Programación 2

Lenguaje Java - Conceptos claves

Profesor: Eduardo Godoy
eduardo.gl@gmail.coml
Material elaborado por Rodrigo Olivares
rodrigo.olivares@uv.cl

17 de agosto de 2017

Contenido

Comportamiento Principal

Constructor

- En programación orientada a objetos (POO), un **constructor** es una método cuya misión es inicializar un objeto de una clase. En el constructor se asignan los valores iniciales del nuevo objeto.
- Debe tener el **mismo nombre de la Clase**.
- En Java, el constructor es un método *especial* dentro de una clase, que se llama automáticamente cada vez que se instancia un objeto de esa clase.
- Una clase puede no tener un constructor. En este caso, el compilador de Java crea uno vacío.
- Una clase puede tener más de un constructor. Éstos se diferencian por los parámetros que recibe (*sobrecarga de método*).

Comportamiento Principal

Bloques

- Inicialización: Se ejecutan cuando una una clase es instanciada. Incluso antes del constructor.
- Estáticos: A diferencia de los de Inicialización, se ejecutan cuando una una clase es cargada.

Comportamiento Principal

Constructor

```
public class Auto {

        private String color;
        private String marca;
        private boolean estado;

        public Auto() {
        }

        public Auto(String color) {
                this.color = color;
        }

        public Auto(String color, String marca) {
                this.color = color;
                this.marca = marca;
        }
}
```

Lectura de datos

BufferedReader/InputStreamReader

Nos apoyamos en la clase System

```
public class AutoImpl {

        public static void main(String[ ] args) {
                String texto;
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                do {
                        System.out.println("Ingreso algun texto");
                        texto = br.readLine();
                        System.out.println(texto);
                } while(!sTexto.equals("FIN"));
        }
}
```

Lectura de datos

Scanner

Nos apoyamos en la clase System

```
public class AutoImpl {

        public static void main(String[ ] args) {
                String texto;
                Scanner s = new Scanner(System.in);
                do {
                        System.out.println("Ingreso algun texto");
                        texto = s.nextLine();
                        System.out.println(texto);
                } while(!sTexto.equals("FIN"));
        }
}
```

Lectura de datos

Scanner

Algunos métodos de la clase Scanner

- *nextBoolean*(): Scans the next token of the input into a boolean value and returns that value.
- *nextByte*(): Scans the next token of the input as a byte.
- *nextDouble*(): Scans the next token of the input as a double.
- *nextFloat*(): Scans the next token of the input as a float.
- *nextInt*(): Scans the next token of the input as an int.
- *nextLine*(): Advances this scanner past the current line and returns the input that was skipped.
- *nextLong*(): Scans the next token of the input as a long.
- *nextShort*(): Scans the next token of the input as a short.

...

Revisar API *http://docs.oracle.com/javase/6/docs/api/*

Argumentos Principales

Lectura de argumentos

¿Cómo podemos leer una lista de argumentos ingresados al momento de ejecutar el código?

```
public class AutoImpl {

        public static void main(String[ ] args) {
                for(int i = 0: i < args.length; i++) {
                        System.out.println(args[i]);
                }
        }
}
```

$ *java* Auto Chevrolet Camaro amarillo 24000000

Almacenamiento Múltiples

Arreglos

Los arreglos son una forma de almacenar una lista de elementos. Cada espacio/celda del arreglo guarda un elemento individual. Los arreglos pueden tener cualquier tipo de dato (primitivos u objetos), pero **no puede almacenar distintos tipos en un mismo arreglo**.

Para crear un arreglo se debe:

- Declarar una variable para guardar el arreglo.
- Crear/instanciar un nuevo objeto de arreglo y asignarlo a la variable de arreglo.
- Almacenar los valores en el arreglo.

Almacenamiento Múltiples

Arreglos

Declaración

Las variables de arreglo indican el tipo de objeto que el arreglo contendrá y el nombre del arreglo. Los corchetes vacíos pueden incluirse después del tipo de dato o después del nombre del arreglo, indistintamente.

String *palabras[]*;
int *vertices[]*;
Personas *personas[]*;
String[] *palabras*;
int[] *vertices*;
Personas[] *personas*;

Almacenamiento Múltiples

Arreglos

Instanciación

Usar **new**:

Por ejemplo, **String[]** *nombre* = **new String[***10***]**;

Esta linea crea un arreglo de String con 10 espacios/celdas. **Siempre** que se instancie un arreglo con **new** se debe indicar de que tamaño será (número de celdas).

El arreglo se inicializará con :

- 0 para arreglos numéricos.
- **false** para booleanos.
- $'/0'$ para arreglos de caracter.
- **null** para objetos.

Almacenamiento Múltiples

Arreglos

Instanciación

Por extensión:

Por ejemplo, **String[]** *nombre* = {*"pedro"*, *"rodrigo"*, *"carlo"*, *"andres"*};

Instanciación

Cada elemento dentro de las llaves debe ser del mismo tipo y coincidir con el tipo de variable que contiene el arreglo

Almacenamiento Múltiples

Arreglos

Acceso al dato

Para acceder a un elemento del arreglo se usan los sub-índices:

```
String[] texto = new String[10];
texto[9] = "test";
int largoTexto = texto.length;
```

La última línea de código permite ver la longitud del arreglo. Este atributo está disponible para todos los objetos arreglo sin importar el tipo. En java no es posible asignar un valor, a una celda fuera del límite de éste. Los subindices se inician en 0.

Almacenamiento Múltiples

Arreglos

Modificación del dato

Para asignar elementos a una celda del arreglo:

```
array[1] = 7;
promedio[9] = 9,7;
texto[4] = "test";
cadenas[10] = cadenas[1];
```

Al igual que con los objetos, un arreglo de objetos en Java consiste en un arreglo de referencias a dichos objetos. Cuando asigna un valor a una celda en un arreglo, crea un referencia a ese objeto. Cuando desplaza valores dentro de un arreglo sólo se reasigna la referencia, **no** se copia el valor de una casilla a otra, a dfirencia de los arreglos de tipos primitivos que **si** copian los valores de una celda a otra.

Almacenamiento Múltiples

Arreglos

Arreglos multidimensionales

Java no soporta los arreglos multidimensionales. Sin embargo, se pueden declarar un **arreglo de arreglos** y acceder a él de la siguiente manera:

```
int[][] coods = new int [12][12];
coords [0][0] = 2;
```

Almacenamiento Múltiples

Arreglos

Ejercicios

- Escriba una clase que construya un arreglo de tamaño variable (ingreso manualmente) y lo llene con números enteros entre 0 y 100 generados aleatoreamente. Cada comportamiento debe ser implementado de forma independiente.
- Recorra el arreglo y sume su contenido.
- Imprima cada valor y la suma total en pantalla.
- Para generar números aleatorios, puede utilizar el método **random** de la clase **Math** que está en el paquete **java.lang** o bien puede utilizar el método **nextInt** de la clase **Random** que se encuentra en el paquete **java.util**.

Almacenamiento Múltiples

Arreglos

Ejercicios

Mostrar las notas de un alumno

- Cree la clase **Estudiante** y declare un arreglo de notas de tipo número y una constante con la cantidad de notas totales del estudiante (NB_NOTAS)
- Cree un método **llenar** que compete el arreglo con notas aleatorias entre 1 y 7.
- Cree un método **mostrar** que imprima en pantalla las notas de un alumno.
- Cree un método **promedio** que calcule el promedio de notas del alumno.
- Cree la clase independiente **EstudianteImpl** y en método principal instancie un objeto del tipo **Estudiante** y llame a los métodos *llenar*, *mostrar* y *promedio*.

Almacenamiento Múltiples

ArrayList

ArrayList

La clase ArrayList ofrece funcionalidades de acceso rápido, comparables a las de un arreglo de objetos. Esta clase es más flexible que los arreglos de objeto, pues su tamaño (cantidad de elementos) puede variar durante la ejecución.

Almacenamiento Múltiples

ArrayList

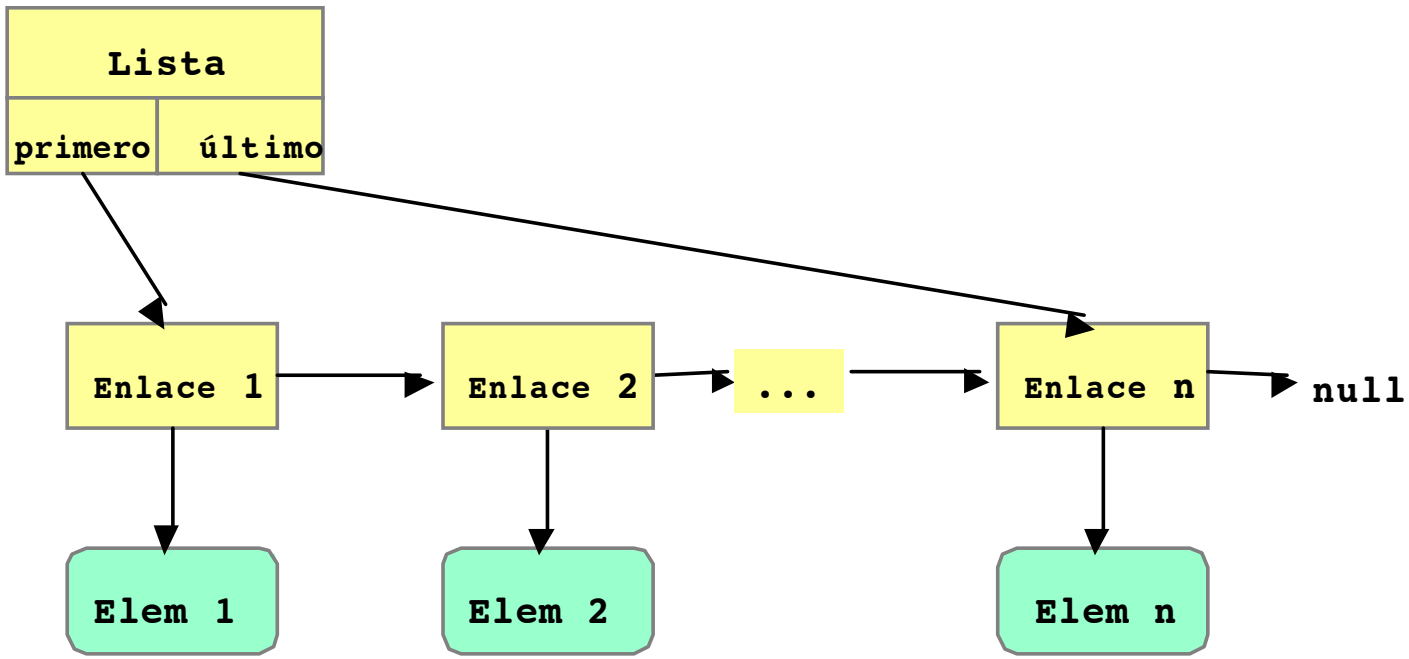
- **Construcción**: Un vector dinámico puede ser construido vacío o a partir de un conjunto de datos.
  - *ArrayList* v1 = **new** *ArrayList*();
  - *ArrayList* v2 = **new** *ArrayList*(c);
- **Agregar un elemento**:
  - Agregar un elemento al final del vector usando *add*(elem);
  - Agregar un elemento en una posición *i* dada *add*(i, elem);
- **Suprimir un elemento**:
  - Suprimir un elemento en la posición *i*, *remove*(i);

    /* El método *remove* retorna el objeto o rango de objetos eliminados (tipo *Object*). Si no elimina el objeto (por que no lo encuentra) retorna *false* */
  - Suprimir un rango consecutivo de elementos, *removeRange*(n,p);
  - Suprimir todo, *removeAll*();

Almacenamiento Múltiples

ArrayList

Esquema gáfico de un ArrayList



Almacenamiento Múltiples

ArrayList

Operadores

- Acceder a los elementos usando *get*(i).

```
public void mostrar(ArrayList v) {

        for (int i = 0; i < v.size(); i++) {
                System.out.println(v.get(i));
        }
}
```

- Modificar los elementos usando *set*(i).

```
public void cambiar(ArrayList v) {

        for (int i = 0; i < v.size(); i++) {
                v.set(i, null);
        }

}
```

Almacenamiento Múltiples

Arreglos

Ejercicios

Escriba un programa que cree un vector dinámico que contenga 10 autos, cada auto posee como atributos: marca, modelo y valor. Se requiere recorrer el arreglo creado con autos y obtener el promedio del valores.

Almacenamiento Múltiples

Arreglos

Ejercicios

Escriba un programa que cree un vector dinámico que contenga 15 objetos de tipo numéricos (aleatorios). Verifique el tamaño del vector (utilizando el método *size()*) e imprima los elementos. Luego, elimine los objetos de la posición 3, 7, 11 y 13. Verifique nuevamente el tamaño del vector. Modifique los valores de la posición 2 y 6 (los nuevos valores deben ser ingresados por la entrada estándar). Muestre los nuevos valores del vector.

Preguntas

Preguntas ?