

Assingment 14.

Part A:

1. How does HashMap work internally?

A HashMap is a data structure in Java that stores elements as key-value pairs and provides fast insertion, deletion, and retrieval operations.

Internally, HashMap uses an array of buckets along with a hashing mechanism.

When a key-value pair is inserted using `put(key, value)`, Java calls the `key.hashCode()` method to determine the bucket index.

If multiple keys map to the same bucket (collision), they are stored using a `LinkedList` or a balanced tree (Red-Black Tree in Java 8+).

When retrieving data using `get(key)`, Java calculates the hash code again, finds the bucket, and searches for the correct key to return its value.

The average time complexity is $O(1)$.

2. What is load factor in HashMap?

The load factor determines how full a HashMap can become before it is resized.

It is calculated as $\text{Load Factor} = \text{Size} / \text{Capacity}$.

The default load factor is 0.75.

When the number of elements exceeds 75% of the capacity, HashMap increases its size and rehashes existing elements. This process is called rehashing and helps maintain performance efficiency.

3. What are generics in Java?

Generics allow classes, interfaces, and methods to operate on specific data types while providing code reusability and type safety.

Introduced in Java 5, generics use angle brackets `<>` to define the type.

For example, `ArrayList<String>` ensures that only String values can be stored in the list.

4. Why are generics used?

Generics are used to ensure type safety, avoid explicit type casting, detect errors at compile time, and improve code reusability.

They make Java programs safer and easier to maintain by enforcing data type restrictions during compilation.

5. What is type safety?

Type safety ensures that only the correct type of data is stored in variables or collections.

It prevents runtime errors such as `ClassCastException` by detecting mistakes during compilation.

For example, an `ArrayList<Integer>` can store only integer values.

Part B:

1. Iterate HashSet using for-each loop

```
import java.util.HashSet;

public class IterateHashSet {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<>();
        set.add("Java");
        set.add("Python");
        set.add("C++");

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

2. Convert HashSet to ArrayList

```
import java.util.HashSet;
import java.util.ArrayList;

public class ConvertSetToList {
    public static void main(String[] args) {
        HashSet<Integer> set = new HashSet<>();
        set.add(10);
        set.add(20);
        set.add(30);

        ArrayList<Integer> list = new ArrayList<>(set);
    }
}
```

```

        System.out.println(list);
    }
}

```

3. Union and Intersection using Set

```
import java.util.HashSet;
```

```

public class UnionIntersection {
    public static void main(String[] args) {
        HashSet<Integer> set1 = new HashSet<>();
        set1.add(1);
        set1.add(2);
        set1.add(3);

        HashSet<Integer> set2 = new HashSet<>();
        set2.add(2);
        set2.add(3);
        set2.add(4);

        HashSet<Integer> union = new HashSet<>(set1);
        union.addAll(set2);
        System.out.println("Union: " + union);

        HashSet<Integer> intersection = new HashSet<>(set1);
        intersection.retainAll(set2);
        System.out.println("Intersection: " + intersection);
    }
}

```

4. Check if HashSet contains an element

```
import java.util.HashSet;
```

```

public class ContainsExample {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<>();
        set.add("Apple");
        set.add("Banana");
        set.add("Mango");

        if (set.contains("Banana")) {
            System.out.println("Element found");
        } else {
            System.out.println("Element not found");
        }
    }
}

```

```
}  
}
```

5. Create and iterate a HashMap

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

```
import java.util.Map;
```

```
public class HashMapExample {  
    public static void main(String[] args) {  
        HashMap<Integer, String> map = new HashMap<>();  
        map.put(1, "Nitin");  
        map.put(2, "Rahul");  
        map.put(3, "Amit");  
  
        for (Map.Entry<Integer, String> entry : map.entrySet()) {  
            System.out.println(entry.getKey() + " " + entry.getValue());  
        }  
    }  
}
```