ICI341 Principios de Lenguajes de Programación

Java: Sobrecarga y Colecciones

Claudio Cubillos

Material original del Prof. Franco Guidi Polanco
Escuela de Ingeniería Industrial-PUCV

Sección 1

Sobrecarga de Métodos y Constructores

Sobrecarga de constructores

- Una clase puede tener más de un constructor.
- Los constructores se diferencian por cantidad, tipo y orden de parámetros.

  Ejemplo: constructores distintos de la clase `Observación`

```
public Observación(){…
public Observación(int a){…
public Observación(int a, double b){…
public Observación(double a, int b){…
```

- Esto permite instanciar objetos considerando distintos tipos de datos disponibles.

Sobrecarga de constructores (cont.)

```
public class Valor {
    private int x;
    private int y;

    public Valor(){
       x = 0;
       y = 0;
    }
    public Valor(int a, int b){
       x = a;
       y = b;
    }
    ...
}
```

Constructor sin parámetros

Constructor con dos parámetros enteros

Sobrecarga de constructores (cont.)

```
public class Valor {
  private int x;
  private int y;

  public Valor(){
     x = 0;
     y = 0;
  }
  public Valor(int a, int b){
     x = a;
     y = b;
  }
  ...
}
```

Clase Valor

```
public class Ejemplo {
 public static void main...

   Valor ob1, ob2;
   ob1 = new Valor( 3, 5 );
   ob2 = new Valor();

   //Las siguientes fallan:
   Valor ob3, ob4;
   ob3 = new Valor( 2 );
   ob4 = new Valor( 2.0, 3.0);
   ...
 }
}
```

Aplicación

No existe el constructor de un parámetro **int**

No existe el constructor de dos parámetros **double**

Sobrecarga de métodos

- Una clase puede tener más de un método con el mismo nombre.
- Los métodos se diferencian por nombre del método, y cantidad, tipo y orden de sus parámetros. Todo esto constituye la "firma del método" (*method signature*).

Ejemplo: métodos distintos de una clase

```
public double sumaTiempo(){…
public double sumaTiempo(int a){…
public double sumaTiempo(double a){…
public double sumaTiempo(int a, double b){…
public double sumaTiempo(double a, int b){…
```


Sobrecarga de métodos

- IMPORTANTE: el tipo de valor retornado **no forma parte** de la "firma del método" (no es utilizado para distinguir entre métodos).

```
public double sumaTiempo(int a){…
public float  sumaTiempo(int b){…
public int    sumaTiempo(int a){…
```

Java no distingue entre ellos: genera un **error de compilación**.

Sobrecarga de métodos (cont.)

```
public class Persona {
  private String nombre;
  private int edad;

  public void setEdad(int x){
     edad = x;
  }
  public void setEdad(double x){
     edad = (int) x;
  }
  ...
}
```

Método **setEdad(int)**

Método **setEdad(double)**

Sobrecarga de métodos (cont.)

```
public class Persona {

  private String nombre;
  private int edad;

  public void setEdad(int x){
     edad = x;
  }

  public void setEdad(double x){
     edad = (int) x;
  }
  ...
}
```

Clase Persona

```
public class Ejemplo {

  public static void main...

   Persona p1;
   p1 = new Persona();
   …

   P1.setEdad( 35 );
   p1.setEdad( 35.0 );

   …
 }
}
```

Aplicación

Sobrecarga de métodos y promoción de argumentos

```
public class Valor {

  private int dato;

  public void setEdad(short x){
     dato = x;
  }

  public void setEdad(int x){
     dato = x;
  }

  public void setEdad(double x){
     edad = (int) x;
  }
  ...
}
```

Clase Valor

```
public class Ejemplo {
 public static void main...

   Valor v = new Valor();
   …

   v.setEdad( 35 );
   v.setEdad( 35.0 );

   // Aquí hay promoción:

   byte b = 3;
   v.setEdad( b );
   v.setEdad( 35f );
   …
 }
}
```

Aplicación

Sección 2

Arreglos de Objetos

Arreglos de objetos

- Se declaran e instancian como los arreglos de tipos primitivos:

```
Clase[] variable = new Clase[entero];
```

- Ejemplo:

```
CuentaAhorro[] ctas = new CuentaAhorro[200];
Tiempo[] reloj = new Tiempo[n] // con n entero
```

Arreglos de objetos (cont.)

- Un arreglo de objetos puede almacenar en cada posición una **referencia a un objeto de la clase** con que fue definido, o también la dirección **null**.
- La instanciación de un arreglo **no instancia** los objetos que puede referenciar.
- Los objetos a referenciar desde un arreglo deben instanciarse **individualmente**.

Arreglos de objetos (cont.)

- Se debe instanciar cada objeto y luego asociarlo al arreglo:

variable = new Constructor(*parámetros* **);**
nombreArreglo[*entero***] = variable;**

o instanciar y asociar inmediatamente:

nombreArreglo[*entero***] = new Constructor(** *parámetros* **);**

- NOTA: también es posible asociar dos o más posiciones a un mismo objeto.

Arreglos de objetos (cont.)

```
public class Valor {
 public int x;
 private int y;
 public Valor(int a, int b){
    x=a;
    y=b;
 }
 public void set_y( int a ){
    y = a;
 }
 public int get_y(){
    return y;
 }
}
```

Clase Valor

```
public class Ejemplo {

 public static void main...

   Valor[] lista;
   lista = new Valor[100];
   int i = 0;
   while(i <100){
      lista[i]=new Valor(i,2*i);
      i++;
   }
 }

}
```

Aplicación

Arreglos de objetos (cont.)

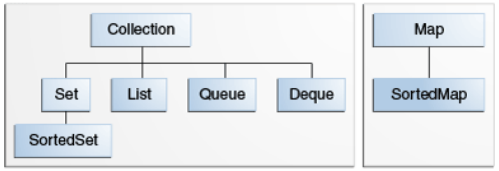
```
public class Ejemplo {
   public static void main(String[] arg) {

     Valor[] lista = new Valor[100];

      // Aquí se llena de objetos con valores
     ...
     // A continuación se suman valores almacenados
     int i = 0;
     int s_x = 0, s_y = 0;
     while(i <100){
        s_x = s_x + lista[i].x;
        s_y = s_y + lista[i].get_y();
        i++;
     }
   }
}
```

Sección 3

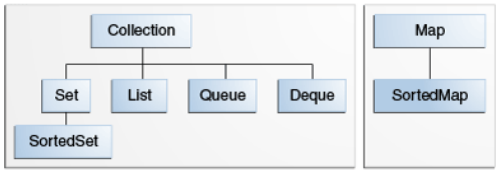
Colecciones en Java

JAVA Collections Framework

- Provee a los desarrolladores con un conjunto de clases e interfaces que facilitan la tarea de manejar colecciones de objetos.
- Las colecciones trabajan con un conjunto de elementos de manera similar a estructuras más básicas como arreglos o listas enlazadas,
- la diferencia es que sus operaciones son de alto nivel y encapsulan los detalles de implementación.
- La mayoría de las colecciones de Java se encuentran en el paquete java.util.



JAVA Collections Framework



- Collection - la raíz de la jerarquía de colecciones. Una colección representa un grupo de objetos conocidos como sus elementos.
- Set - una colección que no puede contener elementos duplicados. Es la abstracción matemática de un conjunto.
- Lista - una colección ordenada (a veces llamado una secuencia). Las listas pueden contener elementos duplicados. El usuario de una lista generalmente tiene un control preciso sobre donde en la lista cada elemento se inserta y puede acceder a los elementos por su índice (posición).
- Mapa - un objeto que mapea claves a valores. Un mapa no puede contener claves duplicadas; a cada clave se le puede asignar a lo más a un valor.

ArrayList

public class **ArrayList<E>**

- Es la implementación de una lista (permite elementos repetidos) mediante arreglos.
- es des-sincronizado(los cambios realizados por múltiples hilos no son reflejados).

Métodos

boolean add(E e) - Appends the specified element to the end of this list.

void add(int index, E element) - Inserts the specified element at the specified position in this list.

void clear() - Removes all of the elements from this list.

boolean contains(Object o) - Returns true if this list contains the specified element.

E get(int index) - Returns the element at the specified position in this list.

int indexOf(Object o) - Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.

boolean isEmpty() - Returns true if this list contains no elements.

E remove(int index) - Removes the element at the specified position in this list.

boolean remove(Object o) - Removes the first occurrence of the specified element from this list, if it is present.

E set(int index, E element) - Replaces the element at the specified position in this list with the specified element.

int size() - Returns the number of elements in this list.

Vector

public class **Vector<E>**

- Es la implementación de una lista (permite elementos repetidos) mediante arreglos.
- es sincronizado (los cambios realizados por múltiples hilos son reflejados).

Métodos

boolean add(E e) - Appends the specified element to the end of this Vector.

void add(int index, E element) - Inserts the specified element at the specified position in this Vector.

void addElement(E obj) - Adds the specified component to the end of this vector, increasing its size by one.

int capacity() - Returns the current capacity of this vector.

void clear() - Removes all of the elements from this Vector.

boolean contains(Object o) - Returns true if this Vector contains the specified element.

E firstElement() - Returns the first component (the item at index 0) of this vector.

E get(int index) - Returns the element at the specified position in this Vector.

int indexOf(Object o) - Returns the index of the first occurrence of the specified element in this Vector, or -1 if this Vector does not contain the element.

void insertElementAt(E obj, int index) - Inserts the specified object as a component in this vector at the specified index.

boolean isEmpty() - Returns true if this Vector contains no elements.

E lastElement() - Returns the last component of the vector.

E remove(int index) - Removes the element at the specified position in this Vector.

boolean remove(Object o) - Removes the first occurrence of the specified element from this Vector, if it is present.

E set(int index, E element) - Replaces the element at the specified position in this Vector with the specified element.

int size() - Returns the number of elements in this Vector.

Vector

- Vector es prácticamente igual a ArrayList, su única diferencia es que Arraylist es des-sincronizado(los cambios realizados por múltiples hilos no son reflejados).

```
Vector vec = new Vector(20,5);/*crea un vector con capacity 20 y
capacityincrement 5 */
vec.addElement("uno");/*agrega el objeto “uno” en la posición final*/
vec.insertElementAt("tres", 1);/*inserta “tres” en la posición 1*/
System.out.println(vec.size());/*imprime el numero de elementos,en este caso
2*/
System.out.println(vec.capacity()); /*imprime la capacidad, en este caso 20*/
vec.removeElement("tres"); /*remueve el elemento “tres” contenido en el
vector*/
vec.removeElementAt(0); /* remueve el elemento en la posicion 0*/
```


Stack

- Se extiende de la clase Vector agregando las funciones para ser operada como Stack.
- Permite guardar objetos almacenándolos en forma de pila tipo LIFO(Last In First Out).

```
Stack pila= new Stack();
pila.push("UNO"); /*agrega a la pila "UNO"*/
pila.push("DOS"); /*agrega a la pila "DOS"*/
pila.pop();/*obtiene el ultimo objeto agregado y lo elimina, en este caso
"DOS"*/
pila.peek(); /*obtiene el ultimo objeto agregado, en este caso "UNO"*/
```


HashSet

public class **HashSet<E>**

- Implementa la interfaz Set mediante una hash table (una instancia de HashMap).
- Permite el elemento null.
- No entrega garantía en cuanto al orden de iteración al interior del set;
  - No garantiza que el orden permanecerá constante en el tiempo.

Métodos

HashSet() - Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).

HashSet(int initialCapacity) - Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).

boolean add(E e) - Adds the specified element to this set if it is not already present.

void clear() - Removes all of the elements from this set.

Object clone() - Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.

boolean contains(Object o) - Returns true if this set contains the specified element.

boolean isEmpty() - Returns true if this set contains no elements.

Iterator<E> iterator() - Returns an iterator over the elements in this set.

boolean remove(Object o) - Removes the specified element from this set if it is present.

int size() - Returns the number of elements in this set (its cardinality).

HashTable

public class **Hashtable<K,V>**

- Implementa una hash table, la cual almacena pares clave-valor (key-value pairs).
- Cualquier objeto no null puede ser utilizado como clave o valor
- Los objetos utilizados como claves deben implementar el método hashCode y el método equals.

Métodos

Hashtable() - Constructs a new, empty hashtable with a default initial capacity (11) and load factor (0.75).

void clear() - Clears this hashtable so that it contains no keys.

boolean contains(Object value) - Tests if some key maps into the specified value in this hashtable.

boolean containsKey(Object key) - Tests if the specified object is a key in this hashtable.

boolean containsValue(Object value) - Returns true if this hashtable maps one or more keys to this value.

Enumeration<V> elements() - Returns an enumeration of the values in this hashtable.

boolean equals(Object o) - Compares the specified Object with this Map for equality, as per the definition in the Map interface.

V get(Object key) - Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

int hashCode() - Returns the hash code value for this Map as per the definition in the Map interface.

boolean isEmpty() - Tests if this hashtable maps no keys to values.

Enumeration<K> keys() - Returns an enumeration of the keys in this hashtable.

V put(K key, V value) - Maps the specified key to the specified value in this hashtable.

protected void rehash() - Increases the capacity of and internally reorganizes this hashtable, in order to accommodate and access its entries more efficiently.

V remove(Object key) - Removes the key (and its corresponding value) from this hashtable.

int size() - Returns the number of keys in this hashtable.

HashTable

- Remplaza a los diccionarios ya que cumplen la misma función y de manera más eficiente.
- Las hashtable son comúnmente usadas para almacenar datos de usuarios y contraseñas debido a su rápido acceso y codificación(hashcode).

```
Hashtable numbers = new Hashtable();
numbers.put("uno", new Integer(1)); /*agrega el valor 1 con la clave "uno"*/
numbers.put("dos", new Integer(2)); /*agrega el valor 2 con la clave "dos"*/
numbers.put("queso", new Integer(3)); /*agrega el valor 3 con la clave "queso"*/
System.out.println(numbers.get("uno")); /*obtiene el valor con la clave "uno",
en este caso 1*/
numbers.remove("uno"); /*remueve el objeto con la clave "uno"*/
```