Assignment-16

Part-A

Que-1)What is the difference between shallow copy and deep copy in collections?

Shallow copy creates a new collection but copies only the object references, so both the original and copied collections point to the same objects; any change to a mutable object is reflected in both.

Deep copy creates a new collection and also copies the actual objects, so the two collections are completely independent and changes in one do not affect the other.

Que-2)How do you sort a collection?

In Java, you can sort a collection easily. If you want the items in ascending order, you can use Collections.sort(). If you want a custom order or descending order, you can give it a Comparator, for example using a lambda expression. Another way is to use the Stream API with sorted(), which also lets you sort and work with the collection in a clean and readable way.

Que-3)How do you convert an array to a list?

```
import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;


public class ArrayToList {

    public static void main(String[] args) {

        String[] arr = {"Java", "Python", "C++"};

        List<String> list = new ArrayList<>(Arrays.asList(arr));
```

```
        System.out.println(list);
    }
}
```

Que-4)How do you convert a list to an array?

```
import java.util.ArrayList;
import java.util.List;

public class ListToArray {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Java");
        list.add("Python");

        String[] arr = list.toArray(new String[0]);

        for (String s : arr) {
            System.out.println(s);
        }
    }
}
```

Que-5)What are lambda expressions?

Lambda expressions are a feature introduced in Java 8 that provide a concise way to represent anonymous functions. They are mainly used to implement functional interfaces, which are interfaces that contain exactly one abstract method. By using lambda expressions, developers

can avoid writing lengthy anonymous class implementations and instead express behavior in a short and readable form. They are commonly used with collections, streams, and multithreading. The basic syntax of a lambda expression is (parameters) -> expression or (parameters) -> { statements; }, where the arrow operator separates the parameters from the body of the function. Overall, lambda expressions help reduce boilerplate code and improve code readability.

Part-B

Que-1)Write a program to find common keys between two HashMaps.

```
import java.util.HashMap;

import java.util.Scanner;


public class CommonKeyHashMap {

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);

		System.out.println("Enter a N: ");

		int n = sc.nextInt();

		HashMap<String, Integer> map = new HashMap<>();

		for (int i = 0; i < n; i++) {

			System.out.println("Enter a key: ");

			String key = sc.next();

			System.out.println("Enter a value: ");

			int value = sc.nextInt();

			map.put(key, value);


		}
```

```
                System.out.println("Enter a M: ");
                int m = sc.nextInt();
                HashMap<String, Integer> map2 = new HashMap<>();
                for (int i = 0; i < m; i++) {
                        System.out.println("Enter a key: ");
                        String key = sc.next();
                        System.out.println("Enter a value: ");
                        int value = sc.nextInt();
                        map2.put(key, value);

                }

                for (String key : map.keySet()) {
                        if (map2.containsKey(key)) {
                                System.out.println("Common key: " + key);
                        }
                }
        }
}
```

Que-2)Write a program to remove a key from HashMap.

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

public class RemoveKeyHashMap {
```

```
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter a N: ");
		int n = sc.nextInt();
		HashMap<String, Integer> map = new HashMap<>();
		for (int i = 0; i < n; i++) {
			System.out.println("Enter a key: ");
			String key = sc.next();
			System.out.println("Enter a value: ");
			int value = sc.nextInt();
			map.put(key, value);

		}

		System.out.println("Enter a key to remove: ");
		String removeKey = sc.next();

		if (map.containsKey(removeKey)) {
			map.remove(removeKey);
			System.out.println("Key removed successfully.");
		} else {
			System.out.println("Key not found in the HashMap.");
		}
	}
}
```

Que-3)Write a program to check if a key exists in a HashMap.

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

public class CheckKeyHashMap {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter a N: ");
		int n = sc.nextInt();
		HashMap<String, Integer> map = new HashMap<>();
		for (int i = 0; i < n; i++) {
			System.out.println("Enter a key: ");
			String key = sc.next();
			System.out.println("Enter a value: ");
			int value = sc.nextInt();
			map.put(key, value);

		}

		System.out.println("Enter a key: ");
		String key = sc.next();

		if (map.containsKey(key))
			System.out.println("Key Exists!!.");
		else
			System.out.println("Key not found !!.");
```

```
	}
}
```

Que-4)Write a program to check if a value exists in a HashMap.

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

public class CheckValueHashMap {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter a N: ");
		int n = sc.nextInt();
		HashMap<String, Integer> map = new HashMap<>();
		for (int i = 0; i < n; i++) {
			System.out.println("Enter a key: ");
			String key = sc.next();
			System.out.println("Enter a value: ");
			int value = sc.nextInt();
			map.put(key, value);

		}

		System.out.println("Enter a value: ");
		int value = sc.nextInt();

		if (map.containsValue(value))
```

```
                System.out.println("value Exists!!.");
            else
                System.out.println("value not found !!.");

        }
}
```

Que-5)Write a program to demonstrate inheritance.

Inheritance is which one class acquires the properties and behaviors of another class using the extends keyword.

```
class Animal {
   void eat() {
      System.out.println("Animal is eating");
   }
}

class Dog extends Animal {
   void bark() {
      System.out.println("Dog is barking");
   }
}

public class InheritanceDemo {
   public static void main(String[] args) {
```

```
        Dog d = new Dog();
        d.eat();
        d.bark();
    }
}
```