(Concept && Coding) Java Collections: Page1

Java Collections Framework
(Introduction)

- **What is Java Collections Framework?**
  - Added in Java version 1.2
  - Collection is a building block of group of Objects.
  - Collection is a single unit of Objects.
  - Framework provide us the architecture to manage those "group of objects" i.e. add, update, delete, search etc.

Why we need Java Collections Framework?

- Prior to JDK1 we have Array, Vector, Hash tables.
  But problem with them is there is no commonly interface, so its difficult to remember the methods for each.

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

        int[] a = {4,5,1,9,2,11};
        Arrays.sort(a, (int[]var1, int[]var2) -> var1[-v1+1]);

        for(int v : a){
            System.out.println(v);
        }
    }
}
```

Output:

```
11
2
1
4
5
9
```

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

        Integer a[] = {4,5,1,9,2,11};
        Arrays.sort(a, (int[]var1, int[]var2) -> var1[-v1+1]);

        for(int v : a){
            System.out.println(v);
        }
    }
}
```

Output:

```
11
9
5
4
2
1
```

```
public class Car {

    String carName;
    String carType;

    Car(String name, String type){
        this.carName = name;
        this.carType = type
    }
}

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

        Car[] cars = new Car[3];
        cars[0] = new Car("Ford", "SUV");
        cars[1] = new Car("BMW-750xi", "SUV");
        cars[2] = new Car("Ford", "Hatchback", "Sub-Type");

        Arrays.sort(cars, (car v1, car v2) -> v1[0].cars.compareTo(v2[0].carTypeS);

        for(Car car : cars){
            System.out.print(car.carName + " - " + car.carTypeS);
        }
    }
}
```

Output:

Hope You liked the creativity (presentation).

Presented By : Rajkishor
Dedicated To : Thanks For Joining the session

Some people get it, some don't but life keeps moving

Find the meaning if you don't get it.

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

      List<Car> cars = new ArrayList<>();
      cars.add(new Car("suv", "petrol"));
      cars.add(new Car("sedan", "diesel"));
      cars.add(new Car("hatchback", "cng"));

      Collections.sort(cars, Car::getCarType);

      cars.forEach(Car car0 -> System.out.printf("%s..%s%n", car0.carName, car0.carType));

  }
}
```

Output:

```
suv..petrol
sedan..diesel
hatchback..cng
```

```
public class Car {

    String carName;
    String carType;

    Car(String name, String type){
      this.carName = name;
      this.carType = type;
    }
}

public class CarNameComparator implements ComparatorCar<> {

    @Override
    public int compare(Car s1, Car s2) {
        return s1.carName.compareTo(s2.carName);
    }
}

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

      List<Car> cars = new ArrayList<>();
      cars.add(new Car("sedan", "diesel"));
      cars.add(new Car("suv", "petrol"));
      cars.add(new Car("hatchback", "cng"));

      Collections.sort(cars, new CarNameComparator());

      cars.forEach(Car car0 -> System.out.printf("%s..%s%n", car0.carName, car0.carType));

  }
}
```

Output:

```
suv..petrol
sedan..diesel
hatchback..cng
```

```
public class Car implements Comparator<Car> {

    String carName;
    String carType;

    Car(){

    }
    Car(String name, String type){
      this.carName = name;
      this.carType = type;
    }

    @Override
    public int compare(Car s1, Car s2) {
        return s1.carName.compareTo(s2.carName);
    }
}

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

      List<Car> cars = new ArrayList<>();
      cars.add(new Car("sedan", "diesel"));
      cars.add(new Car("suv", "petrol"));
      cars.add(new Car("hatchback", "cng"));

      Collections.sort(cars, new Car());

      cars.forEach(Car car0 -> System.out.printf("%s..%s%n", car0.carName, car0.carType));

  }
}
```

Output:

```
hatchback..cng
sedan..diesel
suv..petrol
```

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

      List<Car> cars = new ArrayList<>();
      cars.add(new Car("suv", "petrol"));
      cars.add(new Car("sedan", "diesel"));
      cars.add(new Car("hatchback", "cng"));

      Collections.sort(cars, Car::getCarType);

      cars.forEach(Car car0 -> System.out.printf("%s..%s%n", car0.carName, car0.carType));

  }
}
```

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

      List<Car> cars = new ArrayList<>();
      cars.add(new Car("suv", "petrol"));
      cars.add(new Car("sedan", "diesel"));
      cars.add(new Car("hatchback", "cng"));

      Collections.sort(cars, new CarNameComparator());

      cars.forEach(Car car0 -> System.out.printf("%s..%s%n", car0.carName, car0.carType));

  }
}
```

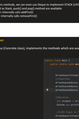
```
public class Main {
  public static void main(String[] args) {

      List<Car> cars = new ArrayList<>();
      cars.add(new Car("suv", "petrol"));
      cars.add(new Car("sedan", "diesel"));
      cars.add(new Car("hatchback", "cng"));

      Collections.sort(cars, new Car());

      cars.forEach(Car car0 -> System.out.printf("%s..%s%n", car0.carName, car0.carType));

  }
}
```



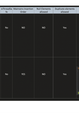
```
public class Main {
  public static void main(String[] args) {

      List<Car> cars = new ArrayList<>();
      cars.add(new Car("suv", "petrol"));
      cars.add(new Car("sedan", "diesel"));
      cars.add(new Car("hatchback", "cng"));

      Collections.sort(cars, Car::getCarType);

      cars.forEach(Car car0 -> System.out.printf("%s..%s%n", car0.carName, car0.carType));

  }
}
```



Comparison

Concurrent	No	Yes	Yes	Yes	

```
public class Test {
    public static void main(String[] args) {
        ConcurrentHashMap<Integer, String> map = new ConcurrentHashMap<>();
        map.put(1, "A");
        System.out.println(map.get(1));
    }
}
```

Vector

Vector is a legacy synchronized implementation of List.
Every method is synchronized, meaning only one thread can access the vector at a time.
This makes it safe for multi-threaded environments.
Slow in single-threaded applications because of the overhead of synchronization.
Use Collections.synchronizedList() or CopyOnWriteArrayList instead for better performance.

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

        Vector<Integer> v = new Vector<>();
        v.add(10);
        v.add(20);

        System.out.println(v.get(0));

    }
}
```

Operation	Time Complexity	Thread Safe	Allows Duplicates	Allows Null Elements	Preserves Insertion Order	Implementation
Insert	O(1)	Yes	Yes	Yes	Yes	N/A

Stack

- **Represents LIFO (Last In First out) operations**
- **Since it extends Vector, its method is also Synchronized.**

How its different from Deque : Deque is not thread safe, stack is.

```
public class StackExample {

    public static void main(String[] args) {

        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);

        System.out.println(stack.pop());

    }
}
```

Time Complexity:

- Push(add) : O(1)
- Pop(remove) : O(1)
- Peek(get) : O(1)
- Search: O(n)

Space Complexity:
O(n)

Operation	Time Complexity	Thread Safe	Allows Duplicates	Allows Null Elements	Preserves Insertion Order	Implementation
Push	O(1)	Yes	N/A	Yes	Yes	N/A