Tema 0

Repaso colecciones y Patrones de Diseño

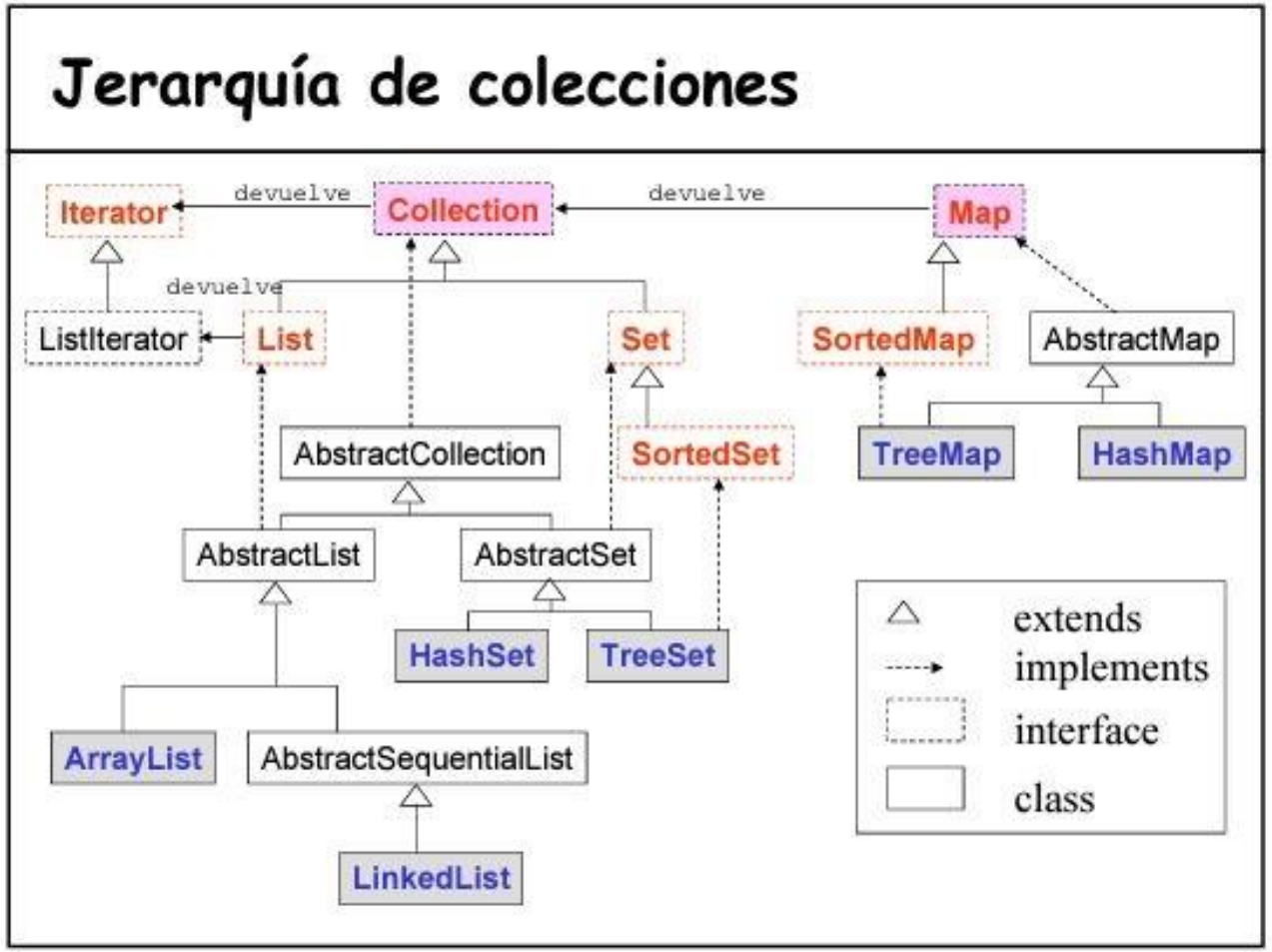
Colecciones

- Interfaz Collection
  - Clases:
    - Listas : ArrayList, LinkedList
    - Set : HashSet y TreeSet
- Interfaz Map
  - Clases
    - TreeMap, HashMap, HashTable
- Iteradores:
  - Interfaz Iterator

Colecciones

- Permiten almacenar y organizar objetos de manera útil para un acceso eficiente
  - Dimensionamiento dinámico
  - Operaciones de acceso O(n) como máximo
- Se encuentran dentro del paquete java.útil
- Las interfaces proporcionan métodos para todas las operaciones comunes
- Las implementaciones concretas especifican las operaciones
- Sobre todas las colecciones podemos recorrerlas mediante los iteradores

Colecciones



Colecciones

- Collection<T>
  - int size()
  - boolean empty()
  - boolean contains(Object elem)
  - Iterator<T> iterator()
  - Object[] toArray
  - boolean add(T elem)
  - boolean remove(Object elem)
  - void clear()

Colecciones

- List<T>
  - Los elementos tienen asignado “orden”
  - void add(int index, T element)
  - T remove(int index)
  - T get(int index)
  - T set(int index, T element)
  - int indexOf(Object o)
  - int lastIndexOf(Object o)
  - List<T> subList(int min, int max)

Colecciones

- Set<T>
  - No puede haber elementos repetidos
  - Los elementos se almacenan “sin orden”
  - Implementan los métodos del interfaz Collection
    - int size()
    - boolean empty()
    - boolean contains(Object elem)
    - Iterator<T> iterator()
    - Object[] toArray
    - boolean add(T elem)
    - boolean remove(Object elem)
    - void clear()

Colecciones

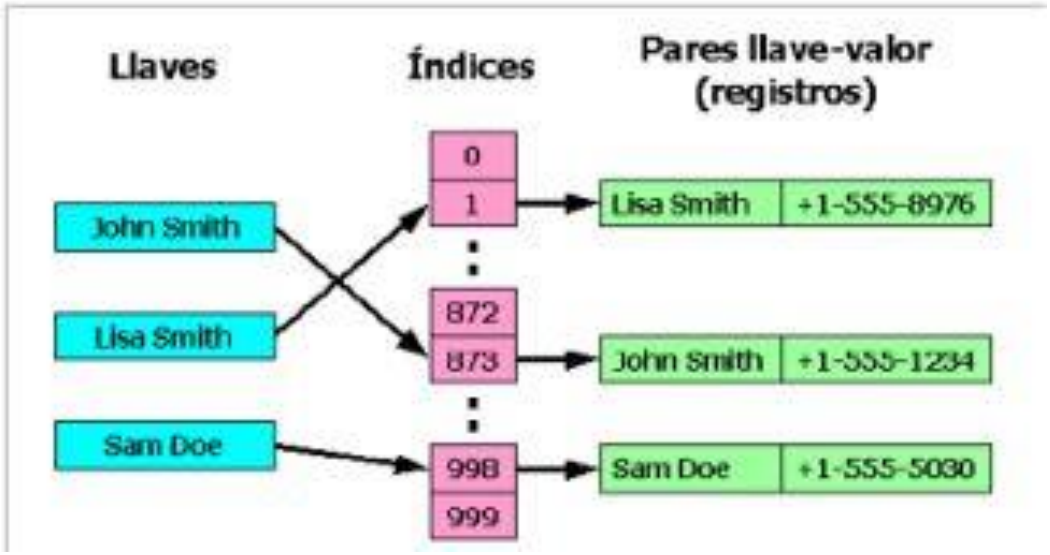
- Map<K,V>
  - Cada objetivo “valor” tiene asignado una clave
  - No puede haber claves duplicadas
  - Object put(Kkey, V value)
  - V remove(Object key)
  - V get(Object key)
  - containsKey, containsValue, isEmpty, size
  - Set<K> keySet() //Colección de claves
  - Collection<V> values() //Colección de valores
  - Set<Map.Entry<K,V>> entrySet() //Colección de pares clave-valor

Entrada = <clave,valor>

- Map.entry
  - Clase que almacena pares <clave,valor>
  - Clase interna java.util.map
  - K getKey() = proporciona la clave del par
  - V getValue() = proporciona el valor del par
  - V setValue(V nuevoValor) = cambiar el campo valor y retorna el valor antiguo

Tablas Hash /Diccionarios

- Estructura de datos formada por pares <clave-valor>
- Soporta de manera eficiente las búsquedas
- Operación de búsqueda tiene complejidad O(1)



Tablas Hash/Diccionario

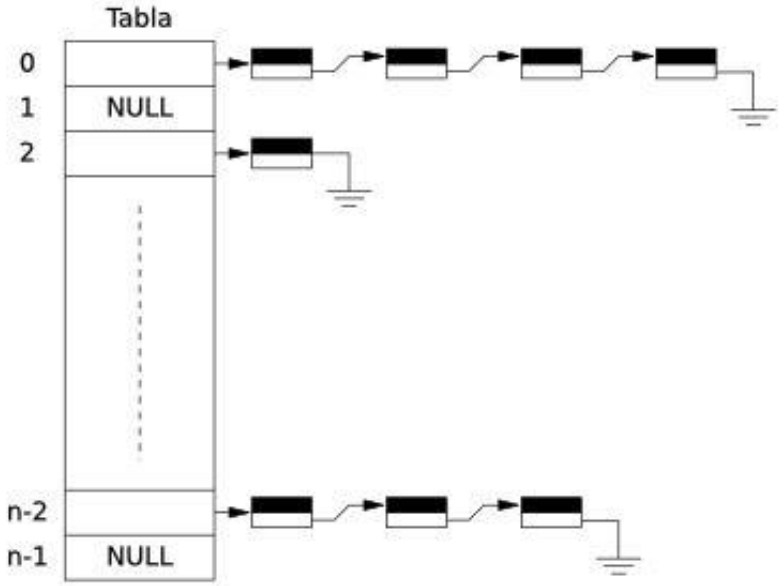
- Operaciones fundamentales
  - Insercción(clave-valor);
  - Busqueda (clave-valor)
- Operaciones secundarias
  - Borrar(clave)
  - Iteración sobre lso elementos
- Elementos:
  - Estructura de acceso directo (Array)
  - Estructura de datos para el valor con una clave
  - Función hash
    - Retorna para la clave un integer

Tablas Hash/Diccionario

- Operación insertar
  - Transformamos la clave en un entero : **función resumen**
    - Trasforma la clave en un valor entero
  - Funcion Hash
    - Hash(x) = X % tamaño de la tabla
    - Conseguimos la posición dentro de la tabla
  - El elemento valor se almacena dentro de la posición obtenida
    - Si la posición esta ocupada => colisiones
      - Se almacena en una lista (tabla hash abierta)
      - Se redimensiona la tabla (tabla hash cerrada)

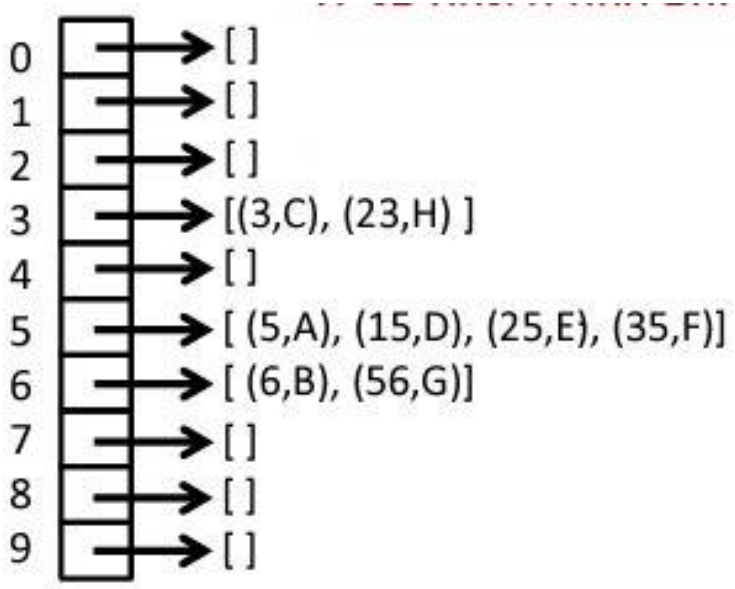
Tablas Hash

- Operación insertar
  - Tabla hash abierta



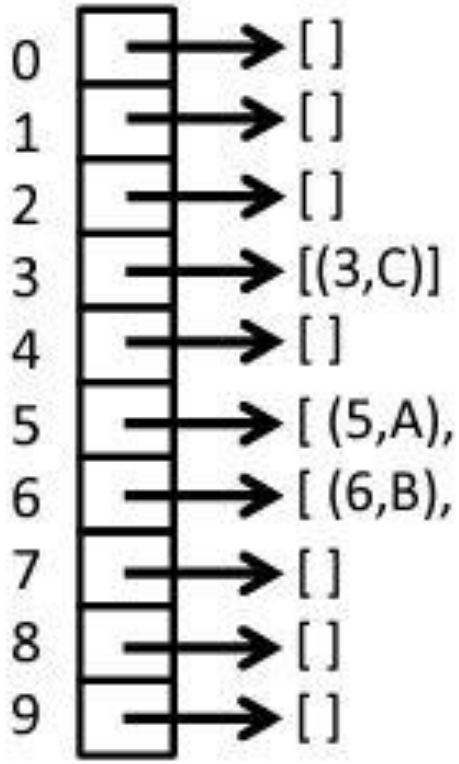
Tablas Hash

- Operación insertar
  - Tabla hash abierta función hash(x) X%10 (10 es el tamaño)



- Operación insertar
  - Tabla hash cerrada

```
M.put( 5, 'A' )
M.put( 6, 'B' )
M.put( 3, 'C' )
M.put( 15, 'D' )
M.put( 25, 'E' )
M.put( 35, 'F' )
M.put( 56, 'G' )
M.put( 23, 'H' )
```



Métodos

Método	Descripción
get(clave)	Obtiene el valor correspondiente a una clave. Devuelve null si no existe esa clave en el diccionario.
put(clave, valor)	Añade un par (clave, valor) al diccionario. Si ya había un valor para esa clave, se machaca.
keySet()	Devuelve un conjunto (set) con todas las claves.
values()	Devuelve una colección con todos los valores (los valores pueden estar duplicados a diferencia de las claves).
entrySet()	Devuelve una colección con todos los pares (clave, valor).
containsKey(clave)	Devuelve true si el diccionario contiene la clave indicada y false en caso contrario.
getKey()	Devuelve la clave de la entrada. Se aplica a una sola entrada del diccionario (no al diccionario completo), es decir a una pareja (clave, valor).
getValue()	Devuelve el contenido de la entrada. Se aplica a una entrada del diccionario (no al diccionario completo), es decir a una pareja (clave, valor).

Definición de una hashMap

- Cuando declaramos un diccionario, debemos declarar el tipo de clave y valor

```
HashMap<Integer, String> m = new HashMap<Integer, String>();
```

  - Definimos un diccionario con key de tipo Integer y String para el valor

- Debemos importar la clase

```
import java.util.HashMap;
```

HashMap

- put
  - Para insertar una entrada usamos el método **put** indicando la clave y el valor

```java
import java.util.HashMap;
public class EjemploHashMap01 {
    public static void main(String[] args) {
        HashMap<Integer, String> m = new HashMap<Integer, String>();

        m.put(924, "Amalia Núñez");
        m.put(921, "Cindy Nero");
        m.put(700, "César Vázquez");
        m.put(219, "Víctor Tilla");
        m.put(537, "Alan Brito");
        m.put(605, "Esteban Quito ");

        System.out.println("Los elementos de m son: \n" + m);
    }
}
```

```
Los elementos de m son:
{921=Cindy Nero, 537=Alan Brito, 219=Víctor Tilla, 924=Amalia Núñez, 700=César Vázquez, 605=Esteban Quito }
```

HashMap

- get
  - Para Para extraer los datos utilizamos el método **get().**
  - Se proporciona la clave
    - El diccionario retorna el valor
    - Si no existe valor retorna null

```
Cindy Nero
Esteban Quito
null
```

```java
import java.util.HashMap;
public class EjemploHashMap011 {
    public static void main(String[] args) {
        HashMap<Integer, String> m = new HashMap<Integer, String>();

        m.put(924, "Amalia Núñez");
        m.put(921, "Cindy Nero");
        m.put(700, "César Vázquez");
        m.put(219, "Víctor Tilla");
        m.put(537, "Alan Brito");
        m.put(605, "Esteban Quito ");

        System.out.println(m.get(921));
        System.out.println(m.get(605));
        System.out.println(m.get(888));
    }
}
```

HashMap

- entrySet
  - Set /HashSet
    - Colección de datos que **no mantiene orden**
    - No admite datos duplicados
    - Su interacción se realiza mediante el bucle for-each
      - For(tipo-dato nombre-vble : conjuntoSet)

```
Set<String> mySet = new HashSet<>();
mySet.add("Apple");
mySet.add("Banana");
mySet.add("Mango");

for(String s : mySet){
    System.out.println("s is : "+s);
}
```

HashMap

- entrySet
  - Obtenemos un conjunto con todas las entradas forma <clave-valor>
    - Map.Entry => es la clave par <clave-valor>
      - Map.Entry par
      - par.getKey()
      - par.getValue()

HashMap

- entrySet
  - Obtenemos un conjunto con todas las entradas

```
import java.util.HashMap;
import java.util.Map;
public class EjemploHashMap02 {
    public static void main(String[] args) {
        HashMap<Integer, String> m = new HashMap<Integer, String>();

        m.put(924, "Amalia Núñez");
        m.put(921, "Cindy Nero");
        m.put(700, "César Vázquez");
        m.put(219, "Víctor Tilla");
        m.put(537, "Alan Brito");
        m.put(605, "Esteban Quito ");

        System.out.println("Todas las entradas del diccionario extraídas con entrySet:");
        System.out.println(m.entrySet());

        System.out.println("\nEntradas del diccionario extraídas una a una:");
        for (Map.Entry pareja: m.entrySet()) {
            System.out.println(pareja);
        }
    }
}
```

HashMap

- entrySet

```
Todas las entradas del diccionario extraídas con entrySet:
[921=Cindy Nero, 537=Alan Brito, 219=Víctor Tilla, 924=Amalia Núñez, 700=César Vázquez, 605=Esteban Quito ]

Entradas del diccionario extraídas una a una:
921=Cindy Nero
537=Alan Brito
219=Víctor Tilla
924=Amalia Núñez
700=César Vázquez
605=Esteban Quito
```

HashMap

- getKey/getValue

```
import java.util.*;
public class EjemploHashMap03 {
    public static void main(String[] args) {
        HashMap<Integer, String> m = new HashMap<Integer, String>();

        m.put(924, "Amalia Núñez");
        m.put(921, "Cindy Nero");
        m.put(700, "César Vázquez");
        m.put(219, "Víctor Tilla");
        m.put(537, "Alan Brito");
        m.put(605, "Esteban Quito ");

        System.out.println("Código\tNombre\n------\t-------------");
        for (Map.Entry pareja: m.entrySet()) {
            System.out.print(pareja.getKey() + "\t");
            System.out.println(pareja.getValue());
        }
    }
}
```

HashMap

- getKey/getValue

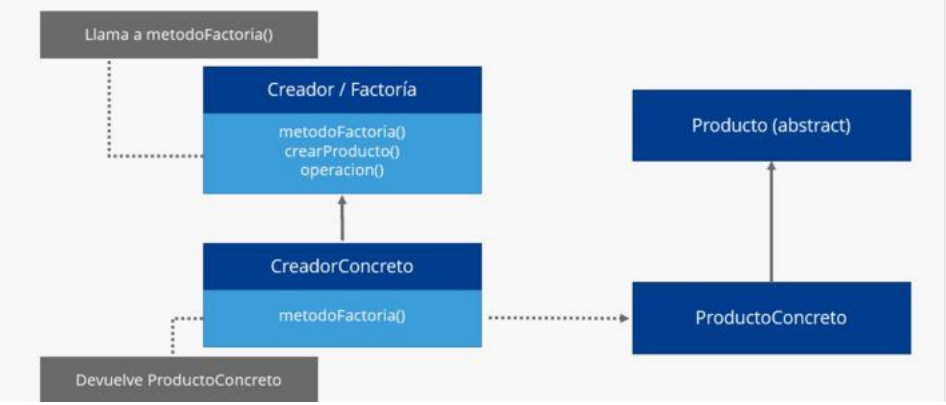
```
Código  Nombre
------  -------------
921     Cindy Nero
537     Alan Brito
219     Víctor Tilla
924     Amalia Núñez
700     César Vázquez
605     Esteban Quito
```

HashMap

- containsKey
  - Determina si una clave existe o no
  - Si existe podemos utilizar el método get

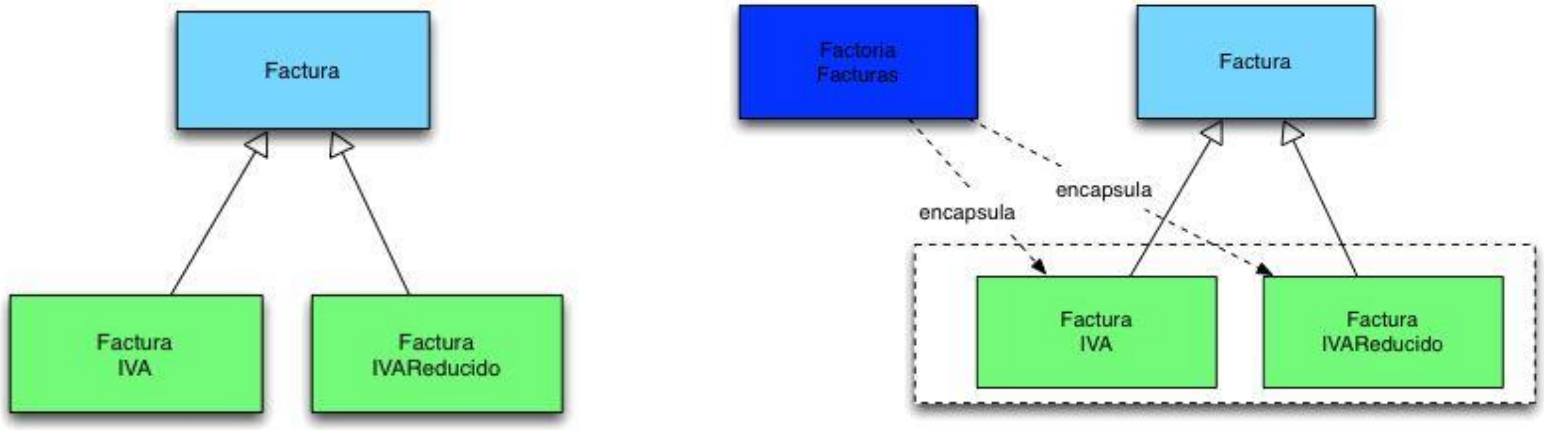
```
public class EjemploHashMap04 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HashMap<Integer, String> m = new HashMap<Integer, String>();

        m.put(924, "Amalia Núñez");
        m.put(921, "Cindy Nero");
        m.put(700, "César Vázquez");
        m.put(219, "Víctor Tilla");
        m.put(537, "Alan Brito");
        m.put(605, "Esteban Quito ");

        System.out.print("Por favor, introduzca un código: ");

        int codigoIntroducido = sc.nextInt();
        if (m.containsKey(codigoIntroducido)) {
            System.out.print("El código " + codigoIntroducido + " corresponde a ");
            System.out.println(m.get(codigoIntroducido));
        }
        else {
            System.out.print("El código introducido no existe.");
        }

        sc.close();
    }
}
```

- Factoría
  - “una clase fabrica “ que produce objetos
  - Crear objetos sin tener que especificar la clase exacta
    - No hacemos new Contructor()
  - Tener unca clase Interfaz o clase base
    - Tenemos clases derivadas especificas

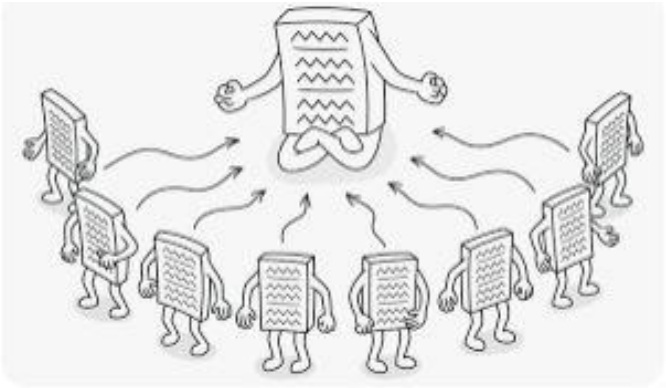


PATRONES DE DISEÑO

- Factoría
  - Ejemplos de uso
    - Sistemas de verificación
    - Cuando los productos concretos no se encuentran definidos
  - Ejemplo:



- Singleton
  - Una clase únicamente permite crear un único objeto
    - Usamos un único objeto para coordinar acciones del sistema
    - Se proporciona acceso global a la instancia de la clase
  - Usado para compartir información en la aplicación
    - Estado de un recurso
    - Control de hilos



- Singleton
  - Implementacion
    - Constructor privado
      - Evitamos que se creen por ejecuciones de terceros
    - Un campo estático que contiene su única clase
      - Referencia al objeto que vamos a crear a treves del constructor
    - Método estático publico
      - Para poder obtener la instancia del objeto
      - Lo instancia la primera vez y lo almacena en la variable estática

- Singleton

```
public class ClaseSingleton {
    private static ClaseSingleton instanciaUnica = new ClaseSingleton();

    private ClaseSingleton() {}

    public static ClaseSingleton getInstance() {
        return instanciaUnica;
    }
}
```

Patrones diseño

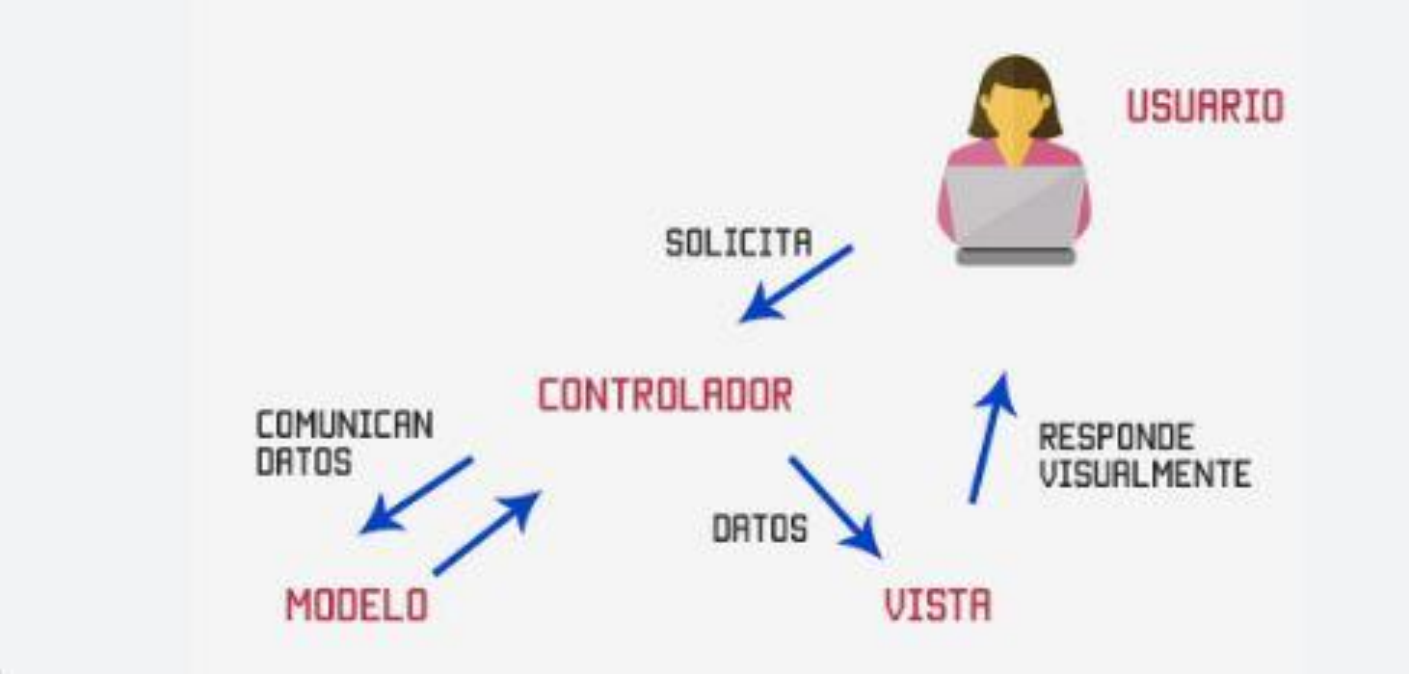
- Vista-controlador
  - “Estilo de arquitectura de software
  - 3 componentes:
    - El modelo :
      - Contiene la representación de los datos
      - Almacena la lógica del negocio
      - Mecanismos de persistencia
    - La vista :
      - Interfaz de usuario
      - Información que se intercambia entre el usuario y la aplicación
    - Controlador:
      - Actúa de intermediario entre el modelo y la vista
      - Gestiona el flujo de información entre ellos

Patrones diseño

- Vista-controlador
  - El modelo es responsable de :
    - Acceder a la capa de persistencia de los datos
    - Define las reglas del negocio (funcionalidad del sistema)
  - El controlador es el responsable de :
    - Recibe eventos de entrada (click, un cambio en un campo de texto etc)
    - Contiene las reglas de gestión de eventos
    - Gestiona las peticiones a la vista o al modelo
  - La vista es responsable de:
    - Recibe datos del modelo y los muestra al usuario
    - Tiene un controlador asociado

Patrones diseño

- Vista-controlador



- Vista-controlador
  - Ejemplo:
    - Queremos implementar una aplicación para la gestión CRUD de datos
    - Tenemos un prototipo de aplicación
      - Las ventanas aún no están definidas
      - La información de la base de datos puede modificarse (inicialmente My Sql)