|  |  |  |
| --- | --- | --- |
| Aspect | Collection Interface | Collections Class |
| Definition | A root interface in the java.util package that represents a group of objects as a single unit. | A utility class in the java.util package that provides static methods for manipulating collections |
| Type | Interface | Class |
| Inheritance | It is implemented by various collection classes like ArraList, HashSet, TreeSet, etc | It cannot be inherited as it is final. |
| Purpose | Provides methods for adding, removing, and iterating elements in a collection. | Offers utility methods like sort(), reverse(), binarysearch(), etc for common collection operations |

Methods of Collections utility class:

* sort(List<T> list)
* reverse(List<?> list)
* shuffle(List<?> list)
* package CollectionsExampe;  
    
  import java.util.Arrays;  
  import java.util.Collections;  
  import java.util.List;  
    
  public class CollectionShuffleExample {  
   public static void main(String[] args) {  
   List<String> cards = Arrays.*asList*("Ace", "King", "Queen", "Jack");  
   System.*out*.println("Before Shuffling: "+ cards);  
   Collections.*shuffle*(cards);  
   System.*out*.println("After Shuffling: "+ cards);  
   }  
  }
* Min() & Max()

package CollectionsExampe;  
  
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*(10,2,20,59,55);  
 int min = Collections.*min*(numbers);  
 int max = Collections.*max*(numbers);  
 System.*out*.println("Minimum: "+ min);  
 System.*out*.println("Maximum: "+ max);  
 }  
}

[Arrays (Java SE 21 & JDK 21)](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Arrays.html)

* Copy()
* package CollectionsExampe;  
    
  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");  
   List<String> destination = new ArrayList<>(Arrays.*asList*("","",""));  
   Collections.*copy*(destination,source);  
   System.*out*.println("Source : "+source);  
   System.*out*.println("Destination : "+destination);  
   }  
  }

package CollectionsExampe;  
  
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<Integer> source = Arrays.*asList*(1,2,3);  
 List<Integer> destination = new ArrayList<>(Arrays.*asList*(0,0,0));  
 Collections.*copy*(destination,source);  
 System.*out*.println("Source : "+source);  
 System.*out*.println("Destination : "+destination);  
 }  
}

package CollectionsExampe;  
  
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<Integer> source = Arrays.*asList*(1,2,3);  
 List<Integer> destination = new ArrayList<>(source));  
 Collections.*copy*(destination,source);  
 System.*out*.println("Source : "+source);  
 System.*out*.println("Destination : "+destination);  
 }  
}

* replaceAll()
* package CollectionsExampe;  
    
  import java.util.Arrays;  
  import java.util.Collections;  
  import java.util.List;  
    
  public class CollectionsReplaceAllExample {  
   public static void main(String[] args) {  
   List<String> names = Arrays.*asList*("Krishna", "Govind", "Gopal");  
   System.*out*.println("Names :" +names);  
   Collections.*replaceAll*(names,"Krishna","Krish");  
   System.*out*.println("After replacing Names :" +names);  
   }  
  }
* synchronizedList(List<T> list),synchronizedSet(Set<T> set)

[Collections (Java SE 21 & JDK 21)](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Collections.html)

Object Class:

* It is the root of the class hierarchy
* Every class is either directly or indirectly derived from the object class
* It is part of java.lang package
* It provides methods that are common to all java objects

Methods:

1. toString()

returns the string representation of the object

1. equals(Object obj)

Compares the given object with the current object

package ObjectClass;  
  
public class Employee {  
 int id;  
 String name;  
  
 Employee(int id,String name){  
 this.id=id;  
 this.name=name;  
 }  
  
 @Override  
 public boolean equals(Object obj){  
 if(this == obj) return true;  
 if(obj == null || getClass() != obj.getClass()) return false;  
 Employee employee = (Employee) obj;  
 return id == employee.id && name.equals(employee.name);  
 }  
}

package ObjectClass;  
  
public class Main {  
 public static void main(String[] args) {  
 Employee emp1 = new Employee(101, "Krishna");  
 Employee emp2 = new Employee(101, "Krishna");  
 System.*out*.println("True? "+emp1.equals(emp2));  
 }  
}

* wait()
* notify()
* notifyAll()
* getClass() - Returns the runtime class of this Object.
* MultiThreading:

What is Multithreading?

* A thread is a lightweight subprocess, the smallest unit of the processing.

Benefits:

* Efficient CPU utilization
* Concurrency
* Responsive Applications

[Thread (Java SE 21 & JDK 21)](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Thread.html)

[Runnable (Java SE 21 & JDK 21)](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Runnable.html)

Thread Class And Runnable Interface

Java provides two main ways to create a thread

1. Extending the thread class
2. Implementing the Runnable interface
3. Extending the thread class
4. package ThreadExamples;  
     
   public class ThreadExampl {  
    public static void main(String[] args) {  
    MyThread thread1 = new MyThread();  
    MyThread thread2 = new MyThread();  
     
    thread1.start(); // Starts the thread and calls the run() method.  
    thread2.start();  
      
    }  
   }
5. package ThreadExamples;  
     
   public class MyThread extends Thread{  
    public void run(){  
    System.*out*.println("Thread is running : "+ Thread.*currentThread*().getName());  
    }  
   }

Loose Coupling with runnable interface

Using the Runnable interface decouples the task(Logic in run()) from the thread(which manages the execution).

The task can exist independently of the thread, making it reusable and more flexible.

Since our class doesn’t extend Thread, its free to extend any other classes.

Thread Lifecycle:\*