1. 动态数组容器泛型类DynamicArray

import java.util.Arrays;

public class DynamicArray<T> {

    private static final int DEFAULT\_CAPACITY = 10;

    private Object[] elementData;

    private int size;

    public DynamicArray() {

        this.elementData = new Object[DEFAULT\_CAPACITY];

        this.size = 0;

    }

    private void ensureCapacity(int *minCapacity*) {

        if (*minCapacity* > elementData.length) {

            int newCapacity = Math.max(elementData.length \* 2, *minCapacity*);

            elementData = Arrays.copyOf(elementData, newCapacity);

        }

    }

    public void add(T *t*) {

        ensureCapacity(size + 1);

        elementData[size++] = *t*;

    }

    @SuppressWarnings("unchecked")

    public T get(int *index*) {

        if (*index* < 0 || *index* >= size) {

            throw new IndexOutOfBoundsException("Index out of bounds: " + *index*);

        }

        return (T) elementData[*index*];

    }

    public int size() {

        return size;

    }

    public void set(int *index*, T *item*) {

        if (*index* < 0 || *index* >= size) {

            throw new IndexOutOfBoundsException("Index out of bounds: " + *index*);

        }

        elementData[*index*] = *item*;

    }

    public void addAll(DynamicArray<? extends T> *other*) {

        for (int i = 0; i < *other*.size(); i++) {

            add(*other*.get(i));

        }

    }

    public static <T> void swap(DynamicArray<T> *arr*, int *i*, int *j*) {

        if (*i* < 0 || *i* >= *arr*.size() || *j* < 0 || *j* >= *arr*.size()) {

            throw new IndexOutOfBoundsException("Index out of bounds");

        }

        T tmp = *arr*.get(*i*);

*arr*.set(*i*, *arr*.get(*j*));

*arr*.set(*j*, tmp);

    }

    public void copy(DynamicArray<? extends T> *source*) {

        elementData = Arrays.copyOf(*source*.elementData, *source*.size);

        size = *source*.size;

    }

    public void copyTo(DynamicArray<? super T> *dest*) {

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

*dest*.add(this.get(i));

        }

    }

    @Override

    public String toString() {

        if (size == 0) {

            return "[]";

        }

        StringBuilder sb = new StringBuilder("[");

        for (int i = 0; i < size - 1; i++) {

            sb.append(elementData[i]).append(", ");

        }

        sb.append(elementData[size - 1]).append("]");

        return sb.toString();

    }

    public static void main(String[] *args*) {

        DynamicArray<Integer> ints = new DynamicArray<>();

        DynamicArray<Number> numbers = new DynamicArray<>();

        ints.add(100);

        ints.add(34);

        ints.copyTo(numbers);

        System.out.println(numbers);

    }

}

1. GMethodTest类

使用removeDuplicates方法实现数组去重时，会对数组中的元素进行比较，在比较是会调用equals方法来判断是否相等，对于基本类型，无需重写该方法，但自定义的类Circle就需要重写equals来进行相等的判断。

在本例中，Cirlce为自定义的类，ArrayList<Circle>中存放的是Circle对象的地址，如果不重写equals方法，将会比较这些地址是否相等，这显然不符合逻辑，故重写equals方法，通过判断Circle对象的radius是否相等来决定对象是否相等。

使用max方法求取最大值时，对于自定义的类Circle，需要实现Comparable<Circle>接口中的compareTo方法才能进行比较，否则会抛出编译异常，在该方法中，比较两个Circle对象的radius来判断大小。

import java.util.ArrayList;

*/\*\**

*\* Circle需要重写compareTo来实现比较大小*

*\* 需要重写equals来实现去重*

*\*/*

class Circle implements Comparable<Circle> {

    private double radius;

    public Circle(double *radius*) {

        this.radius = *radius*;

    }

    @Override

    public String toString() {

        return "Circle radius: " + radius;

    }

    public int compareTo(Circle *other*) {

        return Double.compare(this.radius, *other*.radius);

    }

    @Override

    public boolean equals(Object *o*) {

        if (this == *o*)

            return true;

        if (*o* == null || getClass() != *o*.getClass())

            return false;

        Circle circle = (Circle) *o*;

        return Double.compare(circle.radius, radius) == 0;

    }

}

public class GMethodTest {

    public static <E> ArrayList<E> removeDuplicates(ArrayList<E> *list*) {

        ArrayList<E> newList = new ArrayList<>();

        for (E item : *list*) {

            if (!newList.contains(item)) {

                newList.add(item);

            }

        }

        return newList;

    }

    public static <E extends Comparable<E>> E max(E[] *list*) {

        if (*list* == null || *list*.length == 0) {

            throw new IllegalArgumentException("Input array is empty");

        }

        E maxElement = *list*[0];

        for (int i = 1; i < *list*.length; i++) {

            if (*list*[i].compareTo(maxElement) > 0) {

                maxElement = *list*[i];

            }

        }

        return maxElement;

    }

    public static void main(String[] *args*) {

        ArrayList<Integer> list = new ArrayList<>();

        ArrayList<Circle> circles = new ArrayList<>();

        list.add(14);

        list.add(24);

        list.add(14);

        list.add(12);

        circles.add(new Circle(3));

        circles.add(new Circle(2.9));

        circles.add(new Circle(5.9));

        circles.add(new Circle(3));

        ArrayList<Integer> newList = removeDuplicates(list);

        ArrayList<Circle> newCircles = removeDuplicates(circles);

        System.out.println(newList); *// 输出 [14, 24, 12]*

        System.out.println(newCircles); *// 输出 [Circle radius: 3.0, Circle radius: 2.9, Circle radius: 5.9]*

        Integer[] numbers = { 1, 2, 3 };

        System.out.println(max(numbers)); *// 输出 3*

        Circle[] c = { new Circle(3), new Circle(2.9), new Circle(5.9) };

        System.out.println(max(c)); *// 输出 Circle radius: 5.9*

    }

}