Meno a priezvisko: Delikatnyi Artem

Názov projektu: Casino Simulation

Popis projektu:

Program simuluje rôzne populárne kasínové hry: 21, ruleta, sloty.

Každá hra má svoju vlastnú jedinečnú logiku pre jej implementáciu.

V každej hre, ktorú chce užívateľ hrať, musí vykonať stávku. Používateľ si môže nezávisle vybrať, akú stávku na to, čo chce urobiť (v rámci pravidiel: napríklad minamalná stávka nesmie byť menšia ako 5). Na začiatku dostane každý nový používateľ na herný účet 50000. Ak používateľ vyčerpal herné peniaze na hernom účte, znamená to, že prehral a viac z tohto účtu nemôže komunikovať s hrami. Týmto spôsobom sa realizuje voľba hráča.

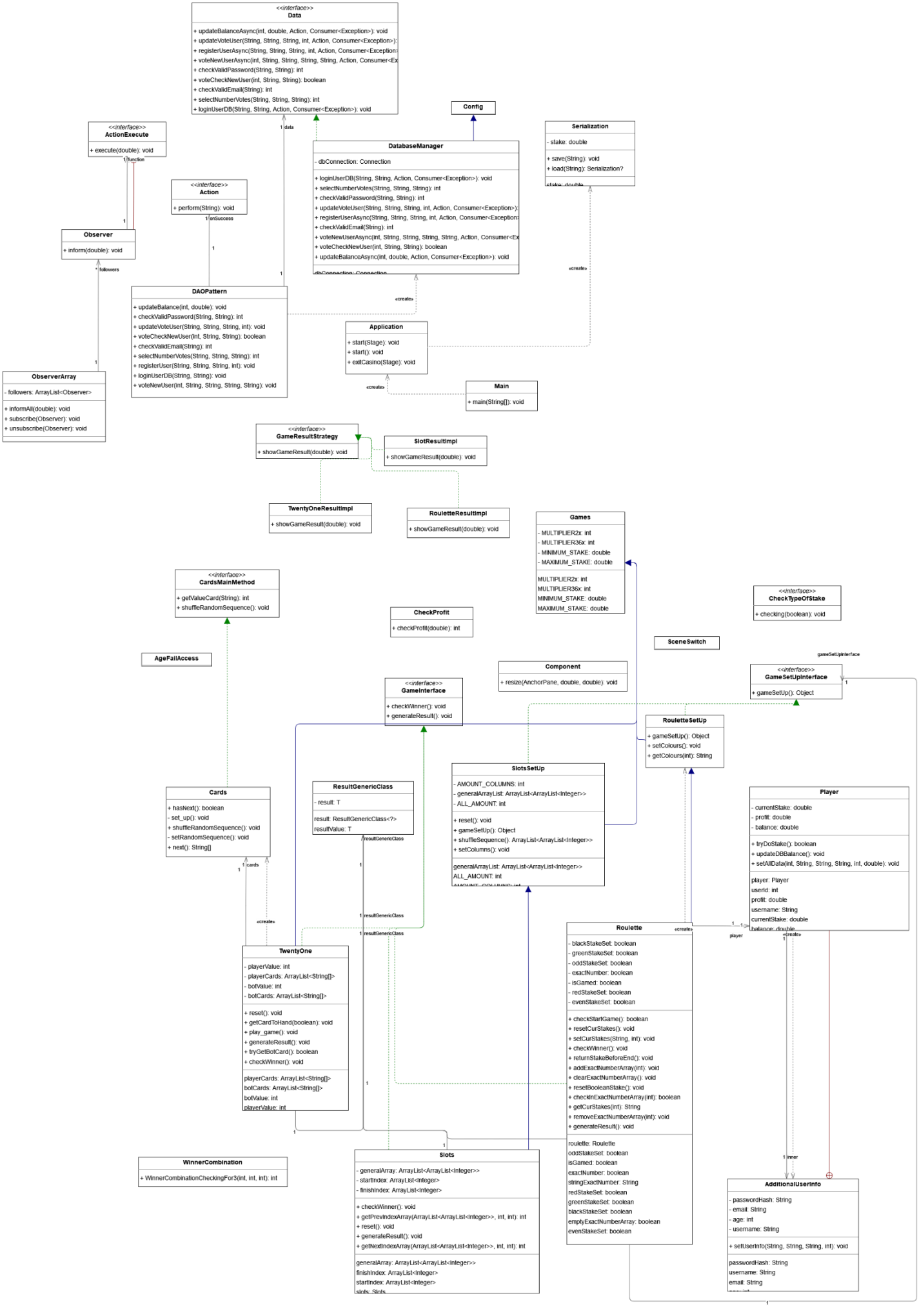
V programe sú dva typy hlasovania: hlasovanie za najlepšiu hru (ktorá je už prítomná) a hlasovanie za hru, ktorú by hráči chceli vidieť (hlasovanie tak dáva vývojárom predstavu, ako zlepšiť svoju aplikáciu a čo chcú používatelia). Po hlasovaní si používatelia môžu pozrieť štatistiky o tom, ako hlasovali ostatní ľudia, a počet hlasov pre jednotlivé kategórie. Ak používateľ zmení názor, môže zmeniť svoje hlasovanie a hlasovať znova.

Informácie o JavaDoc

Javadoc sa bude nachádzať v samostatnom priečinku s názvom "**JavaDoc**"

Schéma

Kompletný diagram sa nachádza vedľa súboru s dokumentáciou ako samostatný súbor s názvom: **"finnal_java_project_diagram.png"**. Ukázalo sa, že konečný diagram je dosť veľký, takže som ho nemohol umiestniť do tohto súboru. V tomto súbore uvediem len časť svojho diagramu, konkrétne časť s modelmi:


<<interface>>
Data
+ updateBalanceAsync(int, double, Action, Consumer<Exception>): void
+ updateVoteUser(String, String, String, int, Action, Consumer<Exception>):
+ registerUserAsync(String, String, String, int, Action, Consumer<Exception>
+ voteNewUserAsync(int, String, String, String, String, Action, Consumer<Ex
+ checkValidPassword(String, String): int
+ voteCheckNewUser(int, String, String): boolean
+ checkValidEmail(String): int
+ selectNumberVotes(String, String, String): int
+ loginUserDB(String, String, Action, Consumer<Exception>): void
Config
<<interface>>
ActionExecute
+ execute(double): void
1 function
1 data
DatabaseManager
- dbConnection: Connection
+ loginUserDB(String, String, Action, Consumer<Exception>): void
+ selectNumberVotes(String, String, String): int
+ checkValidPassword(String, String): int
+ updateVoteUser(String, String, String, int, Action, Consumer<Exception>):
+ registerUserAsync(String, String, String, int, Action, Consumer<Exception>
+ checkValidEmail(String): int
+ voteNewUserAsync(int, String, String, String, String, Action, Consumer<Ex
+ voteCheckNewUser(int, String, String): boolean
+ updateBalanceAsync(int, double, Action, Consumer<Exception>): void
dbConnection: Connection
Serialization
- stake: double
+ save(String): void
+ load(String): Serialization?
stake: double
<<interface>>
Action
+ perform(String): void
1 onSuccess
Observer
+ inform(double): void
* followers
«create»
«create»
DAOPattern
+ updateBalance(int, double): void
+ checkValidPassword(String, String): int
+ updateVoteUser(String, String, String, int): void
+ voteCheckNewUser(int, String, String): boolean
+ checkValidEmail(String): int
+ selectNumberVotes(String, String, String): int
+ registerUser(String, String, String, int): void
+ loginUserDB(String, String): void
+ voteNewUser(int, String, String, String, String): void
Application
+ start(Stage): void
+ start(): void
+ exitCasino(Stage): void
«create»
Main
+ main(String[]): void
ObserverArray
- followers: ArrayList<Observer>
+ informAll(double): void
+ subscribe(Observer): void
+ unsubscribe(Observer): void
<<interface>>
GameResultStrategy
+ showGameResult(double): void
SlotResultImpl
+ showGameResult(double): void
TwentyOneResultImpl
+ showGameResult(double): void
RouletteResultImpl
+ showGameResult(double): void
Games
- MULTIPLIER2x: int
- MULTIPLIER36x: int
- MINIMUM_STAKE: double
- MAXIMUM_STAKE: double
MULTIPLIER2x: int
MULTIPLIER36x: int
MINIMUM_STAKE: double
MAXIMUM_STAKE: double
<<interface>>
CardsMainMethod
+ getValueCard(String): int
+ shuffleRandomSequence(): void
CheckProfit
+ checkProfit(double): int
<<interface>>
CheckTypeOfStake
+ checking(boolean): void
SceneSwitch
AgeFailAccess
Component
+ resize(AnchorPane, double, double): void
gameSetUpInterface
<<interface>>
GameSetUpInterface
+ gameSetUp(): Object
<<interface>>
GameInterface
+ checkWinner(): void
+ generateResult(): void
RouletteSetUp
+ gameSetUp(): Object
+ setColours(): void
+ getColours(int): String
Cards
+ hasNext(): boolean
- set_up(): void
+ shuffleRandomSequence(): void
- setRandomSequence(): void
+ next(): String[]
ResultGenericClass
- result: T
result: ResultGenericClass<?>
resultValue: T
SlotsSetUp
- AMOUNT_COLUMNS: int
- generalArrayList: ArrayList<ArrayList<Integer>>
- ALL_AMOUNT: int
+ reset(): void
+ gameSetUp(): Object
+ shuffleSequence(): ArrayList<ArrayList<Integer>>
+ setColumns(): void
generalArrayList: ArrayList<ArrayList<Integer>>
ALL_AMOUNT: int
Player
- currentStake: double
- profit: double
- balance: double
+ tryDoStake(): boolean
+ updateDBBalance(): void
+ setAllData(int, String, String, String, int, double): void
player: Player
userId: int
profit: double
username: String
currentStake: double
balance: double
TwentyOne
- playerValue: int
- playerCards: ArrayList<String[]>
- botValue: int
- botCards: ArrayList<String[]>
+ reset(): void
+ getCardToHand(boolean): void
+ play_game(): void
+ generateResult(): void
+ tryGetBotCard(): boolean
+ checkWinner(): void
playerCards: ArrayList<String[]>
botCards: ArrayList<String[]>
botValue: int
playerValue: int
Roulette
- blackStakeSet: boolean
- greenStakeSet: boolean
- oddStakeSet: boolean
- exactNumber: boolean
- isGamed: boolean
- redStakeSet: boolean
- evenStakeSet: boolean
+ checkStartGame(): boolean
+ resetCurStakes(): void
+ setCurStakes(String, int): void
+ checkWinner(): void
+ returnStakeBeforeEnd(): void
+ addExactNumberArray(int): void
+ clearExactNumberArray(): void
+ resetBooleanStake(): void
+ checkInExactNumberArray(int): boolean
+ getCurStakes(int): String
+ removeExactNumberArray(int): void
+ generateResult(): void
roulette: Roulette
oddStakeSet: boolean
isGamed: boolean
exactNumber: boolean
stringExactNumber: String
redStakeSet: boolean
greenStakeSet: boolean
blackStakeSet: boolean
emptyExactNumberArray: boolean
evenStakeSet: boolean
WinnerCombination
+ WinnerCombinationCheckingFor3(int, int, int): int
Slots
- generalArray: ArrayList<ArrayList<Integer>>
- startIndex: ArrayList<Integer>
- finishIndex: ArrayList<Integer>
+ checkWinner(): void
+ getPrevIndexArray(ArrayList<ArrayList<Integer>>, int, int): int
+ reset(): void
+ generateResult(): void
+ getNextIndexArray(ArrayList<ArrayList<Integer>>, int, int): int
generalArray: ArrayList<ArrayList<Integer>>
finishIndex: ArrayList<Integer>
startIndex: ArrayList<Integer>
slots: Slots
AdditionalUserInfo
- passwordHash: String
- email: String
- age: int
- username: String
+ setUserInfo(String, String, String, int): void
passwordHash: String
username: String
email: String
1 inner

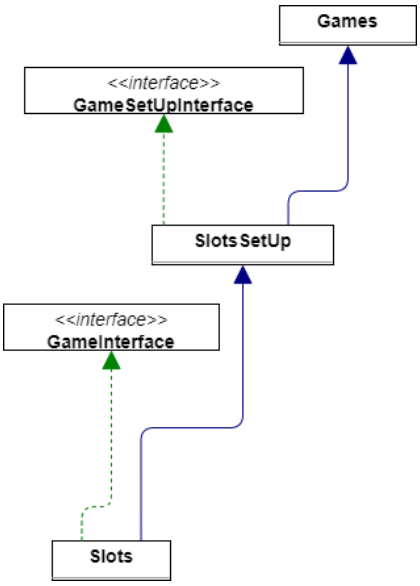
Hlavné kritériá

1. Dedenie + Rozhranie

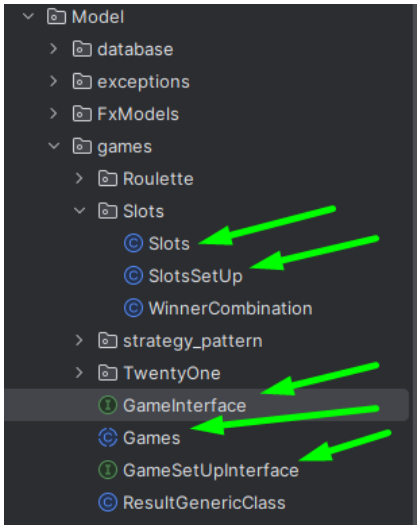
Dedičnosť sa používa na rôznych miestach. Dvojitá dedičnosť sa používa na implementáciu tried rôznych hier. Existuje abstraktná herná trieda “**Game**”, ktorá interne obsahuje základné metódy a atribúty vhodné pre všetky druhy hier. Táto trieda dedí triedu nastavenia hry, napríklad "**SlotsSetUp**". Potom trieda "**SlotsSetUp**" zdedí triedu "**Slots**".

Rozhranie sa často používa v programe. Používa sa na rôznych miestach, jedným z príkladov je rozhranie "GameSetUpInterface" - používa sa na zvýraznenie všetkých základných metód, ktoré musia byť prítomné v triedach nastavenia hry (napríklad: "SlotsSetUp"), a rozhranie "GameInterface" na zvýraznenie všetkých základných metód, ktoré musia byť prítomné vo všetkých hrách (napríklad v "Slots").

Príklad jedného z využití:




```
/**
 * This interface represents a generic game
 * interface providing methods for checking
 * the winner and generating results.
 */
6 usages  3 implementations  De1ik <artemdelikatnyi> +1
public interface GameInterface {
    Checks for the winner of the game.
    Throws: SQLException – If a database access error occurs.
            ClassNotFoundException – If the driver class cannot be found.
    3 usages  3 implementations  De1ik <artemdelikatnyi>
    void checkWinner() throws SQLException, ClassNotFoundException;

    Generates the result of the game.
    2 usages  3 implementations  De1ik <artemdelikatnyi>
    void generateResult();
}
```



2. Zapuzdrenie

Pravidlá údržby zapuzdrenia boli splnené a skrývanie údajov bolo správne implementované. Použitie privátnych atribútov a getter a setter na zobrazenie daných atribútov z prostredia mimo danej triedy.

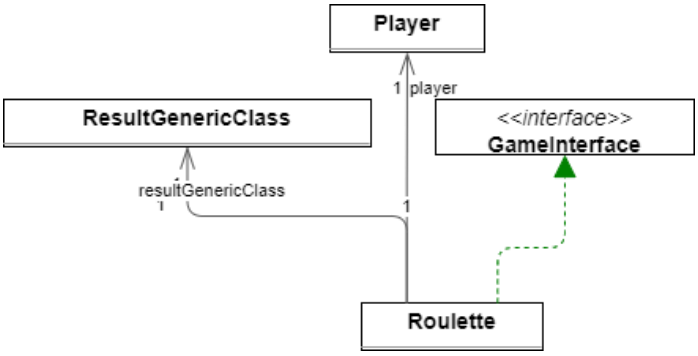
```
De1ik <artemdelikatnyi> +1
public class Roulette extends RouletteSetUp implements GameInterface {
    /** Flag indicating whether the game has started. */
    3 usages
    private boolean isGamed;

    /** Flag indicating whether an even stake has been set. */
    5 usages
    private boolean evenStakeSet;

    /** Flag indicating whether an odd stake has been set. */
    4 usages
    private boolean oddStakeSet;

    /** Flag indicating whether a red stake has been set. */
    4 usages
    private boolean redStakeSet;
```

```
//-------------------------------------Getters/Setters---------
/**
 * Checks if the game has started.
 * @return true if the game has started, otherwise false.
 */
1 usage  De1ik
public boolean isGamed() { return isGamed; }

/**
 * Sets the status of the game.
 * @param gamed The status of the game.
 */
2 usages  De1ik
public void setGamed(boolean gamed) { isGamed = gamed; }

/**
 * Checks if the player bet on even numbers.
 * @return true if the player bet on even numbers, otherwise false.
```

3. Agregácia

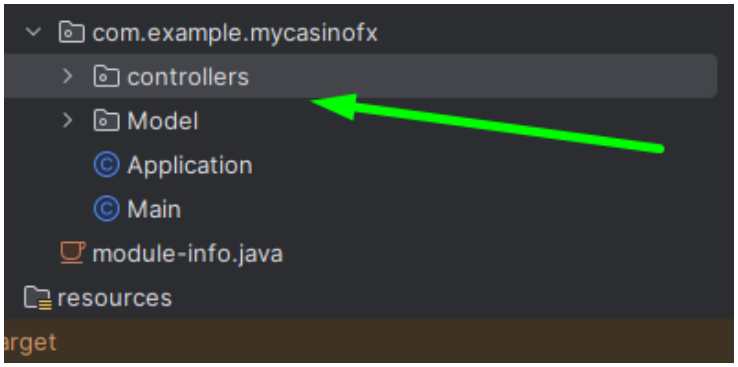
V tomto programe sa často používa aj agregácia. Jedným z príkladov je použitie iných objektov triedy vo vnútri triedy hry. Napríklad vnútri triedy "**Roulette**" sa používa objekt triedy "Player", "GameSetUpInterface" (rozhranie hry) a ResultGenericClass<Integer> (trieda výsledkov).

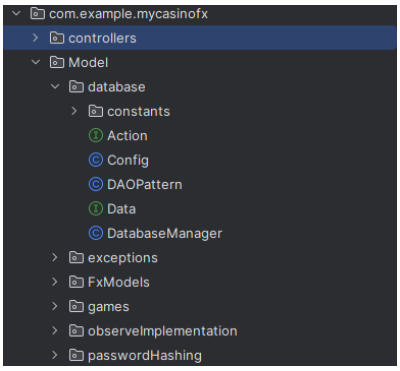
```
Interface for setting up the game.
2 usages
private GameSetUpInterface gameSetUpInterface;

The player participating in the game.
25 usages
private Player player;

The generic class for storing game results.
3 usages
private ResultGenericClass<Integer> resultGenericClass;
```



4. dôsledné oddelenie aplikačnej logiky od používateľského rozhrania; kód vhodne organizovaný do balíkov; prehľadná dokumentácia so všetkými položkami podľa opisu vyšši

Logika aplikácie je v package "**Model**", logika kontrolérov, ktoré sú zodpovedné za gui, je v package "**controllers**". Všetky logické prvky sú vo vlastných priečinkoch.





Ďalšie kritériá

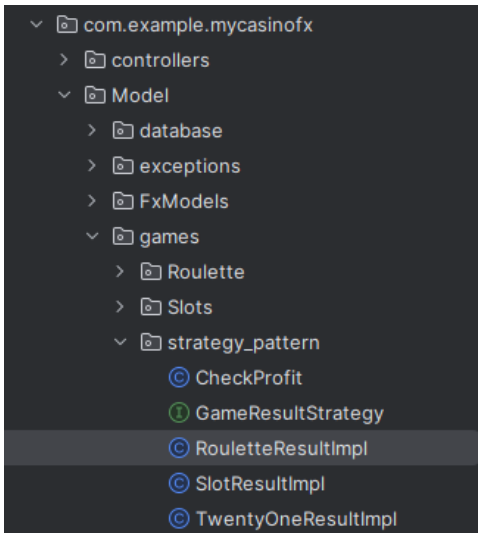
1. použitie návrhových vzorov okrem návrhového vzoru Singleton – každý implementovaný návrhový vzor sa počíta ako splnenie jedného ďalšieho kritériá, ale implementácia všetkých návrhových vzorov sa posudzuje maximálne na úrovni splnenia troch ďalších kritérií

Boli použité tri ďalšie vzory (okrem Singleton): Strategy, Observer a DAO

- **Strategy:**

Slúži na zobrazenie “logov” - vždy, keď používateľ hrá hru, do konzoly sa vypíše výsledok a názov hry.

Volá sa v rámci tried “**game controllers**”.



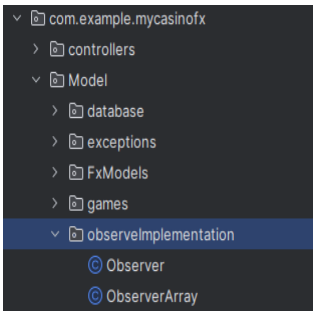

```java
/**
 * Implementation of the GameResultStrategy interface for displaying roulette game results.
 */
2 usages  De1ik <artemdelikatnyi>
public class RouletteResultImpl implements GameResultStrategy {

    /**
     * Shows the game result for Roulette game based on the profit.
     * @param profit The profit from the game.
     */
    3 usages  De1ik <artemdelikatnyi>
    @Override
    public void showGameResult(double profit) {
        int res = CheckProfit.checkProfit(profit);
        if (res == 1){
            System.out.println("User won in Roulette Game " + profit);
        }
        else if (res == 0){
            System.out.println("User save his money in Roulette");
        }
        else {
            System.out.println("User lose in Roulette Game " + profit);
        }
    }
}
```

- **Observer**

Observer sa používa na realizáciu zmien stávky. Keď chce používateľ zmeniť stávku, zobrazí sa okno, v ktorom je možné to zmeniť. Ak bola stávka zmenená, vyskakovacie okno informuje okno aplikácie a číslo aktuálnej stávky sa automaticky aktualizuje. Volá sa v triede **"SetStakeCustomDialog".**

```java
public class Observer {

    /**
     * Functional interface for defining actions to execute.
     */
    2 usages  De1ik <artemdelikatnyi>
    @FunctionalInterface
    public static interface ActionExecute{
        /**
         * Executes an action with a given number.
         * The number represents the user's rate.
         * @param number The number to execute the action with.
         */
        1 usage  De1ik <artemdelikatnyi>
        void execute (double number);
    }

    /**
     * The action to be executed by the observer.
     */
    2 usages
    private ActionExecute function;

    /**
     * Constructs an Observer with the specified action.
     * @param function The action to be executed.
     */
    1 usage  De1ik <artemdelikatnyi>
    public Observer(ActionExecute function) { this.function = function; }

    /**
     * Informs the observer about a change and executes the associated action.
     * @param newAmount The new amount to inform the observer about.
     */
    1 usage  De1ik <artemdelikatnyi>
    public void inform(double newAmount){
        function.execute(newAmount);
    }
}
```

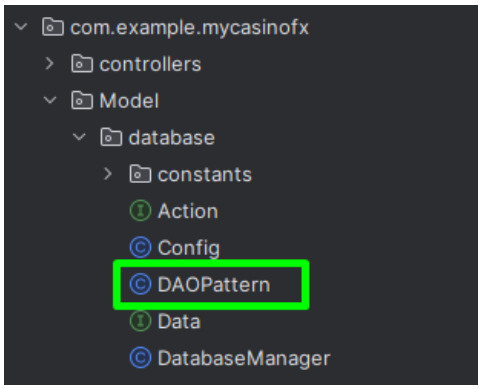
```java
package com.example.mycasinofx.controllers.custom_dialog_stake;

import ...

/**
 * Utility class for setting custom stakes in dialog window.
 */
De1ik <artemdelikatnyi>
public class SetStakeCustomDialog {

    /**
     * Executes the stake setting dialog.
     * @param borderPane The border pane containing the main content.
     * @param dialog_window The anchor pane representing the dialog window.
     * @param warningsLabel The label for displaying warning messages.
     * @param amountStake The label for displaying the stake amount.
     * @throws IOException If an I/O error occurs.
     */
    4 usages  De1ik <artemdelikatnyi>
    @FXML
    public static void doStake(BorderPane borderPane, AnchorPane dialog_window, Label

        dialog_window.setMouseTransparent(false);

        FXMLLoader loader = new FXMLLoader(Application.class.getResource( name: "view
        AnchorPane nextAnchorPane = loader.load();

        DialogController dialogController = loader.getController();
        dialogController.updateBalanceLabel();
        dialogController.subscribe(
                new Observer((double number) -> {
                    borderPane.setDisable(false);
                    dialog_window.setMouseTransparent(true);
                    if (number > -1){
                        warningsLabel.setText("The Stake Was Changed");
                        amountStake.setText("Stake: " + number);
                    }
                }));
```



- **Dao** **(Dôkaz, že sa považuje za vzor)**

Používa sa na abstraktnú interakciu s databázou. Volá sa na rôznych miestach, kde je potrebné pristupovať k databáze. (Napríklad v hrách na zmenu zostatku)

```java
/**
 * The DAOPattern class provides methods for interacting with the databa
 * It contains static methods for updating balances, voting, selecting d
 * and registering new users asynchronously.
 */
31 usages  De1ik <artemdelikatnyi> +1 *
public class DAOPattern {

    /**
     * The data manager responsible for database operations.
     */
    9 usages
    static Data data = new DatabaseManager();

    /**
     * Updates the balance of a user asynchronously.
     * @param id The ID of the user.
     * @param newBalance The new balance to be set.
     */
    1 usage  De1ik <artemdelikatnyi>
    public static void updateBalance(int id, double newBalance ){
        data.updateBalanceAsync(id, newBalance, onSuccess, onError);
    }

    /**
     * Updates the vote of a user asynchronously.
     * @param setCategory The category to be set.
```

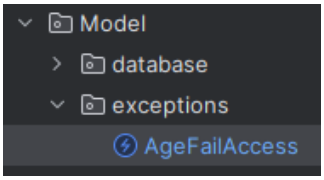


2. ošetrenie mimoriadnych stavov prostredníctvom vlastných výnimiek – stačí jedna vlastná výnimka, ale musí byť skutočne vyhadzovaná a ošetrovaná

Používa vlastnú výnimku na zobrazenie chyby (a tip pre používateľa) keď sa používateľ mladší ako 18 rokov pokúsi zaregistrovať v hre.

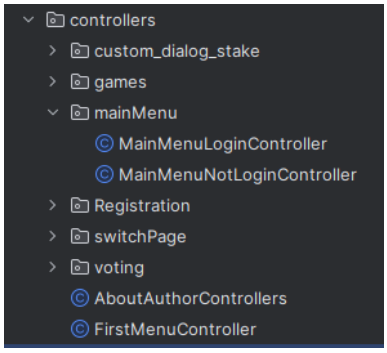
```java
/**
 * The AgeFailAccess class represents an exception
 * that is thrown when access is denied due to age restrictions.
 */
6 usages  De1ik +1 *
public class AgeFailAccess extends Exception {

    /**
     * Constructs a new AgeFailAccess exception with
     * the specified error message.
     * @param errorMessage The error message explaining
     *                     the reason for the age access failure.
     */
    1 usage  De1ik
    public AgeFailAccess(String errorMessage) { super(errorMessage); }
}
```

```java
import com.example.mycasinofx.Model.exceptions.AgeFailAccess;

/**
 * Utility class for checking the validity of data.
 */
De1ik <artemdelikatnyi> +1 *
public class CheckValidData {

    /**
     * Validates the age against a minimum threshold.
     * @param age The age to validate.
     * @throws AgeFailAccess If the age is below the minimum threshold.
     */
    1 usage  new *
    static void validate (int age) throws AgeFailAccess {
        if(age < 18){
            throw new AgeFailAccess( errorMessage: "Minimum Age Is 18 Years");
        }
    }
```



3. poskytnutie grafického používateľského rozhrania oddelene od aplikačnej logiky a s aspoň časťou spracovateľov udalostí (handlers) vytvorenou manuálne – počíta sa ako splnenie dvoch ďalších kritérií

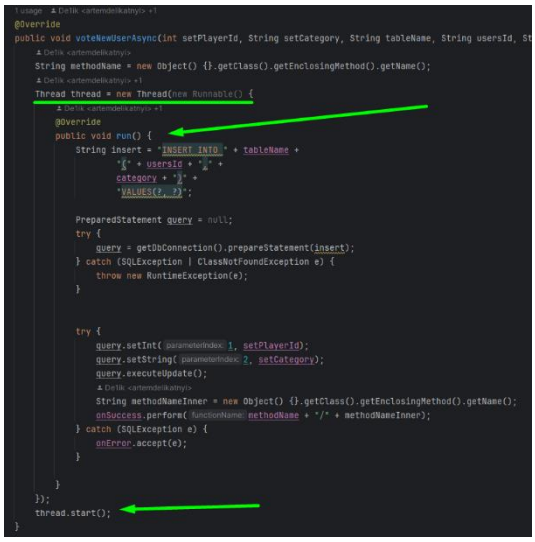
Na implementáciu grafického rozhrania sa používa JavaFx. Kontroléry sú oddelené od hlavnej logiky programu a majú v sebe hlavne metódy na interakciu s gui.



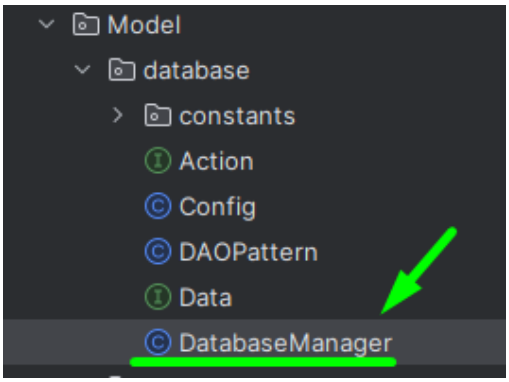

```java
/**
 * Controller class for the main menu and for users who have registered.
 */
De1ik <artemdelikatnyi> +1
public class MainMenuLoginController {
    /**
     * The anchor pane for the main menu.
     */
    @FXML
    private AnchorPane mainMenuLogin;

    /**
     * The anchor pane for the dialog window.
     */
    @FXML
    private AnchorPane dialogWindow;
```

4. explicitné použitie viacniťovosti (multithreading) – spustenie vlastnej nite priamo alebo prostredníctvom API vyššej úrovne (trieda **Task** a pod.)

Viacvláknovosť sa používa pri niektorých metódach prístupu k databáze, pri ktorých nie je dôležitá okamžitá implementácia metódy.



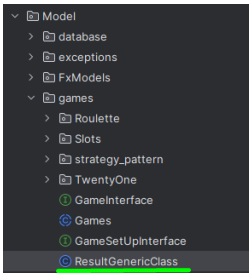
```
@Override
public void voteNewUserAsync(int setPlayerId, String setCategory, String tableName, String usersId, St
    String methodName = new Object() {}.getClass().getEnclosingMethod().getName();
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            String insert = "INSERT INTO " + tableName +
                    "(" + usersId + "," +
                    category + ")" +
                    "VALUES(?, ?)";

            PreparedStatement query = null;
            try {
                query = getDbConnection().prepareStatement(insert);
            } catch (SQLException | ClassNotFoundException e) {
                throw new RuntimeException(e);
            }


            try {
                query.setInt( parameterIndex: 1, setPlayerId);
                query.setString( parameterIndex: 2, setCategory);
                query.executeUpdate();
                String methodNameInner = new Object() {}.getClass().getEnclosingMethod().getName();
                onSuccess.perform( functionName: methodName + "/" + methodNameInner);
            } catch (SQLException e) {
                onError.accept(e);
            }

        }
    });
    thread.start();
}
```



5. použitie generickosti vo vlastných triedach – implementácia a použitie vlastnej generickej triedy (ako v príklade spájaného zoznamu poskytnutého k prednáške 5)

Táto trieda slúži ako spoločný kontajner na ukladanie výsledkov rôznych typov hier. Poskytuje abstrakčnú vrstvu na prístup k výsledkom hier a ich nastavenie. Keď hra vygeneruje výsledok, uloží sa do tejto triedy, a keď je potom potrebné porovnať výsledky používateľa s víťaznými výsledkami, výsledok hry sa prevezme z tejto triedy. Volanie metód tejto triedy sa realizuje v ovládačoch rôznych hier

```
This class serves as a generic container for storing results of different types of games. It provides an
abstraction layer for accessing and setting game results. Once the game has generated a result, it is
stored in this class, then when the user's results need to be compared with the winning results, the
game's result is taken from this class.
Type parameters: <T> – The type of result to be stored.
25 usages  De1ik <artemdelikatnyi>
public class ResultGenericClass<T>{

    The result stored in this class.
    2 usages
    private T result;
    Singleton instance of ResultGenericClass.
    3 usages
    private static ResultGenericClass<?> resultGenericClass = null;

    Private constructor to enforce singleton pattern.
    1 usage  De1ik <artemdelikatnyi>
    private ResultGenericClass() {
    }

    Retrieves the singleton instance of ResultGenericClass.
    Returns: The singleton instance.
    De1ik <artemdelikatnyi>
    public static synchronized ResultGenericClass<?> getResult(){
        if (resultGenericClass == null){
            resultGenericClass = new ResultGenericClass<>();
        }
        return resultGenericClass;
    }

    Retrieves the stored result value.
    Returns: The stored result value.
    5 usages  De1ik <artemdelikatnyi>
    public T getResultValue() { return result; }
```

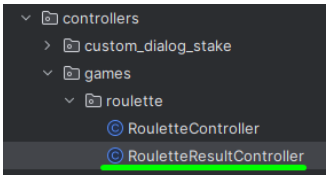



```
package com.example.mycasinofx.controllers.games.slots;

import ...

Controller class for managing the UI of the SlotsController game page.
De1ik <artemdelikatnyi> +1 *
public class SlotsController implements Initializable {

    Represents the player participating in the slots game.
    16 usages
    private Player player;

    The instance for managing generic game results.
    2 usages
    private ResultGenericClass resultGenericClass;
```

6. explicitné použitie RTTI – napr. na zistenie typu objektu alebo vytvorenie objektu príslušného typu (ako v hre s obrami a rytiermi pri zisťovaní počtu bytostí)

Používa sa v triede "**RouletteResultController**" na rôznych miestach na porovnávanie typov. Napríklad pri metóde "**changeColor**", ktorá vytvára animačný efekt - **instanceof**, sa používa na porovnanie typu objektu, ak je typ objektu Label, potom sa majú vykonať niektoré akcie, ak nie, potom iné.

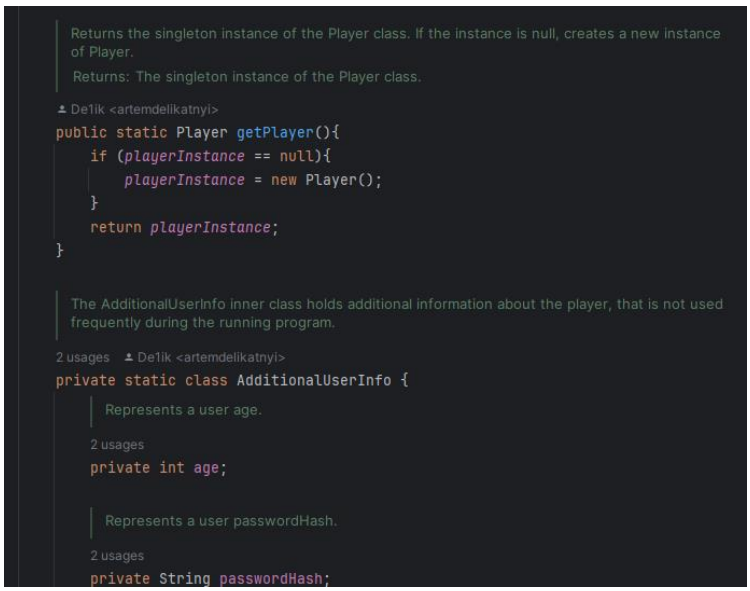
```
2 usages  De1ik
@FXML
public void setCurrentStakeText() {
    int index = 0;
    //set labels by text
    for (int i = 0; i < 5; i++) {
        if (roulette.getCurStakes(i) != null) {
            if (textFlow.getChildren().get(index) instanceof Label) {
                Label label = (Label) textFlow.getChildren().get(index);
                label.setText(roulette.getCurStakes(i));
                index += 1;
            }
        }
    }

    //set the label disabled
    for (int i = index; i < 5; i++) {
        if (textFlow.getChildren().get(i) instanceof Label) {
            Label label = (Label) textFlow.getChildren().get(i);
            label.setVisible(false);
        }
    }

}
```



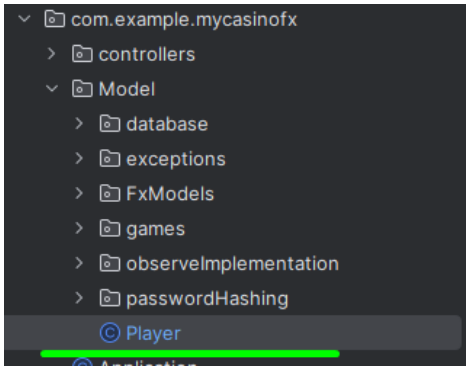
```
Changes the color of roulette grid elements to simulate animation.
4 usages  De1ik +1
private void changeColor(GridPane gridPane, int index, int delay, int curGeneration) {
    if (index < 55) {
        int number = ((curGeneration + 19) + index) % 37;
        if (number == 36) {
            zeroPolygon.setFill(Color.YELLOW);
            if (index + 1 != 55) {
                PauseTransition pause = new PauseTransition(Duration.millis(delay));
                pause.setOnFinished(event -> {
                    zeroPolygon.setFill(Color.GREEN);
                    int newDelay = (int) (delay * 1.1);
                    changeColor(gridPane,  index: index + 1, newDelay, curGeneration);
                });
                pause.play();
            } else {
                setMessageLabel();
            }
        //RTTI -> instanceof
        } else if (gridPane.getChildren().get(number) instanceof Label) {

            Label label = (Label) gridPane.getChildren().get(number);
```

7. použitie vhniezdených tried a rozhraní – počíta sa iba použitie v aplikačnej logike, nie v GUI, pričom rozhrania musia byť vlastné (jedna možnosť je v príklade vnútorných tried k prednáške 4)

Vnorená trieda sa používa "**AdditionalUserInfo**" v triede "**Player**" na ukladanie informácií o aktuálnom používateľovi, ktoré sa používajú len zriedka (napr. e-mail).



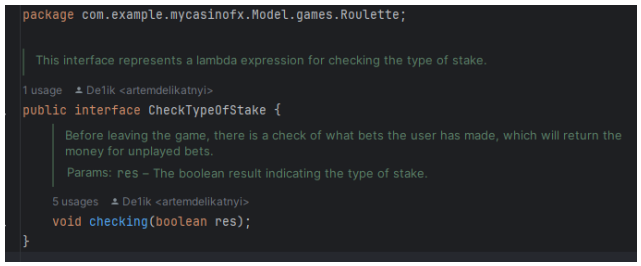
```
Returns the singleton instance of the Player class. If the instance is null, creates a new instance of Player.
Returns: The singleton instance of the Player class.
De1ik <artemdelikatnyi>
public static Player getPlayer(){
    if (playerInstance == null){
        playerInstance = new Player();
    }
    return playerInstance;
}

The AdditionalUserInfo inner class holds additional information about the player, that is not used frequently during the running program.
2 usages  De1ik <artemdelikatnyi>
private static class AdditionalUserInfo {
    Represents a user age.
    2 usages
    private int age;

    Represents a user passwordHash.
    2 usages
    private String passwordHash;
```



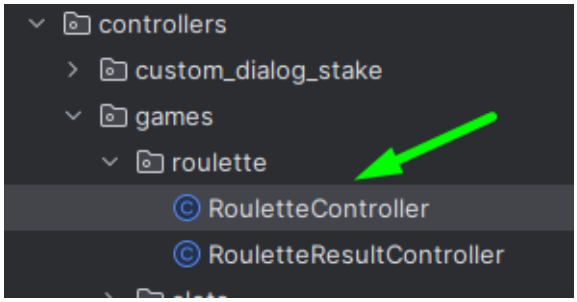
8. použitie lambda výrazov alebo referencií na metódy (method references) – počíta sa iba použitie v aplikačnej logike, nie v GUI (jedna možnosť je v príklade referencií na metódy a lambda výrazov k prednáške 4)

Lambda funkcie sa používajú v triede "**RouletteController**" pre metódu "**returnStakeBeforeEnd**", ktorá implementuje vrátenie nepoužitých stávok späť používateľovi.

```
Returns the stake to the player before the game ends, if the game has not started. The stake is
returned based on the types of bets made by the player. If the player bet on even, odd, red,
black, or green, and the game has not started, the corresponding stake amount is returned to the
player's balance. Additionally, if the player bet on exact numbers and the game has not started,
the total stake amount for exact numbers is returned to the player's balance.

1 usage   De1ik <artemdelikatnyi> +1
public void returnStakeBeforeEnd() {
    if (!isGamed()) {

        CheckTypeOfStake lambda_checking = (boolean res) -> {
            if (res){
                player.setBalance(player.getBalance() + player.getCurrentStake());
            }
        };

        lambda_checking.checking(isEvenStakeSet());
        lambda_checking.checking(isOddStakeSet());
        lambda_checking.checking(isRedStakeSet());
        lambda_checking.checking(isBlackStakeSet());
        lambda_checking.checking(isGreenStakeSet());

        if (!isEmptyExactNumberArray()) {
            double amount = exactNumberArray.size() * player.getCurrentStake();
            player.setBalance(player.getBalance() + amount);
        }
    }
    setGamed(false);
}
```



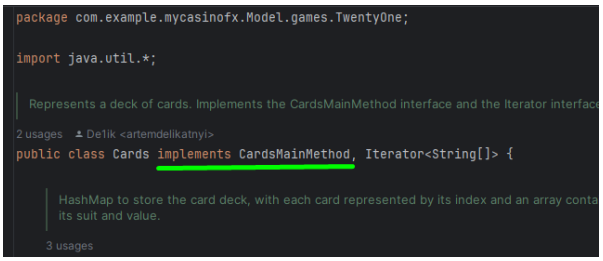
```
package com.example.mycasinofx.Model.games.Roulette;

This interface represents a lambda expression for checking the type of stake.

1 usage   De1ik <artemdelikatnyi>
public interface CheckTypeOfStake {
    Before leaving the game, there is a check of what bets the user has made, which will return the
    money for unplayed bets.
    Params: res – The boolean result indicating the type of stake.
    5 usages   De1ik <artemdelikatnyi>
    void checking(boolean res);
}
```



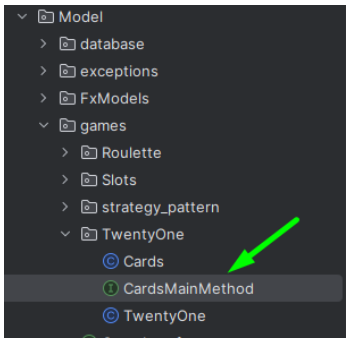
9. použitie implicitnej implementácie metód v rozhraniach (default method implementation)

Default method sa používa v rozhraní "**CardsMainMethod**" na vrátenie bodových hodnôt niektorých kariet. Toto rozhranie implementuje trieda "**Cards**"

```
Gets the numerical value of a card based on its type.
Params: type – The type of the card.
Returns: The numerical value of the card.
2 usages   De1ik <artemdelikatnyi>
default int getValueCard(String type){
    if (CARD_NUMBERS.contains(type)){
        return Integer.parseInt(type);
    }

    switch (type){
        case "jack":
            return 3;
        case "queen":
            return 4;
        case "king":
            return 5;
        case "ace":
            return 1;
    }

    return 0;
}
```



```
package com.example.mycasinofx.Model.games.TwentyOne;

import java.util.*;

Represents a deck of cards. Implements the CardsMainMethod interface and the Iterator interface
2 usages   De1ik <artemdelikatnyi>
public class Cards implements CardsMainMethod, Iterator<String[]> {

    HashMap to store the card deck, with each card represented by its index and an array conta
    its suit and value.
    3 usages
```



10. použitie serializácie

Serializácia sa používa na uloženie stavu hodnoty poslednej stávky. Keď používateľ zatvorí aplikáciu, číslo jeho poslednej stávky sa serializuje a uloží. Keď používateľ vstúpi do aplikácie, jeho stávka sa automaticky nastaví na predvolenú hodnotu 5, potom program skontroluje, či existujú údaje pre deserializáciu, ak áno, program aktualizuje číslo stávky používateľa.

```java
/**
 * Handles the exit action for the application.
 * Params: stage – The primary stage for the application.
 */
1 usage  ♟ De1ik <artemdelikatnyi> +1
public void exitCasino(Stage stage){

    ImageView imageView = new ImageView(new Image( url: "/casino-logo.png"));
    imageView.setFitWidth(70);
    imageView.setFitHeight(70);

    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);


    alert.getDialogPane().setGraphic(imageView);

    alert.setTitle("Exit Application");
    alert.setHeaderText("Do you want to exit Casino?");
    alert.setContentText("Are you sure?");

    String css = Objects.requireNonNull(this.getClass().getResource( name: "view/styles/styles.css")).toExternalForm();
    alert.getDialogPane().getStylesheets().add(css);

    alert.initModality(Modality.APPLICATION_MODAL);

    if (alert.showAndWait().get() == ButtonType.OK) {
        Player player = Player.getPlayer();
        Serialization userSettingsToSave = new Serialization(player.getCurrentStake());
        System.out.println("Last user stake was save in 'user_last_stake.ser'");
        userSettingsToSave.save( filename: "user_last_stake.ser");
        stage.close();
    }
}
```

```java
/**
 * Constructs a Player object. Initializes the current stake with a default value of 5, and loads the last
 * user stake from a file if available (Serialization). Creates an instance of the AdditionalUserInfo
 * inner class.
 */
1 usage  ♟ De1ik <artemdelikatnyi>
private Player(){
    setCurrentStake(5);
    Serialization loadedUserSettings = Serialization.load( filename: "user_last_stake.ser");
    if (loadedUserSettings != null) {
        System.out.println("Last user stake was download from 'user_last_stake.ser'");
        setCurrentStake(loadedUserSettings.getStake());
    }
    inner = new AdditionalUserInfo();
}
```