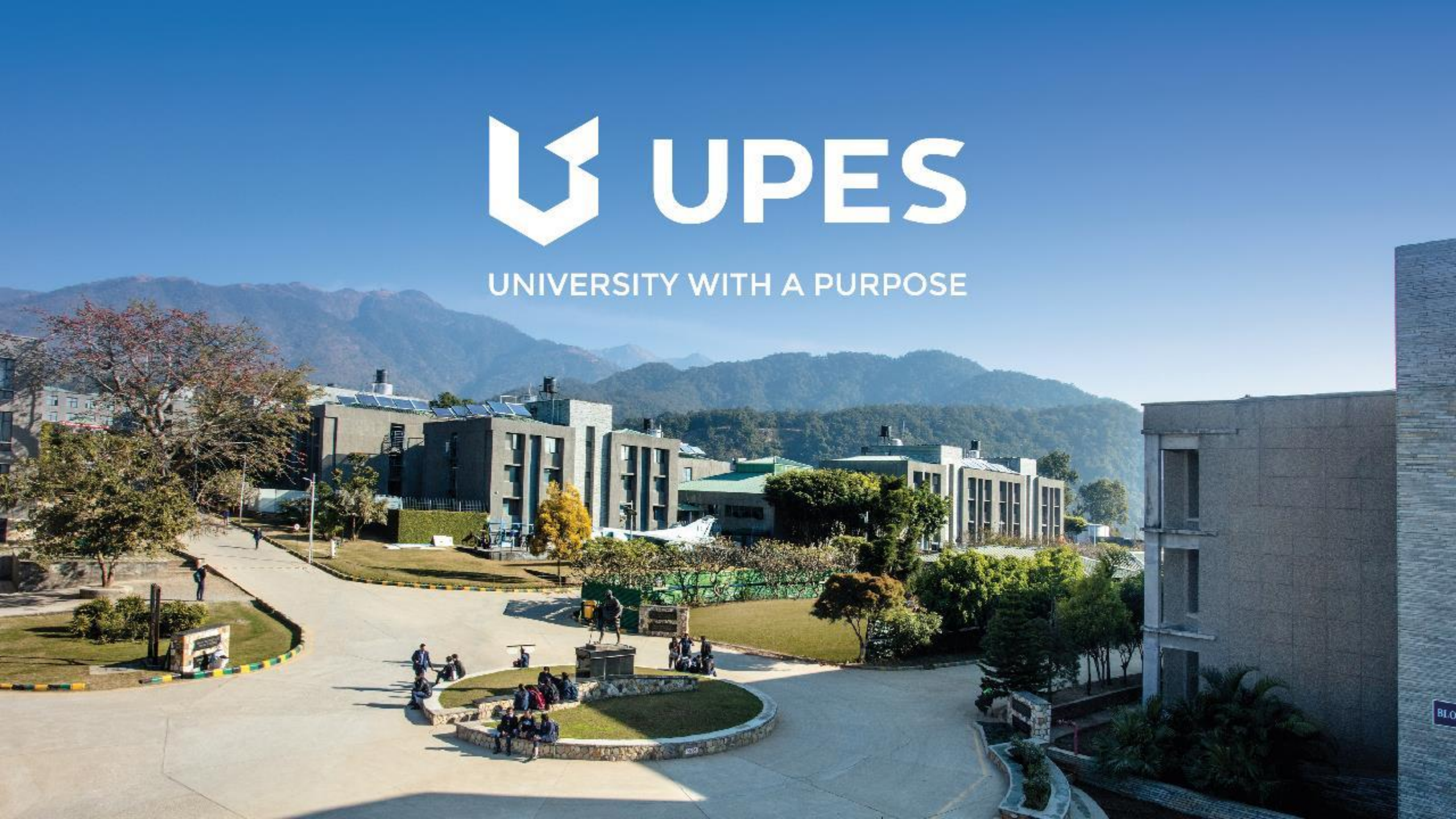
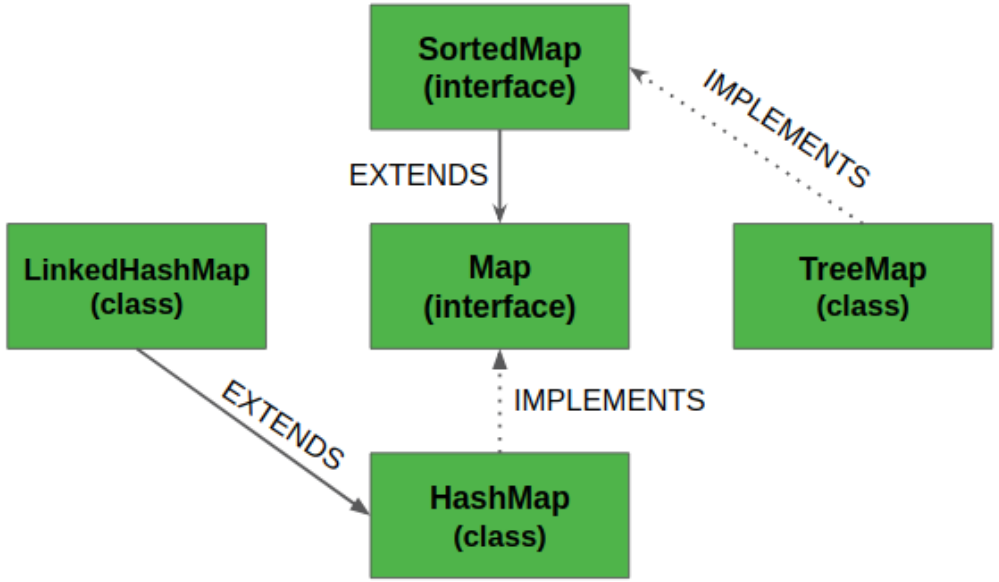

UPES
UNIVERSITY WITH A PURPOSE

Map and SortedMap

Map Interface

- The Map interface present in java.util package represents a mapping between a key and a value.
- The Map interface is not a subtype of the Collection interface.
- Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.



MAP Hierarchy in Java

Map Interface

```
// Java program to demonstrate the working of Map interface
import java.util.*;
class HashMapDemo {
        public static void main(String args[])    {
                Map<String, Integer> hm = new HashMap<String, Integer>();

                hm.put("a", 100);
                hm.put("b", 200);
                hm.put("c", 300);
                hm.put("d", 400);

                // Traversing through the map
                for (Map.Entry<String, Integer> me : hm.entrySet()) {
                        System.out.print(me.getKey() + ":");
                        System.out.println(me.getValue());                    }
        }
}
```

Map Interface

Performing various operations using *Map Interface* and *HashMap Class*

Since Map is an interface, it can be used only with a class that implements this interface. Now, let's see how to perform a few frequently used operations on a Map using the widely used HashMap class. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the map.

Map Interface

1. **Adding Elements:** In order to add an element to the map, we can use the put() method. However, the insertion order is not retained in the hashmap. Internally, for every element, a separate hash is generated and the elements are indexed based on this hash to make it more efficient.

```
// Java program to demonstrate the working of Map interface
import java.util.*;
class GFG {
          public static void main(String args[])  {
                    // Default Initialization of a Map
                    Map<Integer, String> hm1 = new HashMap<>();
                    // Initialization of a Map using Generics
                    Map<Integer, String> hm2 = new HashMap<Integer, String>();
                    // Inserting the Elements
                    hm1.put(1, "Dehradun");
                    hm1.put(2, "is");
                    hm1.put(3, "beautiful");
                    hm2.put(1, " Dehradun ");
                    hm2.put(2, "is");
                    hm2.put(3, "beautiful");
                    System.out.println(hm1);      System.out.println(hm2);      } }
```

Map Interface

2. Changing Elements: After adding the elements if we wish to change the element, it can be done by again adding the element with the put() method. Since the elements in the map are indexed using the keys, the value of the key can be changed by simply inserting the updated value for the key for which we wish to change.

```
// Java program to demonstrate the working of Map interface
import java.util.*;
class GFG {
        public static void main(String args[])     {
                // Initialization of a Map using Generics
                Map<Integer, String> hm1 = new HashMap<Integer, String>();

                // Inserting the Elements
                hm1.put(1, "Dehradun");
                hm1.put(2, "Dehradun");
                hm1.put(3, "Dehradun");

                System.out.println("Initial Map " + hm1);

                hm1.put(2, "For");

                System.out.println("Updated Map " + hm1);
        }
}
```

Map Interface

3. Removing Element: In order to remove an element from the Map, we can use the remove() method. This method takes the key value and removes the mapping for a key from this map if it is present in the map.

```
// Java program to demonstrate the working of Map interface

import java.util.*;
class GFG {

        public static void main(String args[]) {

                // Initialization of a Map using Generics
                Map<Integer, String> hm1 = new HashMap<Integer, String>();

                // Inserting the Elements
                hm1.put(1, “Dehradun");
                hm1.put(2, “is");
                hm1.put(3, “not");
                hm1.put(4, “beautiful");

                // Initial Map
                System.out.println(hm1);
                hm1.remove(3);
                System.out.println(hm1);  } }
```

Map Interface

4. Iterating through the Map: There are multiple ways to iterate through the Map. The most famous way is to use a for-each loop and get the keys. The value of the key is found by using the getValue() method.

```
// Java program to demonstrate the working of Map interface
import java.util.*;
class GFG {
        public static void main(String args[])     {
                // Initialization of a Map using Generics
                Map<Integer, String> hm1 = new HashMap<Integer, String>();
                // Inserting the Elements
                hm1.put(1, "Dehradun");
                hm1.put(2, "is");
                hm1.put(3, "beautiful");
                for (Map.Entry mapElement : hm1.entrySet()) {
                        int key  = (int)mapElement.getKey();

                        // Finding the value
                        String value = (String)mapElement.getValue();

                        System.out.println(key + " : "+ value);
                }
        }
}
```

Why and When to use Maps?

Maps are perfect to use for key-value association mapping such as dictionaries. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys. Some examples are:

- A map of error codes and their descriptions.
- A map of zip codes and cities.
- A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
- A map of classes and students. Each class (key) is associated with a list of students (value).

Classes which implement the Map interface

1. HashMap: HashMap is a part of Java's collection since Java 1.2. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs. To access a value one must know its key. This class uses a technique called Hashing. Hashing is a technique of converting a large String to small String that represents the same String. A shorter value helps in indexing and faster searches. Let's see how to create a map object using this class.

```
// Java Program to illustrate the Hashmap Class
import java.util.*;
public class GFG {

        public static void main(String[] args)     {
                Map<String, Integer> map
                        = new HashMap<>();

                map.put("vishal", 10);
                map.put("sachin", 30);
                map.put("vaibhav", 20);

                for (Map.Entry<String, Integer> e : map.entrySet())
                        System.out.println(e.getKey() + " "+ e.getValue());
        }
}
```

References

https://www.geeksforgeeks.org/map-interface-java-examples/

THANK YOU