Collection Interface & Collections Class

- Collection Interface
  - It is a root interface in the java.util package which represents group of objects as a single unit.
  - It is an interface, which means it defines set of methods that any collection class(like arraylist, hashset, treeset,etc) must implement.
  - Provides methods for basic operations
    - # add(E e)
    - # remove(Object o)
    - # size()

- Collections Class
  - It is a utility class in java.util package which provides static methods for performing operations on collections.
  - It is a class, not an interface
  - Since it is final class, it cannot be inherited
  - Collections (Java SE 21 & JDK 21)
  - Provides methods like sort(), reverse(), shuffle(), min(), max(), copy(), replaceAll()

1. Collections.shuffle(List<?> list)

Purpose: Randomly shuffles the elements in a list.

```
package collectionsMethodsExamples;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class CollectionsShuffleExample {
    public static void main(String[] args) {
        List<String> cards = Arrays.asList("Ace", "King",
"Jack", "Queen");

        System.out.println("Before Shuffling: "+ cards);

        //Shuffling elements of a list
        Collections.shuffle(cards);

        System.out.println("After Shuffling: "+ cards);
    }
}
```

O/P:

```
Before Shuffling: [Ace, King, Jack, Queen]

After Shuffling: [Queen, Jack, King, Ace]
```

2. Collections.min() & Collections.max()

Purpose: It returns the minimum and maximum elements from a collection.

```
package collectionsMethodsExamples;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class CollectionsMinMaxExample {
    public static void main(String[] args) {
        List<Integer> numbers =
Arrays.asList(10,2,20,40,57,68);
```

```
        int min = Collections.min(numbers);
        int max = Collections.max(numbers);

        System.out.println("Minimum: "+ min);
        System.out.println("Maximum: "+ max);

    }
}
```

o/p:

Minimum: 2

Maximum: 68

3. Collections.copy(List<?> dest, List<?> src)

Purpose: Copies the elements from one list to another.
The destination list should have the same size as the source list

```
package collectionsMethodsExamples;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class CollectionsCopyExample {
    public static void main(String[] args) {
        List<String> source = Arrays.asList("A","B","C");

        //Destination list must have the same size
        List<String> destination = new
ArrayList<>(Arrays.asList("","",""));

        Collections.copy(destination,source);

        System.out.println("Source: "+ source);

        System.out.println("Destination: "+destination);
    }
}
```

```
o/p:
Source: [A, B, C]
Destination: [A, B, C]
```

```java
package collectionsMethodsExamples;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class CollectionsCopyExample {
    public static void main(String[] args) {
        List<String> source = Arrays.asList("A","B","C");

        //Destination list must have the same size
        List<String> destination = new ArrayList<>(source);

       // Collections.copy(destination,source);

        System.out.println("Source: "+ source);

        System.out.println("Destination: "+destination);
    }
}
```

```java
package collectionsMethodsExamples;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class Example {
    public static void main(String[] args) {
        List<Integer> source = Arrays.asList(1,2,3);

        //Destination list must have the same size
        List<Integer> destination = new
ArrayList<>(Arrays.asList(0,0,0));

        Collections.copy(destination,source);

        System.out.println("Source: "+ source);

        System.out.println("Destination: "+destination);
    }
}
```

4. Collections.replaceAll(List<T>list,T old value, T new V)

   Purpose: Replaces all occurrences of a specific element with a new element.

```
package collectionsMethodsExamples;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class ReplaceAllExample {
    public static void main(String[] args) {
        List<String> names =
Arrays.asList("Krishna","Gopal","Govind");

        System.out.println("Real(Before Replacement): "+
names);

        Collections.replaceAll(names,
"Krishna","KrishnaYadav");

        System.out.println("After Replacement: "+ names);
    }
}
```

o/p:
Real(Before Replacement): [Krishna, Gopal, Govind]
After Replacement: [KrishnaYadav, Gopal, Govind]

Collections.synchronizedList(List<T> list)
It will return the thread safe version of the given list
Collections.synchronizedMap()
Collections.synchronizedSet()

Sort() - sorts the elements in ascending order
Reverse() - reverses the order of elements

Object Class- Object (Java SE 21 & JDK 21)

- The Object class is the root of the class hierarchy in java
- It is the part of java.lang
- toString(),equals(),getClass()

toString()

Purpose: It returns the string representation of an object

```java
package objectClass;

public class Employee {
    int id;
    String name;

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }

    public static void main(String[] args) {
        Employee emp = new Employee(101,"Krishna");
        System.out.println(emp);
    }
}
```

O/P:

Employee{id=101, name='Krishna'}

getClass()

Returns the runtime class of the object.

```
package objectClass;

public class Test {
    public static void main(String[] args) {
        String str = "Flynaut";
        System.out.println("ClassName: "+
str.getClass().getName());
    }
}
```

o/p:

ClassName: java.lang.String

equals(Object obj)

Compares two objects to check for the equality

Its default implementation checks if the references are the same(==)

We can give custom implementation by overriding this method(Override to compare the values(content))

```
package objectClass;

public class Employee {
    int id;
    String name;

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public boolean equals(Object obj){
        if (this == obj) return true; // same reference check
        if (obj == null || getClass() != obj.getClass())
return false;

        Employee employee = (Employee)obj;
        return id == employee.id &&
name.equals(employee.name);
    }

    public static void main(String[] args) {
        Employee emp1 = new Employee(101,"Krishna");
        Employee emp2 = new Employee(101,"Krishna");

        System.out.println("Are they Equal: "+
emp1.equals(emp2));
    }
}
```

o/p:

```
Are they Equal: true
```