 POLITÉCNICO DE COLOMBIAeducación sin límites	DIPLOMADO EN PROGRAMACIÓN EN JAVA		
	ACTIVIDAD EVALUATIVA - MÓDULO 4		
	CÓDIGO: No aplica	VERSIÓN: 1	Página 1 de 21

<p align="center">Actividad evaluativa</p> <p align="center">Módulo 4: Estructuras de datos</p>

Descripción de la actividad

A partir de la información suministrada en la guía y con tu investigación complementaria dar respuesta a las siguientes preguntas.

1. Consultar las dos siguientes estructuras de datos: (30 %)

- Java.util.Map

- Java.util.HashMap

¿Para qué sirven? ¿Cuáles son sus métodos principales?

2. ¿Qué son los generics en Java? (15 %)

3. Construya con sus propias palabras un texto (mínimo una [1] página, máximo tres [3]) con base en las estructuras de datos, su definición, principal método y la diferencia que tiene frente a las otras estructuras. (45 %)

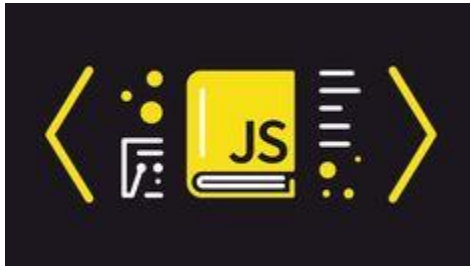
- Listas

- Pilas

- Colas

4. Adicionalmente se tendrá en cuenta las referencias bibliográficas y piezas de código, pantallazos o ilustraciones que permitan mejorar la calidad del entregable. (10 %)

Indice Módulo 4: Estructuras De Datos



Control + Clic Para Seguir El Vínculo Al Título

Contenido

Indice Módulo 4: Estructuras De Datos	2
Consulta De Estructuras de datos	4
<i>Java.util.Map</i>	4
¿Para qué sirve?	4
Características	4
Implementaciones comunes	5
Ejemplo De Aplicación	6
<i>Java.util.HashMap</i>	7
¿Para qué sirve?	7
Características	7
Métodos principales	7
Ejemplo De Aplicación	8
Ventajas	8
Desventajas	9
Alternativas	9
Texto De Estructura De Datos	9
Diferencias	9
¿Qué Son Los Generics En Java?	10
<i>Conceptos Clave de los Generics</i>	10
Ejemplo De Aplicación	10
<i>Uso de Generics en Clases</i>	11
<i>Uso de Generics en Métodos</i>	12
<i>Uso de Generics en Interfaces</i>	14

<i>Bounded Type Parameters (Parámetros de Tipo Acotado)</i>	14
<i>Raw Types</i>	15
<i>Type Erasure</i>	15
<i>Ventajas de Usar Generics</i>	16
Ejemplo De Aplicación	16
Listas, Pilas y Colas	17
<i>Listas</i>	17
Tipos de listas	17
Métodos comunes de Lista	18
<i>Pilas</i>	18
Condicionales	19
Métodos de la clase Stack	19
<i>Colas</i>	19
Clases que implementan Queue.....	20
Métodos de la interfaz Queue.....	20
Referencias Bibliográficas	20
<i>Bibliografía</i>	20

Consulta De Estructuras de datos

Java.util.Map



Java Util Map representa una interfaz existente en el lenguaje de programación Java -que conforma parte del paquete java.util que hace uso de un conjunto de métodos que establecen y manejan datos en clave-valor. A fin de ser más exactos, es una interfaz perteneciente a la biblioteca estándar de Java, que se apoya en la colección de pares clave-valor.

Un par, entonces, es decir, un mapa consiste precisamente en un valor y una clave que no puede ser repetida, por lo que la existencia de estos pares es muy relevante, pues permite a las personas recuperar y manipular información y almacenar y hacer recuperaciones según una clave o ejecutar varios datos en proceso. Un map es en otras palabras, una colección de datos donde hay relación clave y personas que pueden obtener los valores a través de las claves.

¿Para qué sirve?

- ❖ Almacenar y recuperar utilizando una clave única.
- ❖ Realizar búsquedas rápidas basadas en una única clave.
- ❖ Implementar cachés y utilizar el cache para almacenar configuraciones.

Características

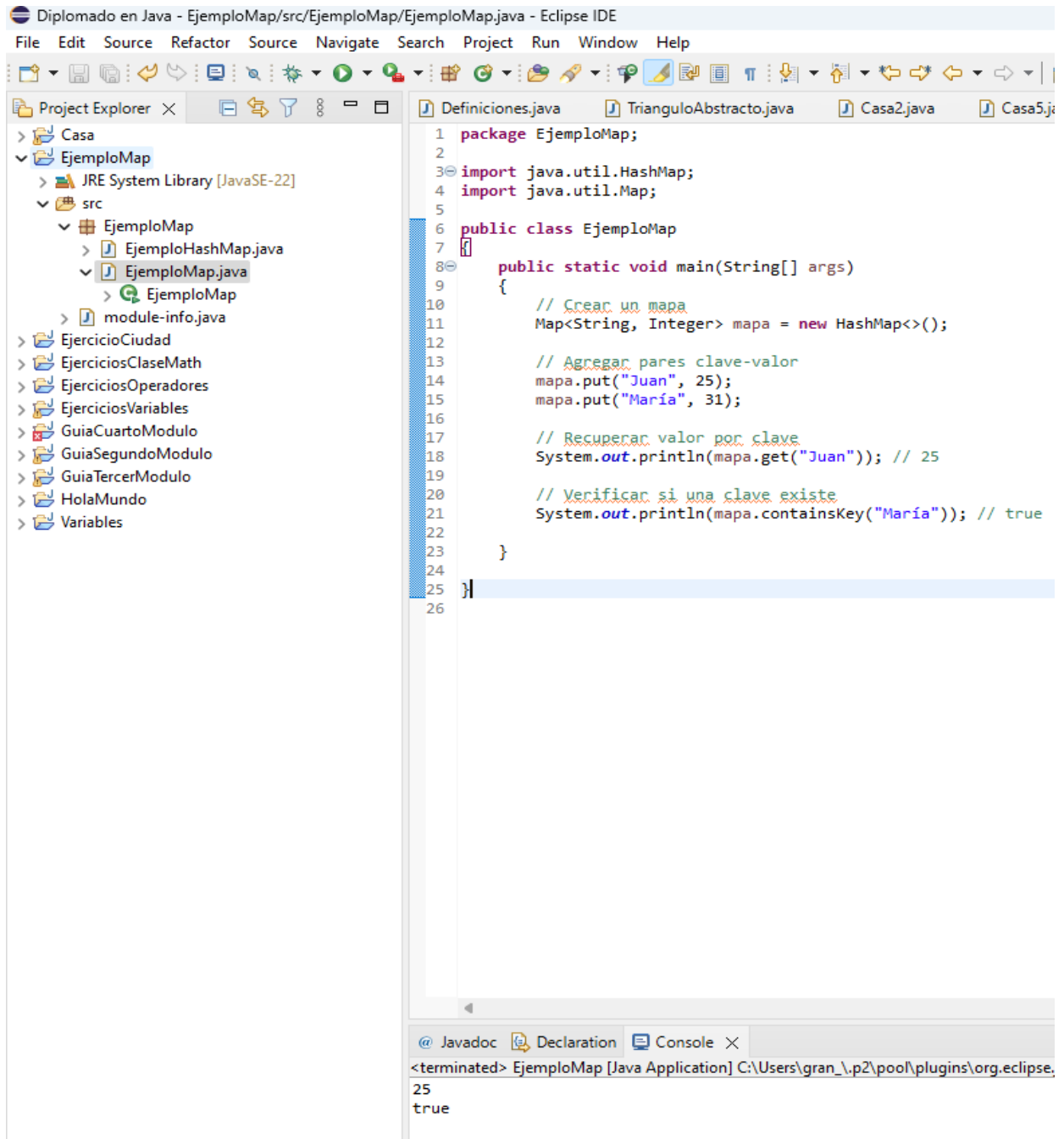
- ❖ Almacenamiento en pares clave-valor: Cada elemento en un Map está compuesto por una clave y un valor. Las claves en un Map deben ser únicas mientras que los valores pueden estar repetidos.

- ❖ Implementaciones Comunes: Algunas implementaciones comunes de Map se encuentran en Java y son descritas a continuación:
 - HashMap: Implementación basada en la tabla hash que permite muy eficiente inserción y búsqueda. No garantiza el orden de los elementos.
 - LinkedHashMap: Extiende HashMap y conserva un orden basado en la inserción de elementos, por lo que resulta útil para iterar elementos en el mismo orden en el que se introdujeron.
 - TreeMap: Implementación basada en el árbol rojo-negro. Arreglos naturales de los elementos en base a la clave o a un comparador dado por el usuario.
 - Hashtable: Muy similar a HashMap, pero sincronizado, por tanto, es útil en entornos multiproceso.
 - ConcurrentHashMap: Variante segura para los hilos para permitir operaciones concurrentes.
 - Claves Únicas y Valores Duplicados: No se permite que un Map tenga claves duplicadas, aunque sí se permiten valores duplicados asociados a las claves.
- ❖ Métodos Comunes de Map:
 - put(K key, V value): Este método se usa para insertar un par clave-valor en el mapa.
 - get(Object key): Devuelve el valor al más que la clave especificada.
 - containsKey(Object key): Este método se usa para verificar si el mapa contiene la clave dada.
 - containsValue(Object value): Este método se usa para verificar si el mapa contiene el valor dado.
 - remove(Object key): Elimina el par clave-valor asociado a la clave dada.
 - keySet(): Devuelve un conjunto que contiene todas las claves.
 - values(): Devuelve una colección que contiene todos los valores.
- ❖ entrySet(): Devuelve un conjunto que contiene los pares clave-valor.
- ❖ size(): Devuelve el número de parejas.
- ❖ isEmpty(): Este método se usa para verificar si este mapa contiene parejas clave-valor.

Implementaciones comunes

- ❖ HashMap: una implementación basada en hash tables, adecuada para la mayoría de los casos.
- ❖ TreeMap: una implementación basada en árboles binarios, adecuada para datos ordenados.
- ❖ LinkedHashMap: hash table con enlaces para mantener el orden de inserción.

Ejemplo De Aplicación



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Source, Navigate, Search, Project, Run, Window, and Help. The Project Explorer on the left shows a project named 'EjemploMap' with a source folder 'src' containing 'EjemploHashMap.java' and 'EjemploMap.java'. The main editor displays the code for 'EjemploMap.java'. The code defines a package 'EjemploMap', imports 'java.util.HashMap' and 'java.util.Map', and defines a public class 'EjemploMap' with a main method. The main method creates a HashMap, adds two entries ('Juan', 25) and ('María', 31), prints the value for 'Juan' (25), and prints whether 'María' is a key (true). The bottom console shows the output 'true'.

```
1 package EjemploMap;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public class EjemploMap
7 {
8     public static void main(String[] args)
9     {
10         // Crear un mapa
11         Map<String, Integer> mapa = new HashMap<>();
12
13         // Agregar pares clave-valor
14         mapa.put("Juan", 25);
15         mapa.put("María", 31);
16
17         // Recuperar valor por clave
18         System.out.println(mapa.get("Juan")); // 25
19
20         // Verificar si una clave existe
21         System.out.println(mapa.containsKey("María")); // true
22     }
23 }
24
25
26
```

@ Javadoc Declaration Console X
<terminated> EjemploMap [Java Application] C:\Users\gran_.p2\pool\plugins\org.eclipse.
25
true

Java.util.HashMap



Java Util HashMap es una clase que implementa la interfaz Map para el paquete java.util. Es una implementación de hash tables y almacena datos en forma de clave valor.

¿Para qué sirve?

- ❖ Almacenar y recuperar datos rápidamente utilizando claves únicas.
- ❖ Búsqueda rápida por clave.
- ❖ Implementar caches y almacenamiento de configuraciones.
- ❖ Representar datos en memoria estructurada.

Características

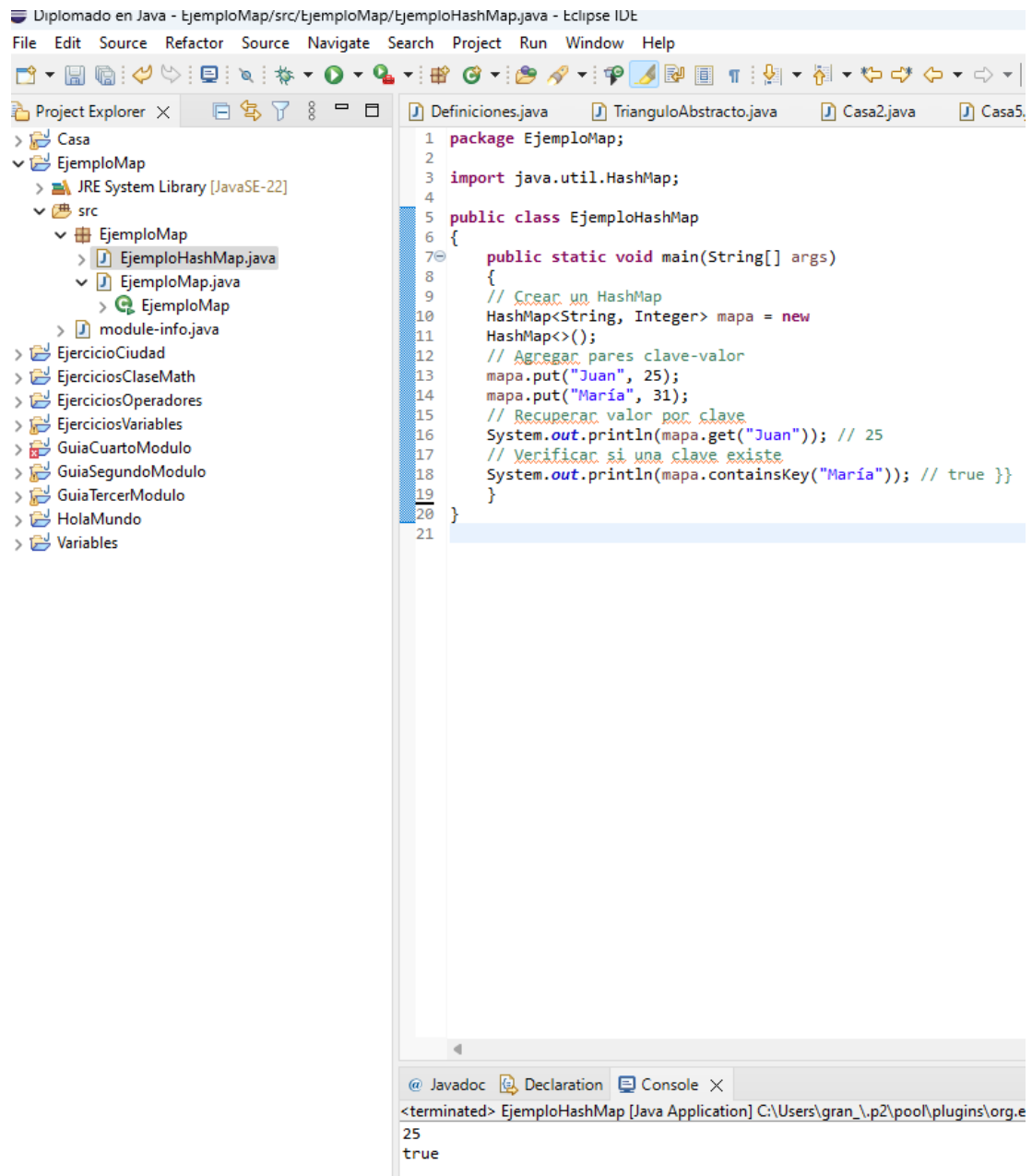
- ❖ No ordenada: no contiene la secuencia de inserción.
- ❖ No sincronizado: no es seguro para el acceso concurrente.
- ❖ Null: Permite null como clave y valor.
- ❖ Load Factor: Tiene un factor de carga (load factor).

Métodos principales

- ❖ put(K clave, V valor): agrega un nuevo par clave-valor o actualiza el valor de una clave existente.
- ❖ get(Object clave): recupera el valor asociado a una determinada clave.
- ❖ containsKey(Object clave): verifica si una clave existe en el mapa.
- ❖ remove(Object clave): elimina un par clave-valor.
- ❖ size(): devuelve el número de pares clave-valor en el mapa.

- ❖ isEmpty(): verifica si está vacío el mapa.
- ❖ keySet(): devuelve un conjunto de todas las claves en el mapa.
- ❖ values(): devuelve una colección de todos los valores Mappings en el mapa.

Ejemplo De Aplicación



The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the project structure: 'Casa' > 'EjemploMap' > 'src' > 'EjemploMap' > 'EjemploHashMap.java'. The main editor shows the code for 'EjemploHashMap.java'.

```

1 package EjemploMap;
2
3 import java.util.HashMap;
4
5 public class EjemploHashMap
6 {
7     public static void main(String[] args)
8     {
9         // Crear un HashMap
10        HashMap<String, Integer> mapa = new
11        HashMap<>();
12        // Agregar pares clave-valor
13        mapa.put("Juan", 25);
14        mapa.put("María", 31);
15        // Recuperar valor por clave
16        System.out.println(mapa.get("Juan")); // 25
17        // Verificar si una clave existe
18        System.out.println(mapa.containsKey("María")); // true }}
19    }
20 }
21

```

At the bottom, the Console window shows the output of the program:

```

<terminated> EjemploHashMap [Java Application] C:\Users\gran_\p2\poof\plugins\org.e
25
true

```

Ventajas

- ❖ Rendimiento rápido para búsquedas e inserciones.

- ❖ Fácil de usar y configurar.

Desventajas

- ❖ No segura para acceso concurrente.
- ❖ No mantiene el orden de inserción.

Alternativas

- ❖ TreeMap: ordenada.
- ❖ LinkedHashMap: mantiene el orden de inserción.
- ❖ ConcurrentHashMap: segura para acceso concurrente.

Texto De Estructura De Datos

JAVA UTIL MAP	JAVA UTIL HASHMAP
<ul style="list-style-type: none">❖ Definición: Interfaz que almacena datos en forma de clave-valor.❖ Método principal: put(), get(), containsKey(), remove().-❖ Propósito: Almacenar y recuperar datos rápidamente.❖ Características: Permite null como clave y valor, no ordenada.	<ul style="list-style-type: none">❖ Definición: Implementación de la interfaz Map basada en hash tables.❖ Método principal: put(), get(), containsKey(), remove().-❖ Propósito: Almacenar y recuperar datos rápidamente❖ Características: No ordenada, no sincronizada, permite null como clave y valor.



Diferencias

- ❖ Implementación: Map es una interfaz, mientras que HashMap es una implementación concreta de esa interfaz.
- ❖ Orden: HashMap no mantiene el orden de inserción, a diferencia de otras implementaciones como TreeMap o LinkedHashMap

- ❖ Sincronización: HashMap no es segura para acceso concurrente, a diferencia de ConcurrentHashMap.
- ❖ Rendimiento: HashMap ofrece rendimiento rápido para búsquedas e inserciones.
- ❖ En resumen, Map es la interfaz base de almacenamiento de datos en forma de clave-valor, mientras que HashMap es la específica para obtener un alto rendimiento y flexibilidad, pero con limitaciones en su sincronización y orden.

¿Qué Son Los Generics En Java?



Los generics en Java es una funcionalidad del lenguaje que permite parametrizar clases, interfaces y métodos con tipos, de modo que puedan funcionar con varios tipos de datos sin perder la seguridad de tipos en la compilación. Los generics se utilizan para escribir un código flexible y reutilizable y garantizan que los errores tipográficos se encuentren en la compilación y no en la ejecución.

Conceptos Clave de los Generics

Los conceptos clave son los siguientes se basan en los parámetros de tipo. Los generics usan parámetros de tipo que permiten la definición de clases, métodos e interfaces con “tipo” genérico, que se especifica entre ángulos <>.

Ejemplo De Aplicación

```
1 package Ejemplos;
2
3 // Definición de la clase contenedora con un nombre diferente (por ejemplo, "EjemploGenerics").
4 public class EjemploGenerics
5 {
6     // Clase genérica Box definida dentro de la clase contenedora.
7     class Box<T>
8     {
9         private T value; // Campo privado de tipo genérico T.
10
11         // Método para establecer el valor.
12         public void setValue(T value) {
13             this.value = value;
14         }
15
16         // Método para obtener el valor.
17         public T getValue() {
18             return value;
19         }
20     }
21
22     public static void main(String[] args)
23     {
24         // Instanciar la clase contenedora para acceder a la clase interna.
25         EjemploGenerics ejemploGenerics = new EjemploGenerics();
26         Box<Integer> integerBox = ejemploGenerics.new Box<>(); // Crear un Box de tipo Integer.
27         // Usar los métodos de la clase genérica.
28         integerBox.setValue(123);
29         System.out.println("Valor en la caja: " + integerBox.getValue()); // Imprime: Valor en la caja: 123
30     }
31 }
32
33
```

@ Javadoc Declaration Console X

<terminated> EjemploGenerics [Java Application] C:\Users\gran_lp2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_22.0.2.v20240116

Valor en la caja: 123

En el siguiente ejemplo, T es un parámetro de tipo que se utiliza para el tipo del objeto almacenado en la clase Box y la letra T es generalmente convencional, pero se pueden usar otras letras (E, K, V) o incluso nombres más descriptivos.

Uso de Generics en Clases

Cuando se aplican los generics se crean estructuras de datos flexibles. En el siguiente ejemplo, el parámetro T se especifica al crear el objeto:

Diplomado en Java - EjemploMap/src/Ejemplos/Ejemplo2Generics.java - Eclipse IDE

File Edit Source Refactor Source Navigate Search Project Run Window Help

Project Explorer Workspace

- Casa
 - EjemploMap
 - JRE System Library [JavaSE-22]
 - src
 - EjemploMap
 - EjemploHashMap.java
 - Main.java
 - Ejemplos
 - Ejemplo2Generics.java
 - EjemploGenerics.java
 - EjercicioCiudad
 - EjerciciosClaseMath
 - EjerciciosOperadores
 - EjerciciosVariables
 - GuiaCuartoModulo
 - GuiaSegundoModulo
 - GuiaTercerModulo
 - HolaMundo
 - Variables

```
1 package Ejemplos;
2
3 // Clase genérica Box que puede almacenar cualquier tipo de valor.
4 class Box<T> {
5     private T value;
6
7     // Método para establecer el valor.
8     public void setValue(T value) {
9         this.value = value;
10    }
11
12    // Método para obtener el valor.
13    public T getValue() {
14        return value;
15    }
16 }
17
18 public class Ejemplo2Generics {
19     public static void main(String[] args) {
20         // Crear un Box de tipo String.
21         Box<String> stringBox = new Box<>();
22
23         // Establecer un valor en la caja.
24         stringBox.setValue("Hello Generics");
25
26         // Imprimir el valor de la caja.
27         System.out.println(stringBox.getValue()); // Imprime: Hello Generics
28     }
29 }
30
```

@ Javadoc Declaration Console X

<terminated> Ejemplo2Generics [Java Application] C:\Users\gran_\p2\pool\plugins\org.eclipse.justj.openjd
Hello Generics

Uso de Generics en Métodos

También se pueden definir métodos con los generics dentro de las clases, ya sean generics o no. El parámetro de tipo se especifica entre <> antes del tipo de retorno:

```
1 package Ejemplos;
2
3 public class Ejemplo3Generics {
4
5     // Método genérico para imprimir los elementos de un array.
6     public <U> void printArray(U[] array) {
7         for (U element : array) {
8             System.out.println(element);
9         }
10    }
11
12    public static void main(String[] args) {
13        // Crear una instancia de la clase.
14        Ejemplo3Generics ejemplo = new Ejemplo3Generics();
15
16        // Definir diferentes tipos de arrays.
17        Integer[] arrayEnteros = { 1, 2, 3, 4, 5 };
18        String[] arrayStrings = { "Hola", "Mundo", "Generics" };
19        Double[] arrayDoubles = { 1.1, 2.2, 3.3 };
20
21        // Llamar al método printArray con diferentes tipos de arrays.
22        System.out.println("Array de Enteros:");
23        ejemplo.printArray(arrayEnteros); // Imprime cada número en una línea nueva.
24
25        System.out.println("\nArray de Strings:");
26        ejemplo.printArray(arrayStrings); // Imprime cada cadena en una línea nueva.
27
28        System.out.println("\nArray de Doubles:");
29        ejemplo.printArray(arrayDoubles); // Imprime cada número en una línea nueva.
30    }
31 }
32
33
```

@ Javadoc Declaration Console X

<terminated> Ejemplo3Generics [Java Application] C:\Users\gran_\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre!

Array de Enteros:

1
2
3
4
5

Array de Strings:

Hola
Mundo
Generics

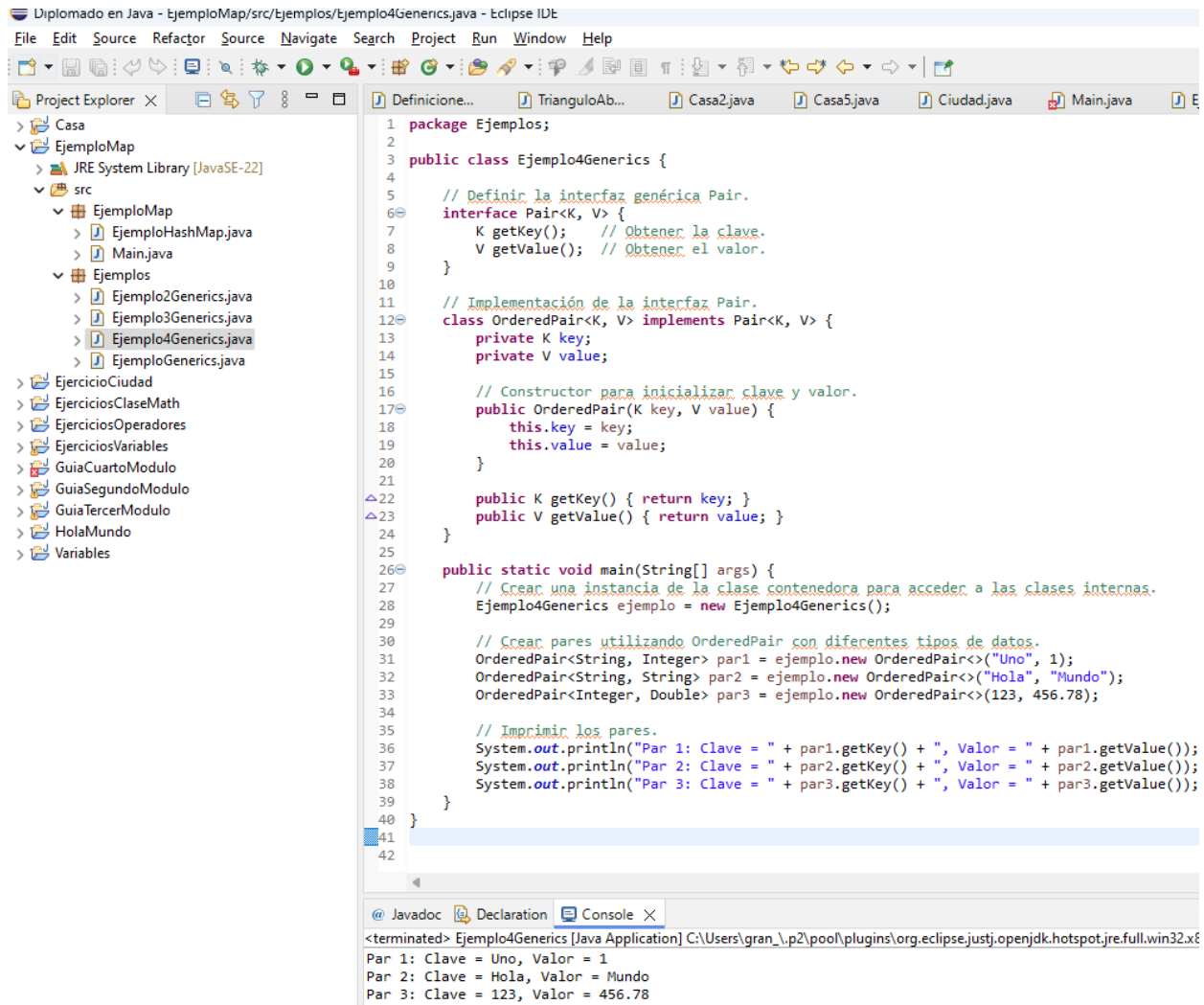
Array de Doubles:

1.1
2.2
3.3

En el ejemplo, U es el parámetro de tipo del método printArray, y puede funcionar con cualquier tipo de arreglo.

Uso de Generics en Interfaces

Las interfaces también pueden ser generics:



```
1 package Ejemplos;
2
3 public class Ejemplo4Generics {
4
5     // Definir la interfaz genérica Pair.
6     interface Pair<K, V> {
7         K getKey(); // Obtener la clave.
8         V getValue(); // Obtener el valor.
9     }
10
11     // Implementación de la interfaz Pair.
12     class OrderedPair<K, V> implements Pair<K, V> {
13         private K key;
14         private V value;
15
16         // Constructor para inicializar clave y valor.
17         public OrderedPair(K key, V value) {
18             this.key = key;
19             this.value = value;
20         }
21
22         public K getKey() { return key; }
23         public V getValue() { return value; }
24     }
25
26     public static void main(String[] args) {
27         // Crear una instancia de la clase contenedora para acceder a las clases internas.
28         Ejemplo4Generics ejemplo = new Ejemplo4Generics();
29
30         // Crear pares utilizando OrderedPair con diferentes tipos de datos.
31         OrderedPair<String, Integer> par1 = ejemplo.new OrderedPair<>("Uno", 1);
32         OrderedPair<String, String> par2 = ejemplo.new OrderedPair<>("Hola", "Mundo");
33         OrderedPair<Integer, Double> par3 = ejemplo.new OrderedPair<>(123, 456.78);
34
35         // Imprimir los pares.
36         System.out.println("Par 1: Clave = " + par1.getKey() + ", Valor = " + par1.getValue());
37         System.out.println("Par 2: Clave = " + par2.getKey() + ", Valor = " + par2.getValue());
38         System.out.println("Par 3: Clave = " + par3.getKey() + ", Valor = " + par3.getValue());
39     }
40 }
41
42
```

@ Javadoc Declaration Console X

```
<terminated> Ejemplo4Generics [Java Application] C:\Users\gran_\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\java.exe
Par 1: Clave = Uno, Valor = 1
Par 2: Clave = Hola, Valor = Mundo
Par 3: Clave = 123, Valor = 456.78
```

En este ejemplo, Pair es un interfaz genérico con dos parámetros llamados K y V, y OrderedPair implementa estas interfaces.

Bounded Type Parameters (Parámetros de Tipo Acotado)

Los tipos que se pueden usar en los generics pueden restringirse mediante bounded type parameters. Esto se hace con la palabra clave extends para limitarlos a una clase base o una interfaz:

```
1 package Ejemplos;
2
3 public class Ejemplo5Generics {
4
5     // Clase genérica 'Box' que solo acepta tipos que extiendan 'Number'.
6     class Box<T extends Number> {
7         private T value;
8
9         public void setValue(T value) {
10             this.value = value;
11         }
12
13         public T getValue() {
14             return value;
15         }
16     }
17
18     public static void main(String[] args) {
19         // Crear una instancia de la clase contenedora 'Ejemplo5Generics'.
20         Ejemplo5Generics ejemplo = new Ejemplo5Generics();
21
22         // Crear instancias de 'Box' con diferentes tipos de números.
23         Box<Integer> integerBox = ejemplo.new Box<>();
24         integerBox.setValue(123); // 'Box' de tipo 'Integer'.
25         System.out.println("Valor en integerBox: " + integerBox.getValue()); // Imprime: Valor en integerBox: 123
26
27         Box<Double> doubleBox = ejemplo.new Box<>();
28         doubleBox.setValue(456.78); // 'Box' de tipo 'Double'.
29         System.out.println("Valor en doubleBox: " + doubleBox.getValue()); // Imprime: Valor en doubleBox: 456.78
30
31         Box<Float> floatBox = ejemplo.new Box<>();
32         floatBox.setValue(9.99f); // 'Box' de tipo 'Float'.
33         System.out.println("Valor en floatBox: " + floatBox.getValue()); // Imprime: Valor en floatBox: 9.99
34
35         // Intentar con tipos no permitidos causará un error de compilación.
36         // Box<String> stringBox = ejemplo.new Box<>(); // Esto NO es válido y causará un error de compilación.
37     }
38 }
39
```

@ Javadoc Declaration Console X

<terminated> Ejemplo5Generics [Java Application] C:\Users\gran\AppData\Local\Temp\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_22.0.2.v20240802-1626

Valor en integerBox: 123
Valor en doubleBox: 456.78
Valor en floatBox: 9.99

Como podemos denotar en este ejemplo, T debe ser un subtipo de Number, como Integer, Double o Float.

Raw Types

Antes de Java 5, las colecciones como List no usaban generics, y esto daba lugar a errores de tipo en tiempo de ejecución. Un raw type es una instancia de una clase genérica que no tiene tipo especificado, ejemplo List en lugar de List<String>. Se desalienta el uso de este tipo debido a la seguridad del tipo perdida proporcionada por los generics.

Type Erasure

Durante la compilación, se eliminan en tipos genéricos (type erasure), es decir, el código generado no contiene información sobre los tipos genéricos. Por ejemplo, List<String> y List<Integer> se transforman en List porque se implementó la retrocompatibilidad en Java.

Ventajas de Usar Generics

- ❖ Código reutilizable: Permite crear clases y métodos para que se reutilicen con diferentes tipos de datos.
- ❖ Seguridad de los tipos en tiempo de compilación: Los errores en los tipos de uso se descubren en tiempo de compilación. Evitando que se produzcan errores en tiempos de ejecución.
- ❖ Eliminación de castings explícitos: Se basan en la ausencia de la necesidad de las conversiones de tipo innecesarias lo que permite el fácil desarrollo y alta legibilidad del código.

Ejemplo De Aplicación

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays a project named 'EjemploMap' with a source folder 'src'. Inside 'src', there is a package 'Ejemplos' containing several Java files, including 'EjemploCompletoGenerics.java' which is currently selected. The main editor window shows the code for 'EjemploCompletoGenerics.java'. The code includes package declarations, imports for 'ArrayList' and 'List', and a public class 'EjemploCompletoGenerics' with a 'main' method. The 'main' method demonstrates the use of generics by creating a 'Box<Integer>' object, adding elements to a 'List<String>', and calling a generic 'printList' method. The 'printList' method is a generic static method that iterates over a 'List<E>' and prints each element. At the bottom, the Console window shows the output of the program: 'Valor en Box: 10', 'Generics', 'in', and 'Java'.

```
1 package Ejemplos;
2 import java.util.ArrayList;
3 import java.util.List;
4
5 public class EjemploCompletoGenerics
6 {
7     public static void main(String[] args)
8     {
9         // Uso de una clase genérica
10        Box<Integer> integerBox = new Box<>();
11        integerBox.setValue(10);
12        System.out.println("Valor en Box: " + integerBox.getValue());
13
14        // Uso de un método genérico
15        List<String> stringList = new ArrayList<>();
16        stringList.add("Generics");
17        stringList.add("in");
18        stringList.add("Java");
19        printList(stringList);
20    }
21
22    // Método genérico con un parámetro de tipo
23    public static <E> void printList(List<E> list)
24    {
25        for (E element : list)
26        {
27            System.out.println(element);
28        }
29    }
30 }
31
32
33
```

@ Javadoc Declaration Console X
<terminated> EjemploCompletoGenerics [Java Application] C:\Users\gran_\p2\pool\plugins\org.ec
Valor en Box: 10
Generics
in
Java

En este ejemplo, Box es una clase genérica; printList es un método genérico que puede aceptar listas de cualquier tipo.

Los generics son una herramienta poderosa para escribir un código seguro, flexible y fácil de mantener.

Listas, Pilas y Colas

Las estructuras de datos son fundamentales en la programación, ya que permiten almacenar información de manera más eficiente se dividen en tres; listas, pilas y colas.

Listas



Son estructuras que almacenan una secuencia de elementos en un orden específico, los elementos pueden ser de cualquier tipo de datos incluyendo números enteros, cadena de textos u objetivos que permiten agregar o eliminar elementos en cualquier posición.

Las listas en java son variables, las cuales permiten almacenar grandes cantidades de datos, son similares a los Arrays o a las matrices. Su uso es vital para el manejo de gran volumen de datos, dentro de las herramientas de programación especialmente en las decisiones, las Listas poseen muchas funcionalidades interesantes lo cual nos permite lograr muchos desarrollos interesantes de manera más rápida y eficiente.

Tipos de listas

- ❖ ArrayList: una lista dinámica que se utiliza en un arreglo interno para almacenar los elementos.
- ❖ LinkedList: Una lista doblemente enlazada y que es utilizada para almacenar los elementos.

Métodos comunes de Lista

- ❖ add(E elemento): Agrega un elemento a la lista.
- ❖ remove(E elemento): Elimina un elemento de la lista.
- ❖ get(int indice): Obtiene un elemento en una posición específica.
- ❖ size(): Devuelve el número de elementos en la lista.
- ❖ contains(E elemento): Verifica si un elemento está en la lista.
- ❖ indexOf(E elemento): Devuelve el índice de un elemento.
- ❖ lastIndexOf(E elemento): Devuelve el índice del último elemento.

Pilas



Las pilas son estructuras de datos fundamentalmente que operan LFO, siendo la implementación inicial la primera en ser eliminada y que cuenta con ciertas características.

Las pilas son una estructura de datos cuya única de restricción es una manera de acceder a los elementos o almacenar en ella ya sea tanto como entrada como salida de los datos por un solo lugar, estas pertenecen a una estructura muy sencilla y muy útil para la aplicación de una implementación del análisis de la complejidad de los métodos que se están conformado.

Condicionales

- ❖ La única forma de ingresar elementos es desde el tope de la fila
- ❖ Su utilización es muy sencilla ya que tiene pocas operaciones
- ❖ Si la fila está vacía no tiene sentido nombrarse un tope ni a un fondo, en caso de que se vaya a utilizar un elemento que no se encuentre en el tope de la fila se debe realizar un volcado de la fila, una vez se realice la operación el elemento a volcar de la fila auxiliar se devuelve a la original.

Métodos de la clase Stack

- ❖ *push(E elemento)*: Se utiliza para agregar un elemento a la pila.
- ❖ *pop()*: Este método se emplea para eliminar el elemento superior de la pila.
- ❖ *peek()*: Este método devuelve el elemento superior o frontal de la pila, pero no lo elimina.
- ❖ *isEmpty()*: Este método determina si la pila está vacía.
- ❖ *search(E elemento)*: Mientras se ejecuta este método, el último valor devuelto por `str.equals(argumento)` se asigna al argumento y se regresa, es decir este método busca un elemento que se encuentre en la pila, determina y devuelve su posición.

Colas



Estructura de datos donde la frecuencia del primer elemento que entra es el primer elemento que sale. La operación de cola también es conocida como FIFO (First In, First Out). Es una estructura de datos en la que los elementos que ingresan primero son los primeros en salir después de realizarse la operación.

Esta existe por un extremo para la operación y otro extremo de la operación usada. Se tiene, los elementos en cola no se pueden manipular porque son en ambos extremos. Todos los datos en cola entran a un lado y salen al mismo lado por el que entraron.

Clases que implementan Queue

- ❖ LinkedList: Una implementación de una cola donde se emplea una lista enlazada.
- ❖ ArrayDeque: Una implementación de una cola utilizando un deque (Double Ended Queue) de arrays.
- ❖ PriorityQueue: Una implementación de una cola con prioridad.

Métodos de la interfaz Queue

- ❖ add(E e): Agrega un elemento a la cola.
- ❖ remove: Elimina el elemento de la parte de enfrente de la cola.
- ❖ element(): Lo mismo que peek y es equivalente, pero lanza una excepción si la cola está vacía.
- ❖ isEmpty(): Comprueba si la cola está vacía.
- ❖ contains: Busca el elemento que contiene en la cola.

Referencias Bibliográficas

Bibliografía



- ❖ Guía didáctica 1: Fundamentos - Diplomado Virtual En Programación En Java – Politécnico De Colombia
- ❖ Guía didáctica 4: Estructuras de datos - Diplomado Virtual En Programación En Java – Politécnico De Colombia

- ❖ Estructuras De Datos En Java – Luis Joyares Aguilar y Ignacio Zahanero Martínez – McGraw Hill.
- ❖ <https://blog.codmind.com/listas-en-java/>
- ❖ <https://academiasanroque.com/>
- ❖ <https://amsoft.medium.com/>
- ❖ <https://aws.amazon.com/es/what-is/java/>
- ❖ <https://azure.microsoft.com/>