|  |  |  |  |
| --- | --- | --- | --- |
|  | **DIPLOMADO EN**  **PROGRAMACIÓN EN JAVA** | | |
| **ACTIVIDAD EVALUATIVA - MÓDULO 4** | | |
| CÓDIGO: No aplica | VERSIÓN: 1 | Página **1** de **1** |

**Actividad evaluativa**

**Módulo 4: Estructuras de datos**

**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 Map es una interfaz en el lenguaje de programación Java, que es parte del paquete java.util. Se basa en un conjunto de métodos que permiten establecer, manejar datos en forma de clave-valor. Para ser más precisos, es una interfaz de la biblioteca estándar de Java, que se basa en la colección de pares clave-valor. En otras palabras, un par de un mapa consiste en exactamente un valor y una clave que no puede ser duplicada. En general, estos pares desempeñan un papel esencial ya que permiten a las personas manipular y recuperar información y almacenar y recuperar en función de la clave o la ejecución de varios datos en proceso. En otras palabras, un map es una colección de datos relacionada con la clave. Las personas pueden recuperar los valores utilizando las claves.

**¿Para qué sirve?**

Map ayuda a:

* Almacenar y recuperar usando una clave única.
* Realizar búsquedas rápidas basadas en una clave única.
* Implementar cachés y usar cache para almacenar configuraciones.

**Características:**

* Almacenamiento en pares clave-valor: cada elemento en un Map consta de una clave y un valor. Las claves deben ser únicas dentro del mapa, mientras que los valores pueden ser duplicados.
* Implementaciones Comunes: Algunas implementaciones comunes del Mapo se utilizan en Java y se describen a continuación:
* *HashMap:* implementación basada en la tabla hash que proporciona 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 es útil para iterar elementos en el mismo orden que se insertaron.
* *TreeMap:* Implementación basada en el árbol rojo-negro. Arreglos naturales de los elementos en función de la clave o de un comparador que proporciona el usuario.
* *Hashtable:* Similar a HashMap pero sincronizado, por lo tanto, es útil en ambientes multiproceso.
* *ConcurrentHashMap:* Variante segura para hilos diseñada para operaciones concurrentes.
* *Claves Únicas y Valores Duplicados:* No se permite que un Map tenga claves duplicadas, aunque los valores asociados con las claves pueden ser duplicados.
* 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 específicadas.
  + containsKey(Object key): Este método se usa para verificar si el mapa contiene la clave dada.
  + containsVaue(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.
  + kaySet(): Devuelve un conjunto que contiene todas las claves.
  + values(0): Devuelve una colección que contiene todos los valores.
* entrySet(): Devuelve un conjunto que contiene los pares clave-valor.
* size(): Devuelve el número de parejas.
* isEmty (): Este método se usa para verificar si este mapa contiene parejas clave-valor.

**Implementaciones comunes**

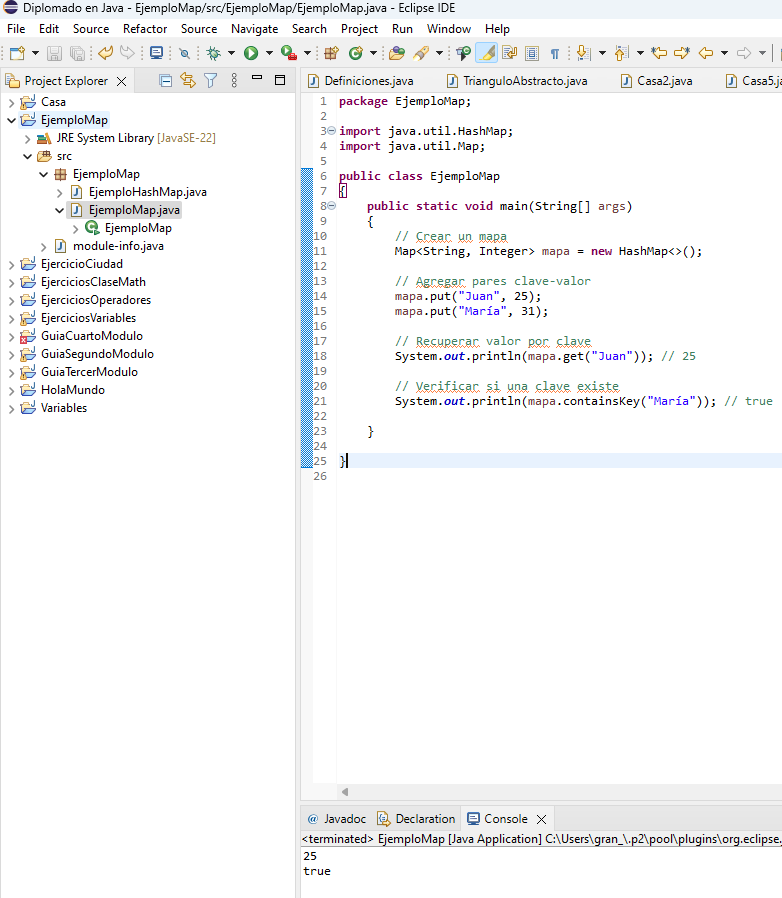
Hay varias implementaciones comunes de la interfaz Map:

❖ 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:**

******

***- 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?**

HashMap se utiliza para:

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

**Características**

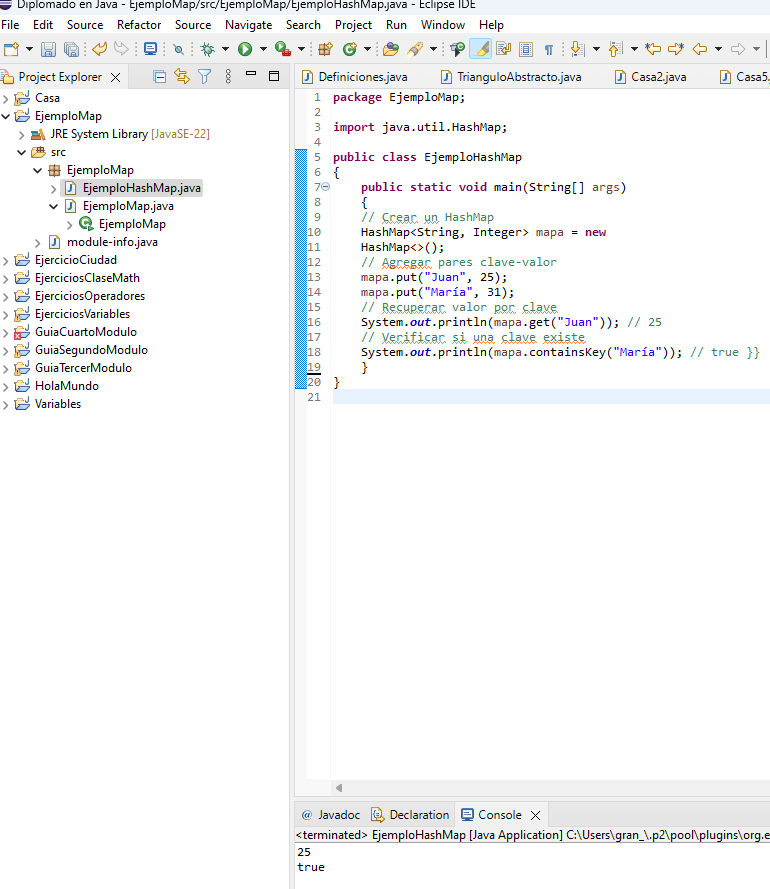
* No ordenada – no contiene la secuencia de inserción.
* No sincronizado – no es seguro para el acceso concurrente.
* Permite null como clave y valor.
* Tiene un factor de carga (load factor).

**Métodos principales**

Algunos de los principales métodos de HashMap son los siguientes:

* 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 valoresMappings en el mapa.

**Ejemplo:**



**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**

****

**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.

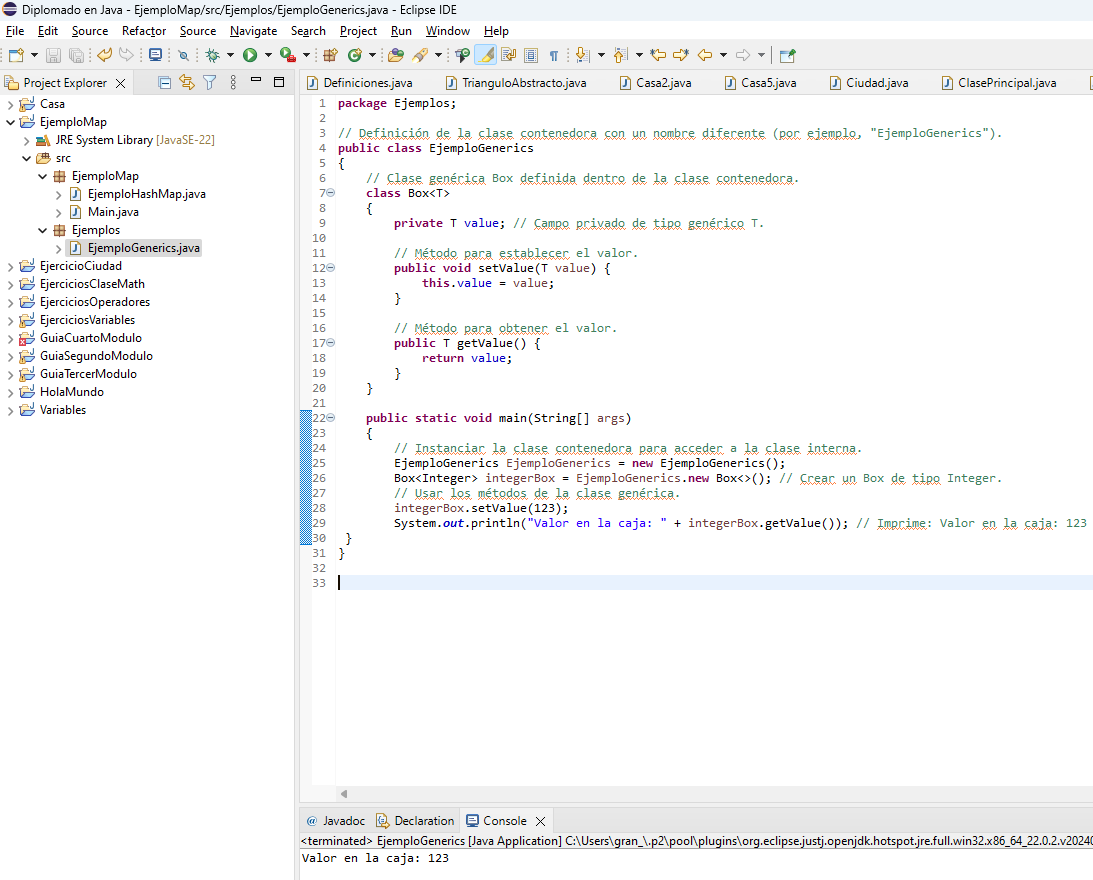
1. ***¿Qué son los generics en Java? (15 %)***

Los generics en Java son una característica del lenguaje que permite crear clases, interfaces y métodos parametrizados con tipos, de forma que puedan funcionar con diferentes tipos de datos sin perder seguridad de tipos en tiempo de compilación. Se utilizan para escribir código más flexible y reutilizable, garantizando que los errores de tipo se detecten en tiempo de compilación y no en tiempo de ejecución.

**Conceptos Clave de los Generics**

Parámetros de Tipo Los generics usan parámetros de tipo que permiten definir clases, métodos e interfaces con un tipo "genérico". Un parámetro de tipo se denota con una letra entre ángulos <>.

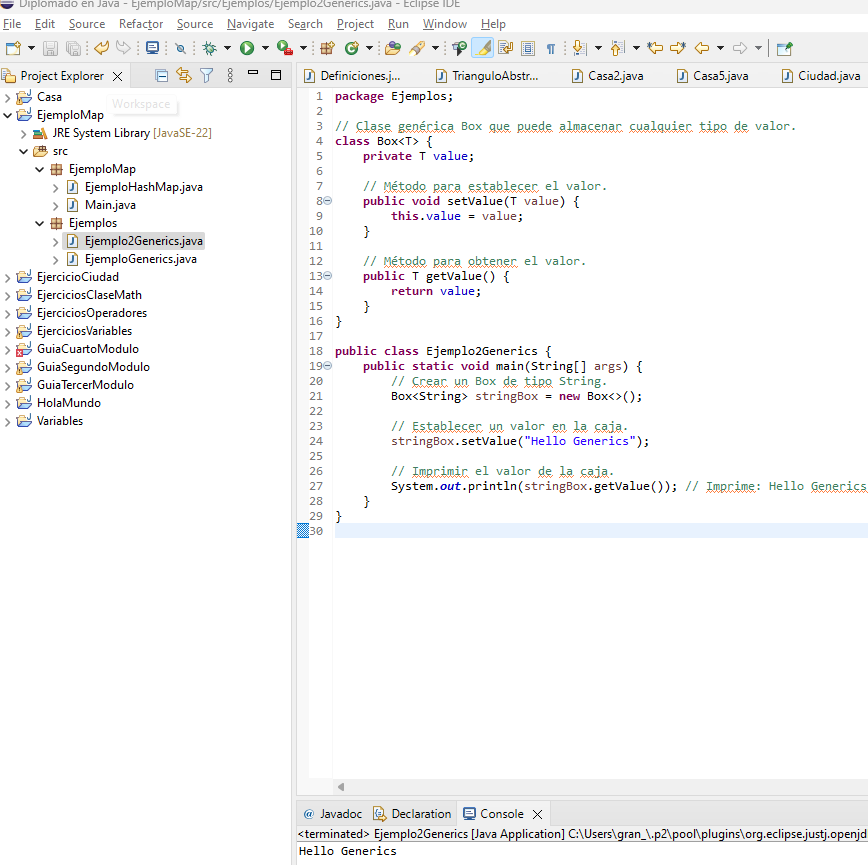
**Ejemplo:**



En este ejemplo, T es el parámetro de tipo que se usa para representar el tipo del objeto almacenado en la clase Box. La letra T es convencional, pero puedes usar otras letras como E, K, V o incluso nombres más descriptivos.

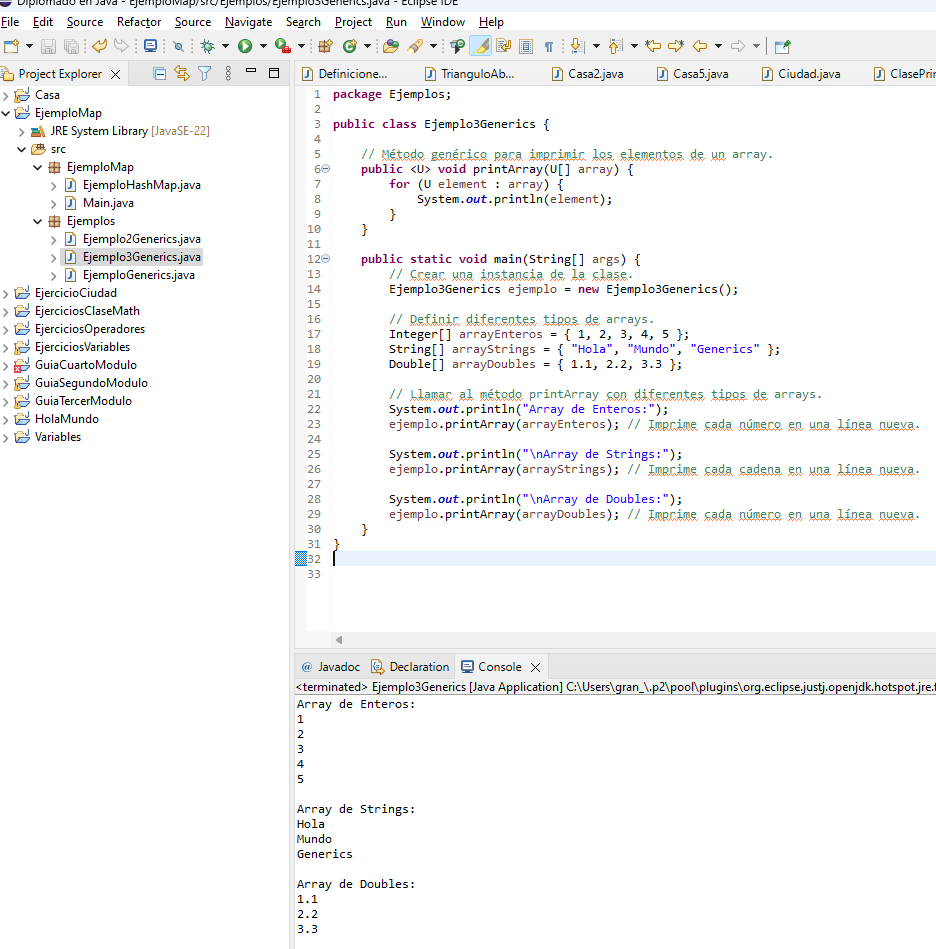
**Uso de Generics en Clases**

Los generics se pueden aplicar a clases para crear estructuras de datos flexibles. Por ejemplo, en la clase Box anterior, el parámetro T se puede especificar al crear un objeto:

****

**Uso de Generics en Métodos**

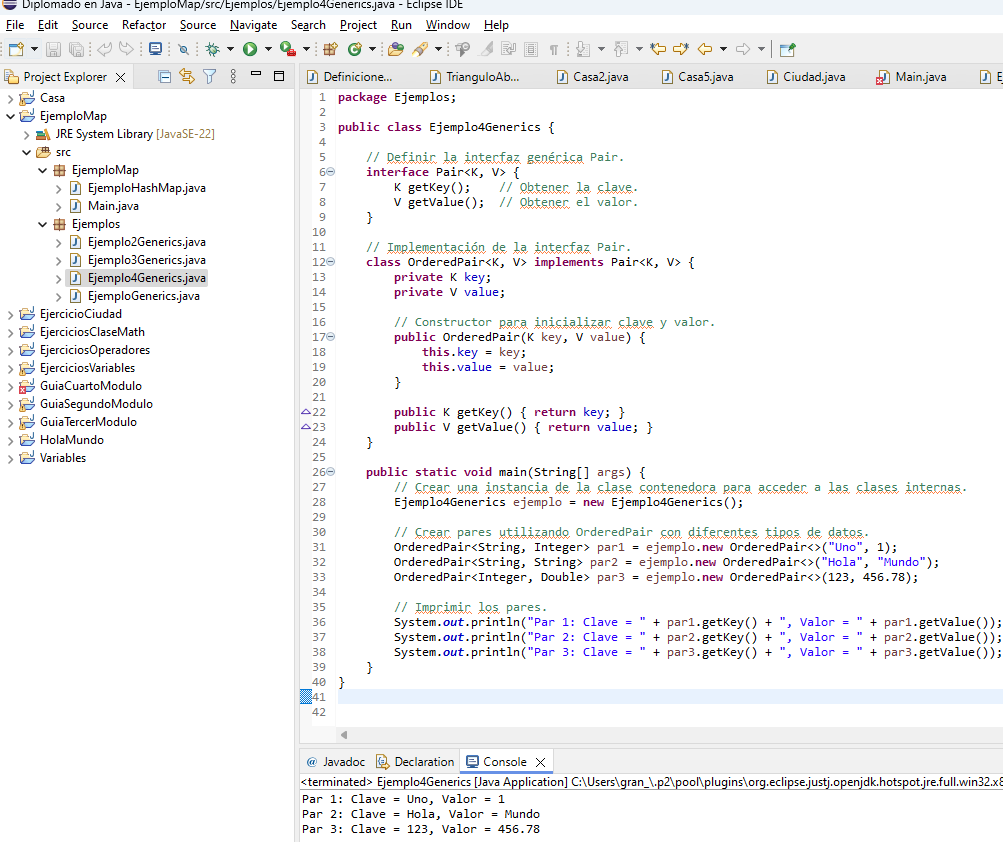
También se pueden definir métodos con generics dentro de clases, ya sean generics o no. El parámetro de tipo se especifica entre <> antes del tipo de retorno:



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

**Uso de Generics en Interfaces**

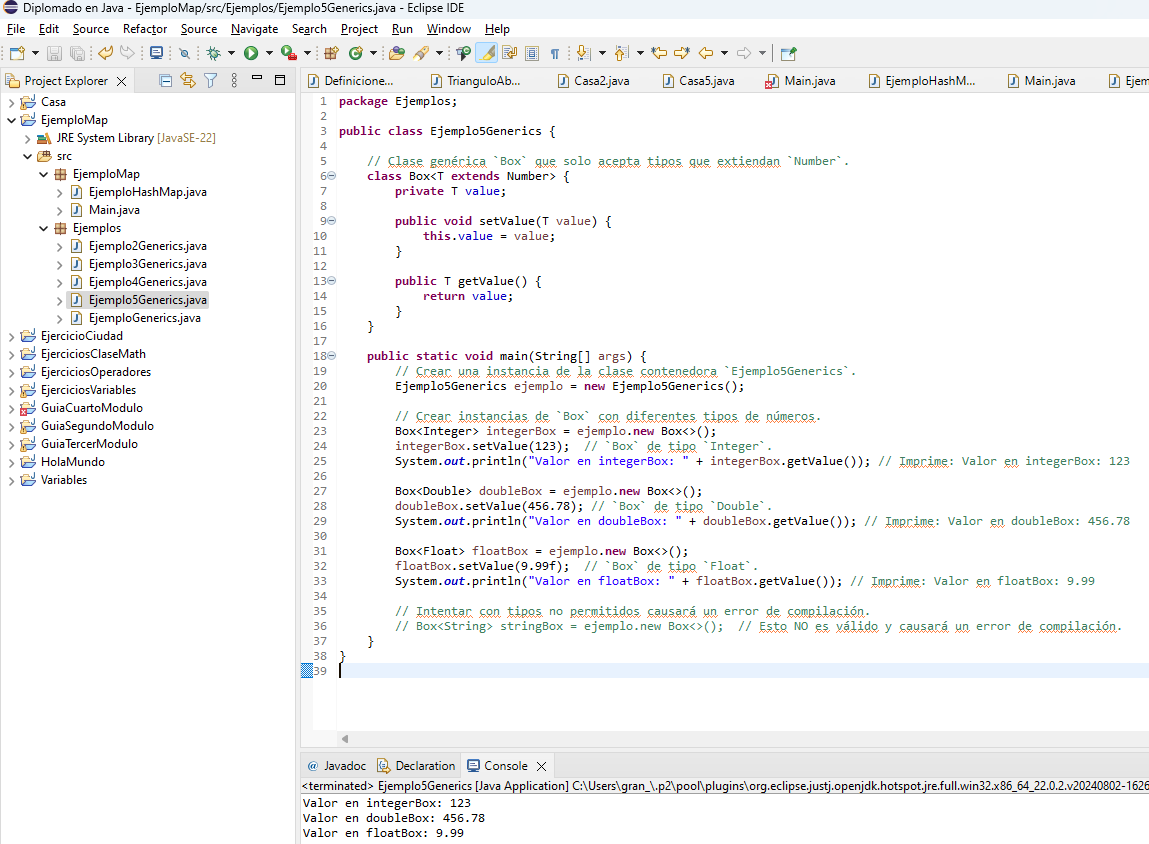
Las interfaces también pueden ser generics:



En este ejemplo, Pair es una interfaz genérica con dos parámetros (K y V), y OrderedPair la implementa usando esos tipos.

**Bounded Type Parameters (Parámetros de Tipo Acotado)**

Se pueden restringir los tipos que se pueden usar con los generics utilizando bounded type parameters. Esto se hace con la palabra clave extends para limitar los tipos a una clase base o a una interfaz:



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 sin especificar un tipo, por ejemplo, List en lugar de List<String>. Se recomienda evitar su uso, ya que pierden la seguridad de tipos que ofrecen los generics.

**Type Erasure**

Durante la compilación, los tipos genéricos se eliminan (type erasure), lo que significa que el código generado no contiene información de tipos genéricos. Por ejemplo, List<String> y List<Integer> se convierten en List. Esto se hace por compatibilidad con versiones anteriores de Java.

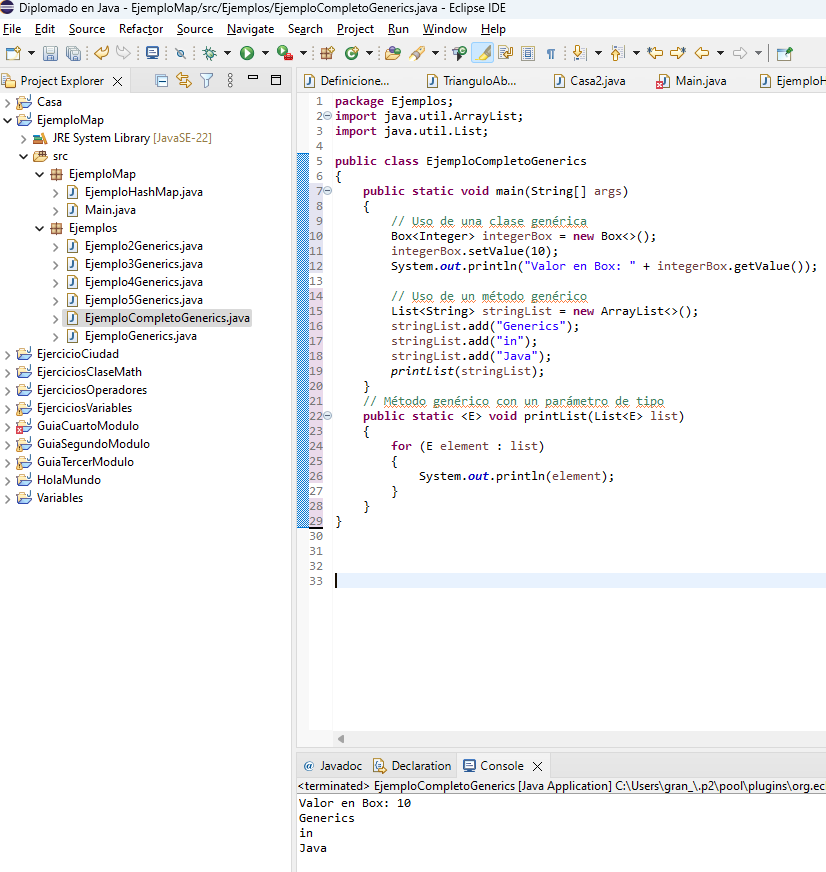
**Ventajas de Usar Generics**

Reutilización de Código: Permiten crear clases y métodos reutilizables para diferentes tipos de datos.

Seguridad de Tipos en Tiempo de Compilación: Los errores de tipo se detectan en tiempo de compilación, evitando errores en tiempo de ejecución.

Eliminación de Castings Explícitos: Se evitan conversiones de tipo innecesarias, facilitando el desarrollo y la legibilidad del código.

**Ejemplo**



En este ejemplo, Box es una clase genérica, y printList es un método genérico que acepta listas de cualquier tipo.

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

1. ***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***

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

**Listas:**

Las listas son estructuras que almacenan una secuencia de elementos en un orden especifico, estos 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 que permiten almacenar grandes cantidades de datos estas son similares a los Arrays o a las matrices, el uso de las listas es esenciales para el manejo de grandes volúmenes de datos, dentro de sus herramientas de programación especialmente en las decisiones que posee muchas funcionalidades interesantes lo cual nos permite lograr muchas cosas fabulosas de manera más rápida y eficiente.

Existen diferentes tipos de líneas:

1. Array list: una lista dinámica que es utilizada en un arreglo interno para almacenar los elementos.

2. Linked list: una lista doblemente enlazada que se utiliza para almacenar los elementos.

**Métodos comunes de Lista**

1. add(E elemento): Agrega un elemento a la lista.

2. remove(E elemento): Elimina un elemento de la lista.

3. get(int indice): Obtiene un elemento en una posición específica.

4. size(): Devuelve el número de elementos en la lista.

5. contains(E elemento): Verifica si un elemento está en la lista.

6. indexOf(E elemento): Devuelve el índice de un elemento.

7. lastIndexOf(E elemento): Devuelve el índice del último elemento.

**Pilas:**

Las pilas son estructuras de datos fundamentalmente que operan LFO lo que significa que es el implemento que se añadió es el primero en ser eliminado y contiene unas características.

Las pilas es 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, este pertenece a una estructura muy sencilla y muy útil para la aplicación de una implementación de análisis de la complejidad de los métodos que están conformado.

1. la única forma de ingresar elementos es desde el tope de la fila.

2. su utilización es muy sencilla ya que tiene pocas operaciones.

3. 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 realiza un volcado de la fila, una vez se realice la operación el elemento a volcar de la fila auxiliar se devuelve a la fila original.

**Métodos de la clase Stack**

1. push(E elemento): Agrega un elemento a la pila.

2. pop(): Elimina el elemento superior de la pila.

3. peek(): Devuelve el elemento superior de la pila sin eliminarlo.

4. isEmpty(): Verifica si la pila está vacía.

5. search(E elemento): Busca un elemento en la pila y devuelve su posición.

**Colas:**

Una cola es una estructura de datos donde el primer elemento en entrar es el primer elemento de salir, una cola es una estructura de datos que se puede caracterizar por una frecuencia de elementos donde la operación se realiza por un extremo y la operación por otro.

Una cola es una estructura de datos que almacena elementos de una lista que permite acceder a los datos por uno de los dos extremos de la lista, un elemento se conecta en la cola por la parte final de una lista suprimiendo o eliminando su parte inicial de lista, todos los datos entran por un lado y salen por el otro en el mismo orden que ingresaron, las colas también se le conocen como estructuras FIFO(first in, first, out).

**Clases que implementan Queue**

1. LinkedList: Una implementación de una cola utilizando una lista enlazada.

2. ArrayDeque: Una implementación de una cola utilizando un deque de arrays.

3. PriorityQueue: Una implementación de una cola con prioridad.

**Métodos de la interfaz Queue**

1. add(E elemento): Agrega un elemento a la cola.

2. remove(): Elimina el elemento frontal de la cola.

3. element(): Devuelve el elemento frontal de la cola sin eliminarlo.

4. isEmpty(): Verifica si la cola está vacía.

5. contains(E elemento): Busca un elemento en la cola.

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

**➢ Bibliografías**

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/

Los criterios de evaluación de la actividad son los siguientes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aspecto para evaluar** | **Ningún desarrollo** | **Nivel 1**  **Necesita mejora** | **Nivel 2**  **Aceptable** | **Nivel 3**  **Excelente** |
| Identifica las estructuras de datos Java.util.Map y Java.util.HashMap, su utilidad y métodos principales  (30 %) | No se han realizado consultas sobre Java.util.Map y Java.util.HashMap. | Se han consultado parcialmente las estructuras de datos, faltando detalles importantes sobre su utilidad y métodos. | Identifica las estructuras de datos en la consulta realizada, pero faltan algunos detalles. | Las consultas están completas, correctas y detalladas, abarcando tanto el propósito como los métodos principales. |
| **(0 puntos)** | **(15 puntos)** | **(20 puntos)** | **(30 puntos)** |
| Describe qué son los *generics* en Java  (15 %) | No se ha realizado la consulta sobre *generics* en Java. | La consulta está incompleta o presenta errores significativos. | La consulta está completa, pero con algunos errores menores o faltan algunos detalles. | La consulta está completa y correcta, mostrando una comprensión clara de los *generics* en Java. |
| **(0 puntos)** | **(5 puntos)** | **(10 puntos)** | **(15 puntos)** |
| Comprende los conceptos de listas, pilas y colas, sus principales métodos y diferencias con otras estructuras  (45 %) | No se ha elaborado el texto sobre las estructuras de datos. | El texto es muy corto o incompleto, con información incorrecta o confusa sobre las estructuras de datos. | El texto está completo, pero hay algunos errores o falta profundidad en la comparación entre estructuras o los métodos principales. | El texto es completo, bien estructurado, y compara de manera precisa las diferentes estructuras de datos con ejemplos claros y correctos. |
| **(0 puntos)** | **(22 puntos)** | **(30 puntos)** | **(45 puntos)** |
| Referencias bibliográficas y elementos adicionales  (10 %) | No se han incluido referencias bibliográficas ni elementos adicionales. | Se han incluido algunas referencias o elementos adicionales, pero son insuficientes o incorrectos. | Las referencias y elementos adicionales están presentes, pero hay algunos errores menores o falta alguno de los componentes. | Las referencias y elementos adicionales están completos, correctos y contribuyen significativamente a la calidad del entregable. |
| **(0 puntos)** | **(5 puntos)** | **(7 puntos)** | **(10 puntos)** |
|