Plan

2023-11-08

1. Homework Review
2. Projects: Game21, Forum

3. Collection, собственная реализация ArrayList

1. Разбор домашнего задания
2. Проекты: Game21, Forum
3. Collection, собственная реализация ArrayList

Theory

▶ **English**

▼ **На русском**

- Java Collection Framework (JCF) - множество классов и интерфейсов которые реализуют наиболее часто используемые структуры данных. JCF состоит из двух больших подразделов: Map и Collection. Мы начинаем наше изучение с коллекций.
- Интерфейс Collection расширяет интерфейс Iterable, т. е. все коллекции итерируемые. Интерфейс Collection определяет некоторый основной набор методов для работы с коллекциями данных. Например добавление, удаление, поиск, получение колличества элементов в коллекции и т. д.
- Есть множество интерфейсов расширяющих интерфейс Collection. Мы рассмотрим интерфейсы Set и List. И начнем с интерфейса List. Интерфейс List определяет коллекции элементы которых имеют индексы, т. е. некий аналог массива, но не имеющий ограничения по размеру. Соответственно в интерфейсе List, помимо методов унаследованных от Iterable и Collection, определены методы работающие с индексами. Например вставка по индексу, удаление по индексу, получение элемента по индексу, поиск индекса заданного аргумента и т. п.
- Одной из имплементаций интерфейса List является класс ArrayList. Для реализации функциональности интерфейса List,
- ArrayList инкапсулирует в себе массив некоторого начального размера. Когда этот массив полностью заполняется, то его элементы копируются в новый массив, но уже большего размера. И теперь ждем когда заполнится новый массив. И т. д. Т. е. простым языком ArrayList представляет из себя "резиновый массив".

Реализация собственной версии ArrayList

Для практического занятия мы можем взять за основу простую структуру данных, аналогичную ArrayList в Java, которую назовем MyArrayList. Вот базовый контур класса для реализации:

```
public interface MyList<E> {

    // Добавляем элемент и увеличиваем размер массива, если нужно
    void add(E o);

    // Получаем элемент по индексу
    E get (int index);

    // устанавливает объект по индексу, смещая объекты
    void set(E o, int index);

    // Возвращаем размер коллекции
    int size();
    boolean contains(E o);

    // Удаляем элемент по значению
    boolean remove(E o);

    // Удаляем элемент по индексу
    E removeByIndex(int index);
}
```

- Когда элементы добавляются в ArrayList и его текущая емкость заполняется, ArrayList должен увеличить свой размер, чтобы вместить больше элементов. Это происходит за счет создания нового массива большего размера и копирования элементов из старого массива в новый.
- Процесс увеличения размера называется "расширением" (resizing) или "перераспределением" (reallocating), и хотя он относительно эффективен, он может быть дорогостоящим с точки зрения производительности при добавлении большого количества элементов, так как при каждом расширении происходит копирование всех элементов. Поэтому рекомендуется, если известно количество элементов или примерный верхний предел, инициализировать ArrayList с этой начальной емкостью:

```
List<String> list = new ArrayList<>(начальная_емкость);
```

Это позволит избежать лишних расширений и увеличить производительность при добавлении большого количества элементов.

Код с урока в github:

- https://github.com/AR1988/Ait-342ArrayList
- https://github.com/AR1988/Game21_AIT

Homework

▶ English

▼ На русском

- дописать методы MyArrayList и тесты к ним

Code

code/Game21_AIT/src/Card.java

```
import java.util.Objects;

/**
 * @author Andrej Reutow
 * created on 08.11.2023
 */
public class Card implements Comparable<Card> {
    /**
     * Значение карты от 1 до 10.
     */
    private final int VALUE;
    /**
     * масть карты.
     */
    private final String SUIT;

    /**
     * Конструктор для инициализации карты с указанными значениями.
     */
    public Card(int value, String suit) {
        this.VALUE = value;
        this.SUIT = suit;
    }

    public int getVALUE() {
        return VALUE;
```

```
    }

    public String getSUIT() {
        return SUIT;
    }

    @Override
    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false

        Card card = (Card) object;

        if (VALUE != card.VALUE) return false;
        return Objects.equals(SUIT, card.SUIT);
    }

    @Override
    public int hashCode() {
        int result = VALUE;
        result = 31 * result + (SUIT != null ? SUIT.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return SUIT + " " + VALUE;
    }

    @Override
    public int compareTo(Card other) {
        int suitCompare = this.SUIT.compareTo(other.getSUIT());
        return suitCompare != 0 ? suitCompare : Integer.compare(this.VALUE,
    }
}
```

code/Game21_AIT/src/Deck.java

```
import java.util.Iterator;
import java.util.Random;

/**
 * @author Andrej Reutow
 * created on 08.11.2023
```

```
 */
// колода
public class Deck implements Iterable<Card> {

    // колода карт
    private Card[] cards;

    /**
     * Конструктор для создания колоды всех карт. Всего ровно 36 карт
     */
    public Deck() {
        this.cards = new Card[36];
        fillCards();
        shuffle();
    }

    public void fillCards() {
        String[] suits = {"Пики", "Червы", "Бубны", "Трефы"};
        int index = 0;
        for (String suit : suits) {
            for (int cardValue = 2; cardValue <= 10; cardValue++) {
                this.cards[index++] = new Card(cardValue, suit);
            }
        }

//        for (int i = 0, suitsLength = suits.length; i < suitsLength; i++)
//            String suit = suits[i];
//            for (int cardValue = 2; cardValue <= 10; cardValue++) {
//                this.cards[index++] = new Card(cardValue, suit);
//            }
//        }
    }

    /**
     * Метод shuffle() для перемешивания колоды.
     */
    public void shuffle() {
        Random random = new Random();
        for (int i = cards.length - 1; i > 0; i--) {
            int index = random.nextInt(i + 1);

            Card temp = cards[index];   // сохраняем элемент во временную п
            cards[index] = cards[i];    // заменяю элемент по индексу на те
```

```
                cards[i] = temp;                // заменяю выбраранный случайный эл
            }
        }

        public Card[] getCards() {
            return cards;
        }

        /**
         * Реализация интерфейса Iterator/Iterable для перебора карт в колоде.
         */
        @Override
        public Iterator<Card> iterator() {
            return new Iterator<>() {
                private int curPos;

                @Override
                public boolean hasNext() {
                    return curPos < cards.length;
                }

                @Override
                public Card next() {
                    return cards[curPos++];
                }
            };
        }
    }
```

code/Game21_AIT/src/DeckTest.java

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

/**
 * @author Andrej Reutow
 * created on 08.11.2023
 */
@DisplayName("Тестирование колоды")
class DeckTest {
```

```
    private Deck deck;


    @BeforeEach
    void setUp() {
        deck = new Deck();
    }

    @Test
    public void test_fillArray() {
        deck.fillCards();

        Card[] cards = deck.getCards();

        for (Card card : cards) {
            Assertions.assertNotNull(card);
        }
    }

    @Test
    public void test_shuffle() {
//        deck.fillCards();
//        Card[] cards = deck.getCards();

//        deck.shuffle();
        Card[] shuffleCards = deck.getCards();

        System.out.println();
    }
}
```

code/Game21_AIT/src/Game21.java

```
import java.util.Iterator;
import java.util.Scanner;

/**
 * @author Andrej Reutow
 * created on 09.11.2023
 */
public class Game21 {

    public static void main(String[] args) {
        Deck deck = new Deck();
```

```
        Scanner scanner = new Scanner(System.in);

        Player player = getPlayer(scanner);
        System.out.println("Добро пожаловать " + player.getName());
        System.out.println("Игра начинается\n");

        // игровой процесс
        startGame(scanner, player, deck);

        scanner.close();
    }

    private static void startGame(Scanner scanner, Player player, Deck deck
        final Iterator<Card> deckIterator = deck.iterator();

        int roundCounter = 1;

        while (deckIterator.hasNext()) {

            System.out.println("Старт раунд: " + roundCounter++);
            System.out.println("#".repeat(30));

            String userInput = null;
            do {
                System.out.println("Введите 'y' что бы получить новую карту
                userInput = scanner.nextLine();

                if (userInput.equalsIgnoreCase("y")) {
                    Card card = deckIterator.next();
                    player.drawCard(card);
                    player.displayHandCards();
                }

                if (userInput.equalsIgnoreCase("n")) {
                    if (player.getRoundScore() == 21) {
                        System.out.println("21! Отлично");
                        player.addPointToTotalScore();
                    } else if (player.getRoundScore() > 21) {
                        System.out.println("У вас перебор, " + player.getRo
                    } else if (player.getRoundScore() < 21) {
                        System.out.println("У вас недобор, " + player.getRo
                    }
                    player.resetRound();
```

```
                }
            } while (!"y".equalsIgnoreCase(userInput) && !"n".equalsIgnoreC
        }

        System.out.println("Игра окончена, ваш результат: " + player.getTot
    }

    private static Player getPlayer(Scanner scanner) {
        System.out.println("Введите имя игрока");
        String playerName = scanner.nextLine();
        return new Player(playerName);
    }
}
```

code/Game21_AIT/src/Player.java

```
import java.util.Arrays;

/**
 * @author Andrej Reutow
 * created on 08.11.2023
 */
public class Player {

    private final String NAME;

    /**
     * Карты игрока в текущем раунде
     */
    private Card[] hand;

    /**
     * сумарное значение карт в руках игрока за текущий раунд
     */
    private int roundScore; // 0
    /**
     * Общий счет за всю игру/ количество побед
     */
    private int totalScore;

    // Конструктор для инициализации имени игрока и начального состояния.
    public Player(String name) {
        this.NAME = name;
        this.hand = new Card[0];
```

```
    }

    // Метод drawCard(Card card) для добавления карты в руку и обновления с
    public void drawCard(Card card) {
        Card[] copy = Arrays.copyOf(this.hand, this.hand.length + 1);
        copy[copy.length - 1] = card;
        this.hand = copy;
        Arrays.sort(copy);
        this.roundScore += card.getVALUE();
    }

    // Метод resetRound() для сброса руки и счета за раунд.
    public void resetRound() {
        this.roundScore = 0;
        this.hand = new Card[0];
    }

    // Метод addPointToTotalScore() для добавления балла к общему счету.
    public void addPointToTotalScore() {
        this.totalScore++;
    }

    public void displayHandCards() {
        System.out.println("#".repeat(20));
        System.out.println("Карты игрока:");

        for (Card card : hand) {
            System.out.println(card);
        }

        System.out.println("#".repeat(20));
    }

    public String getName() {
        return NAME;
    }

    public int getTotalScore() {
        return totalScore;
    }

    public int getRoundScore() {
        return roundScore;
```

```
    }
}
```

code/OurArrayList/src/collection/EmptyListException.java

```
package collection;

/**
 * @author Andrej Reutow
 * created on 09.11.2023
 */
public class EmptyListException extends RuntimeException {
}
```

code/OurArrayList/src/collection/OurArrayList.java

```
package collection;

import java.util.Arrays;
import java.util.Comparator;
import java.util.Iterator;

public class OurArrayList<E> implements OurList<E> {

    private Object[] source;
    private static final int INITIAL_CAPACITY = 16;
    private int size;

    public OurArrayList() {
        source = new Object[INITIAL_CAPACITY];
    }

    @Override
    public int size() {
        return size;
    }

    @Override
    public boolean contains(E o) {
        for (Object object : source) {
            if (object.equals(o)) {
                return true;
            }
        }
```

```
        return false;
    }

    @Override
    public void add(E value) {
        if (source.length == size) {
            int newCapacity = size + (size / 2); // Увеличиваем размер в 1.
            source = grow(newCapacity);
        }
        source[size] = value;
        size++;
    }

    private Object[] grow(int newCapacity) {
        int oldCapacity = source.length;

        if (oldCapacity == Integer.MAX_VALUE) {
            // Массив уже достиг максимального размера
            throw new OutOfMemoryError("Array size cannot be increased beyo
        }

        // Проверяем, не приведет ли увеличение к переполнению
        if (newCapacity < 0) {
            // Устанавливаем максимально возможный размер для массива
            newCapacity = Integer.MAX_VALUE;
        }

        return Arrays.copyOf(source, newCapacity);
    }

    @Override
    public E get(int index) {
        if (index >= size || index < 0)
            throw new IndexOutOfBoundsException();

        return (E) source[index];
    }

    @Override
    public void set(E value, int index) {
        if (index >= size || index < 0)
            throw new IndexOutOfBoundsException();
```

```
        source[index] = value;
    }

    @Override
    public boolean remove(E value) {
        for (int i = 0; i < source.length; i++) {
            if (source[i].equals(value)) {
                removeById(i);
                return true;
            }
        }

        return false;
    }

    @Override
    public E removeById(int index) {
        if (index >= size || index < 0)
            throw new IndexOutOfBoundsException();

        E result = (E) source[index];

        System.arraycopy(source, index + 1, source, index, source.length -
        size--;
        return result;
    }

    public E max() {
        if (source.length == 0)
            throw new EmptyListException();

        Object[] copy = new Object[size];
        System.arraycopy(source, 0, copy, 0, size);
        Arrays.sort(copy);

        return (E) copy[copy.length - 1];
    }

    public E min() {
        if (size == 1 || size == 0) {
            return (E) source[0];
        }
        Object[] copy = new Object[size];
```

```
            System.arraycopy(source, 0, copy, 0, size);
            Arrays.sort(copy);
            return (E) copy[0];
        }

        public void sort(Comparator comparator) {
            Arrays.sort(source, comparator);
        }

        @Override
        public Iterator<E> iterator() {
            return new Iterator<E>() {
                int currentEltId = 0;

                @Override
                public boolean hasNext() {
                    return currentEltId < size;
                }

                @Override
                public E next() {
                    return get(currentEltId++);
                }
            };
        }
    }
```

code/OurArrayList/src/collection/OurList.java

```
    package collection;

    /**
     * @author Andrej Reutow
     * created on 09.11.2023
     */
    public interface OurList<E> extends Iterable {
        void add(E o);

        E get(int index);

        void set(E o, int index);

        int size();
```

```
    boolean contains(E o);

    boolean remove(E o);

    E removeById(int index);
}
```