Plan

2023-11-08

1. Homework Review
2. Practice, console game

---

1. Разбор домашнего задания
2. Практическая работа, консольная игра

Theory

▶ **English**

▼ **На русском**

Конечно, вот обновленное техническое задание с учетом добавления поля для общего счета игры:

Техническое задание для консольной игры "21" на Java:

Общее описание игры:

Игра "21" - карточная игра на одного игрока, где цель состоит в том, чтобы собрать карты на сумму 21 как можно большее количество раз.

Классы и функциональность:

1. **Класс `Card`:** - карта
    - Поля:
        - `value` (int) - значение карты от 1 до 10.
        - `suit` (String) - масть карты.
    - Конструктор для инициализации карты с указанными значениями.
    - Геттеры для доступа к полям класса.
2. **Класс `Deck`:** - колода
    - Поля:
        - `cards` (Card[]) - колода карт.
    - Конструктор для создания колоды всех карт. Всего ровно 40 карт
        - 4 масти "Пики", "Червы", "Бубны", "Трефы"
        - значения от 1 до 10

- Метод `shuffle()` для перемешивания колоды.
- Реализация интерфейса Iterator/Iterable для перебора карт в колоде.

3. **Класс `Player`:** - игрок

   - Поля:
     - `name` (String) - имя игрока.
     - `hand` (Card[]) - текущие карты игрока (карты игрока текущего раунда).
     - `score` (int) - счет за текущий раунд.
     - `totalScore` (int) - общий счет игры.
   - Конструктор для инициализации имени игрока и начального состояния.
   - Метод `drawCard(Card card)` для добавления карты в руку и обновления счета `score`.
   - Метод `resetRound()` для сброса руки и счета за раунд.
   - Метод `addPointToTotalScore()` для добавления балла к общему счету.

4. **Игровой процесс:**

   - При старте игры игроку предлагается ввести имя.
   - Создается объект игрока и перемешанная колода карт.
     - Размер массива карт игрока при инициализации игрока и перезапуске игры равен **0** new `Card[0]`;
     - Каждый раз когда игрок получает новую карту, необходимо увеличить размер массива на **1** (смотри метод `drawCard(Card card)`)
     - Массив карт игрока должен быть всегда сортирован по полю `suit` - в алфавитном порядке и `value` - по возрастанию.
   - Игроку предлагается продолжить раунд или завершить его (ввод в консоль "y" для продолжения раунда):
     - При продолжении раунда (если игрок ввел "y") игрок берет карту из колоды. (используйте Iterator в классе `Deck` что бы получить следующую карту из колоды)
     - При завершении раунда (если игрок ввел "e") проверяется счет и, если достигнута сумма 21, к общему счету добавляется балл (метод `addPointToTotalScore()`).
     - После завершения раунда текущие карты и счет сбрасываются (метод `resetRound()`).
   - Игра заканчивается, когда колода карт иссякнет, с выводом общего счета игрока.

5. Ввод и вывод

- для ввода используйте Scanner
- результаты выводите в консоль

Homework

▶ **English**

▼ На русском

- В проекте Game21 написать сортировку карт в руке игрока. Карты должны быть сортированы по полю масть и значение.
- Написать графический интерфейс для игры.
  - Требования к интерфейсу смотрите в разделе теория пункт 4,5

Code

code/HwSolution42/src/playlist/Playlist.java

```
package playlist;

import java.util.Iterator;

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

//Создайте класс Playlist, который реализует Iterable и содержит список пес
// который позволяет переключаться между песнями.
public class Playlist implements Iterable<String> {

    private String[] songs;
    private int size;

    public Playlist(int playListSize) {
        this.songs = new String[playListSize];
    }

    public boolean addSong(String song) {
        songs[size++] = song;
        return true;
    }

    @Override
    public Iterator<String> iterator() {
        return new PlayListIterator(songs);
    }

    // второй вариант с использованием ананимного класса
```

```
//    @Override
//    public Iterator<String> iterator() {
//        return new Iterator<>() {
//            private int currentPos;
//
//            @Override
//            public boolean hasNext() {
//                return currentPos < songs.length;
//            }
//
//            @Override
//            public String next() {
//                if (!hasNext()) {
//                    throw new NoSuchElementException("Песен в playlist бо
//                }
//                return songs[currentPos++];
//            }
//        };
//    }
}
```

code/HwSolution42/src/playlist/PlaylistApp.java

```
package playlist;

import java.util.Iterator;

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

//Создайте класс Playlist, который реализует Iterable и содержит список пес
// который позволяет переключаться между песнями.

public class PlaylistApp {

    public static void main(String[] args) {
        Playlist playList = new Playlist(3);
        playList.addSong("Song 1");
        playList.addSong("Song 2");
        playList.addSong("Song 3");

        for (String song : playList) {
```

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

        Iterator<String> iterator = playList.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        // в этом случае метод next() выбросит ошибку NoSuchElementExceptio
//        iterator = playList.iterator();
//        while (true) {
//            System.out.println(iterator.next());
//        }

        int[] ints = {1, 3, 5};
        for (int num : ints) {
            System.out.println(num);
        }
    }
}
```

code/HwSolution42/src/playlist/PlayListIterator.java

```
package playlist;

import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 * @author Andrej Reutow
 * created on 08.11.2023
 */
public class PlayListIterator implements Iterator<String> {

    private int currentPos;

    private final String[] SONGS;

    public PlayListIterator(final String[] SONGS) {
        this.SONGS = SONGS;
    }

    @Override
    public boolean hasNext() {
```

```
            return currentPos < SONGS.length;
        }

        @Override
        public String next() {
            // boolean b = true;
            // if (b) - условие выполнено
            // if (!b) - условие не выполнено
            // b = false;
            // if (!b) - условие выполнено
            if (!hasNext()) {
                throw new NoSuchElementException("Песен в playlist больше нет")
            }
            return SONGS[currentPos++];
        }
    }
```

code/HwSolution42/src/olivier/Main.java

```
package olivier;

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


    public static void main(String[] args) {
        String[] ingredients = {"картошка", "огурцы", "яйца", "колбаса", "г
        Olivier olivier = new Olivier(ingredients);

        for (String string : olivier) {
            System.out.println("Нарезаем: " + string);
        }

        System.out.println("Салат Оливье готов!");
    }
}
```

code/HwSolution42/src/olivier/Olivier.java

```java
package olivier;

import java.util.Iterator;
import java.util.NoSuchElementException;

public class Olivier implements Iterable<String> {
    private final String[] ingredients;

    public Olivier(String[] ingredients) {
        this.ingredients = ingredients;
    }


    @Override
    public Iterator<String> iterator() {
        return new Iterator<>() {

            private int position = 0;


            public boolean hasNext() {
                return position < ingredients.length;
            }

            public String next() {
                if (!hasNext()) {
                    throw new NoSuchElementException("Все ингредиенты нарез
                }
                String ingredient = ingredients[position];
                position++;
                return ingredient;
            }
        };
    }
}
```

code/HwSolution42/src/water_cycle_stages/Stage.java

```java
package water_cycle_stages;

/**
```

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

public enum Stage {
    EVAPORATION,
    CONDENSATION,
    PRECIPITATION

}
```

code/HwSolution42/src/water_cycle_stages/WaterCycleStages.java

```
package water_cycle_stages;

import java.util.Iterator;
import java.util.NoSuchElementException;

public class WaterCycleStages implements Iterable<Stage> {

    private final Stage[] STAGES;

    public WaterCycleStages() {
        STAGES = new Stage[]{Stage.EVAPORATION, Stage.CONDENSATION, Stage.P
        //или обратиться к enum Stage.values()
        //STAGES = Stage.values();
    }

    @Override
    public Iterator<Stage> iterator() {
        return new WaterCycleIterator();
    }

    private class WaterCycleIterator implements Iterator<Stage> {
        private int currentIndex = 0;

        @Override
        public boolean hasNext() {
            return currentIndex < STAGES.length;
        }

        @Override
        public Stage next() {
            if (!hasNext()) {
```

```
                throw new NoSuchElementException();
            }
            return STAGES[currentIndex++];
        }
    }
}
```

code/HwSolution42/src/water_cycle_stages/WaterCycleStagesApp.java

```
package water_cycle_stages;

/**
 * @author Andrej Reutow
 * created on 07.11.2023
 */
public class WaterCycleStagesApp {

    public static void main(String[] args) {
        WaterCycleStages stages = new WaterCycleStages();

        for (Stage stage : stages){
            System.out.println(stage);
        }
    }
}
```

code/Game21_AIT/src/Card.java

```
import java.util.Objects;

/**
 * @author Andrej Reutow
 * created on 08.11.2023
 */
public class 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;
    }
}
```

code/Game21_AIT/src/Deck.java

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

```java
/**
 * @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/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
     */
    private int score; // 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;

        this.score += card.getVALUE();
    }

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

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