|  |  |  |  |
| --- | --- | --- | --- |
|  | **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 Java que representa una colección de pares clave-valor. Es una de las clases más utilizadas en Java para almacenar y recuperar datos.

**• Características de java.util.Map**

1. Claves únicas: Cada clave en el mapa es única.

2. Valores asociados: Cada clave tiene un valor asociado.

3. No ordenada: Los elementos en el mapa no tienen un orden específico.

**• Métodos principales de java.util.Map**

1. put(K clave, V valor): Agrega un nuevo par clave-valor al mapa.

2. get(K clave): Recupera el valor asociado con la clave especificada.

3. containsKey(K clave): Verifica si la clave existe en el mapa.

4. containsValue(V valor): Verifica si el valor existe en el mapa.

5. remove(K clave): Elimina el par clave-valor del mapa.

6. size(): Devuelve el número de pares clave-valor en el mapa.

7. isEmpty(): Verifica si el mapa está vacío.

8. keySet(): Devuelve un conjunto de claves en el mapa.

9. values(): Devuelve una colección de valores en el mapa.

10. entrySet(): Devuelve un conjunto de pares clave-valor en el mapa.

**• Implementaciones de java.util.Map**

1. HashMap: La implementación más común, utiliza una tabla de hash para almacenar los datos.

2. TreeMap: Ordena las claves en un árbol binario, útil para datos ordenados.

3. LinkedHashMap: Combina las características de HashMap y TreeMap.

4. ConcurrentHashMap: Diseñada para entornos multihilo.

**• Ejemplo**

import java.util.HashMap; public class EjemploMap

{ public static void main(String[] args)

Map<String, Integer> edadPorNombre = new HashMap<>() edadPorNombre.put("Juan", 25);

edadPorNombre.put("María", 31);

edadPorNombre.put("Pedro", 42);

System.out.println(edadPorNombre.get("Juan")); // 25

System.out.println(edadPorNombre.containsKey("María")); // true

System.out.println(edadPorNombre.size()); // 3 }}

- Java.util.HashMap

Java.util.HashMap es una clase en Java que implementa la interfaz Map y utiliza una tabla de hash para almacenar los datos. Es una de las implementaciones más comunes de Map en Java.

• Características de HashMap

1. Almacenamiento de pares clave-valor: HashMap almacena datos en forma de pares clave-valor.

2. Claves únicas: Cada clave en el mapa es única.

3. Valores asociados: Cada clave tiene un valor asociado.

4. No ordenada: Los elementos en el mapa no tienen un orden específico.

5. Acceso rápido: HashMap proporciona acceso rápido a los datos mediante la clave.

6. Flexible: Puede almacenar objetos de cualquier tipo como claves y valores.

• Métodos principales de HashMap

1. put(K clave, V valor): Agrega un nuevo par clave-valor al mapa.

2. get(K clave): Recupera el valor asociado con la clave especificada.

3. containsKey(K clave): Verifica si la clave existe en el mapa.

4. containsValue(V valor): Verifica si el valor existe en el mapa.

5. remove(K clave): Elimina el par clave-valor del mapa.

6. size(): Devuelve el número de pares clave-valor en el mapa.

7. isEmpty(): Verifica si el mapa está vacío.

8. keySet(): Devuelve un conjunto de claves en el mapa.

9. values(): Devuelve una colección de valores en el mapa.

10. entrySet(): Devuelve un conjunto de pares clave-valor en el mapa.

• Constructores de HashMap

1. HashMap(): Crea un mapa vacío con capacidad inicial predeterminada.

2. HashMap(int inicialCapacity): Crea un mapa vacío con capacidad inicial especificada.

3. HashMap(int inicialCapacity, float carga): Crea un mapa vacío con capacidad inicial y carga especificadas.

• Ejemplo de uso

import java.util.HashMap; public class EjemploHashMap {

public static void main(String[] args)

{ HashMap<String, Integer> edadPorNombre = new HashMap<>(); edadPorNombre.put("Juan", 25);

edadPorNombre.put("María", 31);

edadPorNombre.put("Pedro", 42);

System.out.println(edadPorNombre.get("Juan")); // 25 System.out.println(edadPor

Nombre.containsKey("María")); // true

System.out.println(edadPorNombre.size()); // 3 }

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

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

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

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)** |
|