)

Arreglos

- Almacenar muchos elementos de información uniforme
- Información ordenada
- Homogénea
- Arreglos de datos primitivos (int, double, char, boolean)
- Arreglos de objetos

Arreglos

Definición

```
int[] miArreglo = new int[10];
```

Inicialización por defecto (0 a números, false a boolean, null a objetos)

Arreglos

Acceso a valores (modificación):

```
miArreglo[0] = 1;
miArreglo[5] = 3;
```

Acceso a valores (lectura):

```
println(miArreglo[0]);
println(miArreglo[5]);
```

Arreglos y ciclos

```
package arreglos;
import acm.program.*;

public class Arreglos extends ConsoleProgram{
 public void run(){
  double[] miArreglo = new double[10];
  for(int i = 0; i < miArreglo.length; i++){
   miArreglo[i] = Math.pow(2, i);
  }
  for(int i = 0; i < miArreglo.length; i++){
   println(miArreglo[i]);
  }
 }
}
```

Arreglos de Objetos

```
package arreglos;
import acm.program.*;

public class ArreglosV2 extends ConsoleProgram{
 public void run(){
  Documento[] miArreglo = new Documento[10];
  miArreglo[0] = new Documento("Mi libro", "Mi Autor", 1900)
    ;
  }
}
```

Representación de arreglos

Arreglos se representan

- Como objetos

Representación de arreglos

Arreglos se representan

- Como objetos
- Se pasan referencias a métodos

Representación de arreglos

Arreglos se representan

- Como objetos
- Se pasan referencias a métodos
- Referencia almacenada en el stack

Representación de arreglos

Arreglos se representan

- Como objetos
- Se pasan referencias a métodos
- Referencia almacenada en el stack
- Elementos del arreglo en el heap

Representación de arreglos: Ejemplo

Clase `InvertirArreglos`

```
public void run(){
 int[] arreglo = new int[10];
 leerArreglo(arreglo);
 invertirArreglo(arreglo);
 imprimirArreglo(arreglo);
}
```

Representación de arreglos: Ejemplo

Clase `InvertirArreglos`

```
public void leerArreglo(int[] arreglo){
 for(int i = 0; i < arreglo.length; i++){
  arreglo[i] = readInt("?");
 }
}
```

Representación de arreglos: Ejemplo

Clase `InvertirArreglos`

```
public void imprimirArreglo(int[] arreglo){
 for(int i = 0; i < arreglo.length; i++){
  println(arreglo[i]);
 }
}
```

Representación de arreglos: Ejemplo

Clase `InvertirArreglos`

```
public void invertirArreglo(int[] arreglo){
 for(int i = 0; i < arreglo.length/2; i++){
  invertirElementos(arreglo, i, arreglo.length - i - 1);
 }
}

public void invertirElementos(int[] arreglo, int p1, int p2)
    {
 int temp = arreglo[p1];
 arreglo[p1] = arreglo[p2];
 arreglo[p2] = temp;
}
```

Inicializando Arreglos

Asignar un valor inicial a los elementos del arreglo:

```
int[] valor = {200, 500, 100, 600};

double[] medidas = {8.3, 2.4, 9.2, 5.8};
```

Inicializando Arreglos

```
String[] facultades = {
 "Ciencias y Matematicas",
 "Economia",
 "Ciencias Humanas"
};
```

```
Documento[] miArreglo = {
 new Documento("Matrix Computations", "C. Van Loan", 1997),
 new Documento(),
 new Documento("Can quantum-mechanical description of
    physical reality be considered complete?", "A. Einstein,
    B. Podolsky, N. Rosen", 1935)
};
```

Arreglos Multidimensionales

Arreglos Multidimensionales

- Considere arreglos en los que cada elemento es en sí mismo un arreglo

Arreglos Multidimensionales

- Considere arreglos en los que cada elemento es en sí mismo un arreglo
- Necesitamos entonces dos índices para referirnos a los elementos: uno para el elemento del primer arreglo (externo), y otro para el elemento del arreglo interno

Arreglos Multidimensionales

- Considere arreglos en los que cada elemento es en sí mismo un arreglo
- Necesitamos entonces dos índices para referirnos a los elementos: uno para el elemento del primer arreglo (externo), y otro para el elemento del arreglo interno
- Considere el tablero de ajedrez

Arreglos Multidimensionales

- Considere arreglos en los que cada elemento es en sí mismo un arreglo
- Necesitamos entonces dos índices para referirnos a los elementos: uno para el elemento del primer arreglo (externo), y otro para el elemento del arreglo interno
- Considere el tablero de ajedrez
- Necesitamos dos índices: uno para las filas y otro para las columnas

Arreglos Multidimensionales

- Considere arreglos en los que cada elemento es en sí mismo un arreglo
- Necesitamos entonces dos índices para referirnos a los elementos: uno para el elemento del primer arreglo (externo), y otro para el elemento del arreglo interno
- Considere el tablero de ajedrez
- Necesitamos dos índices: uno para las filas y otro para las columnas
- Considere una imagen

Arreglos Multidimensionales

- Considere arreglos en los que cada elemento es en sí mismo un arreglo
- Necesitamos entonces dos índices para referirnos a los elementos: uno para el elemento del primer arreglo (externo), y otro para el elemento del arreglo interno
- Considere el tablero de ajedrez
- Necesitamos dos índices: uno para las filas y otro para las columnas
- Considere una imagen
- Necesitamos dos índices para referirnos a las dos coordenadas

Arreglos Multidimensionales

- Considere arreglos en los que cada elemento es en sí mismo un arreglo
- Necesitamos entonces dos índices para referirnos a los elementos: uno para el elemento del primer arreglo (externo), y otro para el elemento del arreglo interno
- Considere el tablero de ajedrez
- Necesitamos dos índices: uno para las filas y otro para las columnas
- Considere una imagen
- Necesitamos dos índices para referirnos a las dos coordenadas
- Arreglo bidimensional: matriz

Creando Arreglos Bidimensionales

```
int[][] matriz = new int[5][5];

String[][] tablero = new String[8][8];
```

Accediendo Arreglos bidimensionales

Posición superior izquierda:

```
String a = tablero[0][0];
```

Posición superior derecha:

```
String b = tablero[0][7];
```

Posición inferior izquierda:

```
String c = tablero[7][0];
```

Posición inferior derecha:

```
String b = tablero[7][7];
```

Inicializando Arreglos bidimensionales

```
int[][] tablero = {
  {1  0  1},
  {0  1  1},
  {1  0  1}
};
```

Ejemplo: Imágenes como arreglos bidimensionales

```
public class ImagenArreglo extends GraphicsProgram{

 GImage im;

 public static double FACTOR_ESCALA = 0.5;

 public void init(){
  im = new GImage("../data/foto.jpeg");
  im.scale(FACTOR_ESCALA);
  add(im);
  add(new JButton("Invertir Imagen"), NORTH);
  addActionListeners();
 }
```

Ejemplo: Imágenes como arreglos bidimensionales

```
public void actionPerformed(ActionEvent e){
 if(e.getActionCommand().equals("Invertir␣Imagen")){
  int[][] pixeles = im.getPixelArray();
  invertirImagenVerticalPix(pixeles);
  im = new GImage(pixeles);
  im.scale(FACTOR_ESCALA);
  add(im);
 }
}
```

Ejemplo: Imágenes como arreglos bidimensionales

```
private void invertirImagenVerticalPix(int[][] pixeles){
 int altura = pixeles.length;
 for(int p1=0; p1 < altura/2; p1++){
  int p2 = altura - p1 - 1;
  int[] temp = pixeles[p1];
  pixeles[p1] = pixeles[p2];
  pixeles[p2] = temp;
 }
}

}
```

Listas en Java: ArrayList

Listas en Java: ArrayList

- Más versátiles que los arreglos

Listas en Java: ArrayList

- Más versátiles que los arreglos
- Dinámicas: permiten agregar y eliminar elementos

Listas en Java: ArrayList

- Más versátiles que los arreglos
- Dinámicas: permiten agregar y eliminar elementos
- Hay varios tipos pero por ahora nos fijaremos en `ArrayList`

Listas en Java: ArrayList

- Más versátiles que los arreglos
- Dinámicas: permiten agregar y eliminar elementos
- Hay varios tipos pero por ahora nos fijaremos en `ArrayList`
- Paquete `java.util`

Listas en Java: ArrayList

- Más versátiles que los arreglos
- Dinámicas: permiten agregar y eliminar elementos
- Hay varios tipos pero por ahora nos fijaremos en `ArrayList`
- Paquete `java.util`
- Parte del `Collections Framework`

Listas en Java: ArrayList

- Almacenan objetos (no datos primitivos) de una misma clase

Listas en Java: ArrayList

- Almacenan objetos (no datos primitivos) de una misma clase
- Generics

Listas en Java: ArrayList

- Almacenan objetos (no datos primitivos) de una misma clase
- `Generics`
- Al crear la lista se debe especificar el tipo de elementos que va almacenar

Listas en Java: ArrayList

- Almacenan objetos (no datos primitivos) de una misma clase
- Generics
- Al crear la lista se debe especificar el tipo de elementos que va almacenar

```
ArrayList<String> miLista = new ArrayList<String>();
```

Ejemplo: clase EjemploArrayLists

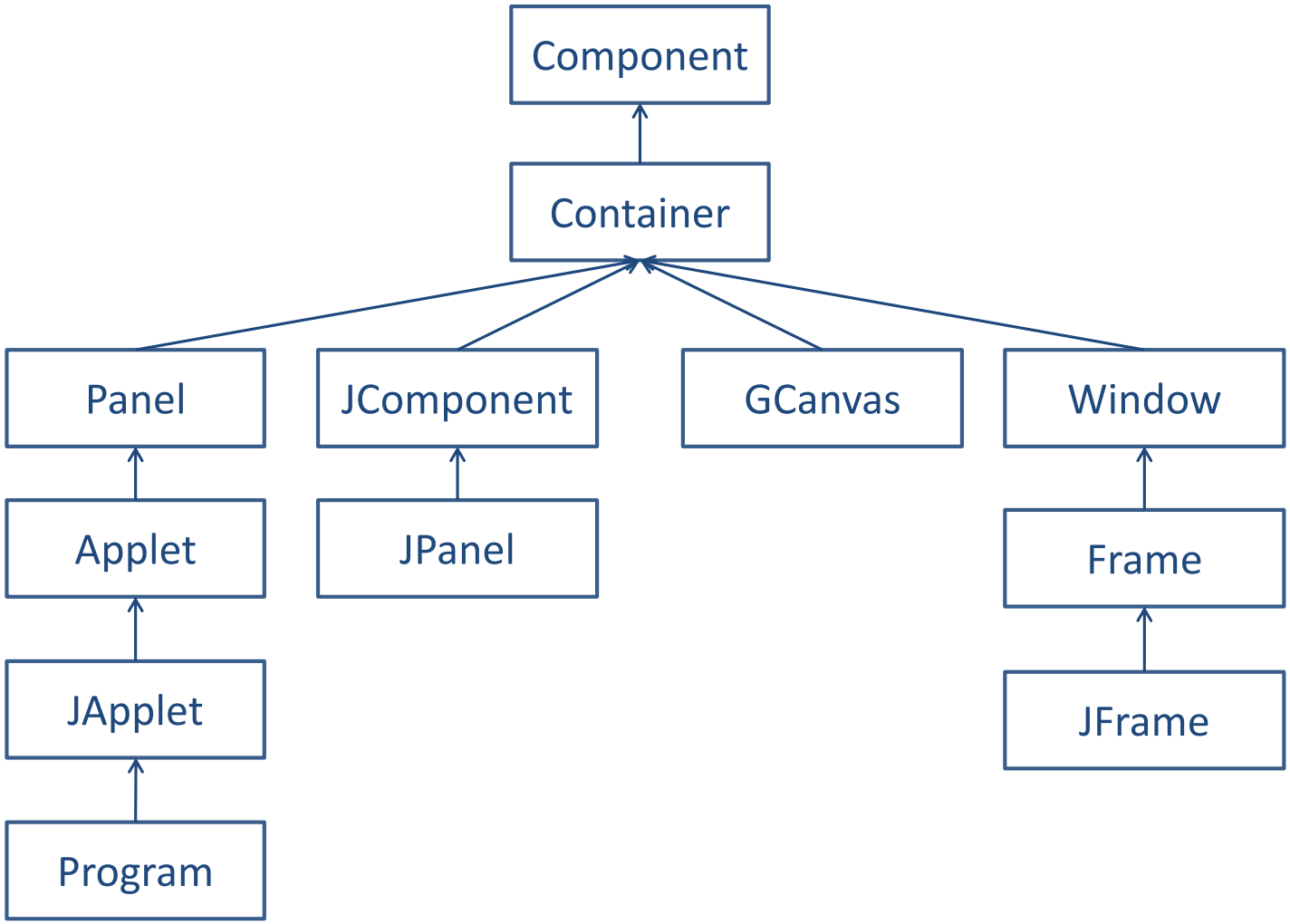
```
public void run(){
 String frase = "esta␣es␣una␣frase␣completa␣con␣varias␣
    palabras";
 StringTokenizer str = new StringTokenizer(frase);
 ArrayList<String> miLista = new ArrayList<String>();
 while(str.hasMoreTokens()){
  miLista.add(str.nextToken());
 }
 imprimirLista(miLista);

 miLista.add("FINAL");
 miLista.add(1, "NO");
 miLista.remove(5);
 imprimirLista(miLista);
 }
```

Ejemplo: clase EjemploArrayLists

```
private void imprimirLista(ArrayList<String> lista){
 for(int i = 0; i < lista.size(); i++){
  println(lista.get(i));
 }
 println("");
}
```

Layouts en GUIs

Jerarquía de Clases de Ventanas en Java



Layouts en GUIs

- Container: un Component que contiene múltiples Components

Layouts en GUIs

- Container: un Component que contiene múltiples Components
- Organización de los Components: Layout manager

Layouts en GUIs

- Container: un Component que contiene múltiples Components
- Organización de los Components: Layout manager
- Varias opciones

Layouts en GUIs

- Container: un Component que contiene múltiples Components
- Organización de los Components: Layout manager
- Varias opciones
- BorderLayout

Layouts en GUIs

- Container: un Component que contiene múltiples Components
- Organización de los Components: Layout manager
- Varias opciones
- BorderLayout
- FlowLayout

Layouts en GUIs

- Container: un Component que contiene múltiples Components
- Organización de los Components: Layout manager
- Varias opciones
- BorderLayout
- FlowLayout
- TableLayout

BorderLayout

Ubicaciones de componentes:

- North

BorderLayout

Ubicaciones de componentes:

- North
- South

BorderLayout

Ubicaciones de componentes:

- North
- South
- West

BorderLayout

Ubicaciones de componentes:

- North
- South
- West
- East

BorderLayout

Ubicaciones de componentes:

- North
- South
- West
- East
- Center

BorderLayout

Ubicaciones de componentes:

- North
- South
- West
- East
- Center
- Prelación a North/South, luego West/East, y luego Center

BorderLayout Ejemplo

```
public class EjemploBorderLayout extends Program{
 public void init(){
  JPanel panel = new JPanel();
  panel.setLayout(new BorderLayout());
  panel.add(new JButton("Norte"),BorderLayout.NORTH);
  panel.add(new JButton("Oeste"),BorderLayout.WEST);
  panel.add(new JButton("Sur"),BorderLayout.SOUTH);
  panel.add(new JButton("Este"),BorderLayout.EAST);
  panel.add(new JButton("Centro"),BorderLayout.CENTER);
  add(panel);
 }
}
```

FlowLayout

- Componentes se adicionan como en una lista

FlowLayout

- Componentes se adicionan como en una lista
- Se muestran tantas por fila como lo permita el tamaño de la ventana

FlowLayout Ejemplo

```
public class EjemploFlowLayout extends Program{
 public void init(){
  JPanel panel = new JPanel();
  panel.setLayout(new FlowLayout());
  panel.add(new JButton("Botón␣1"));
  panel.add(new JButton("Botón␣2"));
  panel.add(new JButton("Botón␣3"));
  panel.add(new JButton("Botón␣4"));
  panel.add(new JButton("Botón␣5"));
  panel.add(new JButton("Botón␣6"));
  add(panel);
 }
}
```

GridLayout

- Componentes se ordenan en una grilla (grid)

GridLayout

- Componentes se ordenan en una grilla (grid)
- Número de filas

GridLayout

- Componentes se ordenan en una grilla (grid)
- Número de filas
- Número de columnas

GridLayout

- Componentes se ordenan en una grilla (grid)
- Número de filas
- Número de columnas
- Componentes cubren todo el espacio disponible

GridLayout Ejemplo

```
public class EjemploGridLayout extends Program{
 public void init(){
  JPanel panel = new JPanel();
  panel.setLayout(new GridLayout(2,3));
  panel.add(new JButton("Botón␣1"));
  panel.add(new JButton("Botón␣2"));
  panel.add(new JButton("Botón␣3"));
  panel.add(new JButton("Botón␣4"));
  panel.add(new JButton("Botón␣5"));
  panel.add(new JButton("Botón␣6"));
  add(panel);
 }
}
```

TableLayout

- Componentes se ordenan en una grilla (grid)

TableLayout

- Componentes se ordenan en una grilla (grid)
- Número de filas

TableLayout

- Componentes se ordenan en una grilla (grid)
- Número de filas
- Número de columnas

TableLayout

- Componentes se ordenan en una grilla (grid)
- Número de filas
- Número de columnas
- Componentes toman su tamaño preferido

TableLayout Ejemplo

```
public class EjemploTableLayout extends Program{
 public void init(){
  JPanel panel = new JPanel();
  panel.setLayout(new TableLayout(2,3));
  panel.add(new JButton("Botón 1"));
  panel.add(new JButton("Botón 2"));
  panel.add(new JButton("Botón 3"));
  panel.add(new JButton("Botón 4"));
  panel.add(new JButton("Botón 5"));
  panel.add(new JButton("Botón 6"));
  add(panel);
 }
}
```

Otro Ejemplo de TableLayout

- `EjemploTableLayoutV2`

Otro Ejemplo de TableLayout

- `EjemploTableLayoutV2`
- Dos `JLabel`

Otro Ejemplo de TableLayout

- `EjemploTableLayoutV2`
- Dos `JLabel`
- Dos `JField`

Otro Ejemplo de TableLayout

- `EjemploTableLayoutV2`
- Dos `JLabel`
- Dos `JField`
- Dos `JButton`