Poker Online

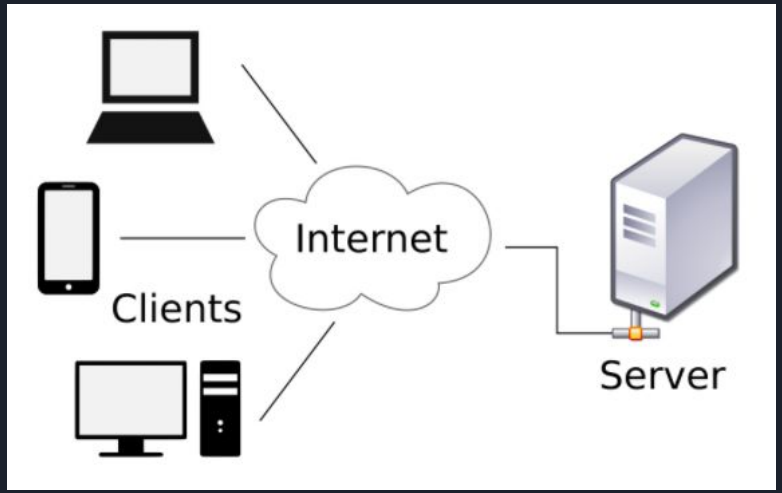
Wykonawcy:

Christian Białek - cbialek@student.agh.edu.pl

Mateusz Zdyrski - zdyrski@student.agh.edu.pl

Plan prezentacji

1. Demonstracja naszego projektu
2. Programowanie socketowe
3. JavaFX
4. Wielowątkowość - JavaFX



1. Demonstracja projektu

Nasz projekt korzysta z architektury klient-serwer, a tłem całego projektu jest w pełni zaimplementowana gra karciana Poker.

Serwer nasłuchując na porcie 5000 czeka na zestawienie nowego połączenia. Gdy klient łączy się z serwerem, startuje nowy wątek ClientHandler, który jest odpowiedzialny za obsługę tego klienta i komunikację z nim. Z reguły do serwera podłącza się większa liczba klientów, którym serwer udostępnia swoje zasoby.

W naszym przypadku po podłączeniu i zalogowaniu się 3 klientów, zaczyna się gra. Nasz klient zawiera wszystkie interfejsy użytkownika, a serwer całą implementację pokera. Klient wysyłając wiadomość odnośnie swojego zachowania przy “stole”, czeka na odpowiedź od serwera, gdzie ta wiadomość jest przetwarzana, sprawdzana ze wszystkimi warunkami gry
i następnie z powrotem odesłana do klienta. Klient po otrzymaniu wiadomości zwrotnej aktualizuje GUI wyświetlając wszystkie zmiany

i tak w kółko… :)

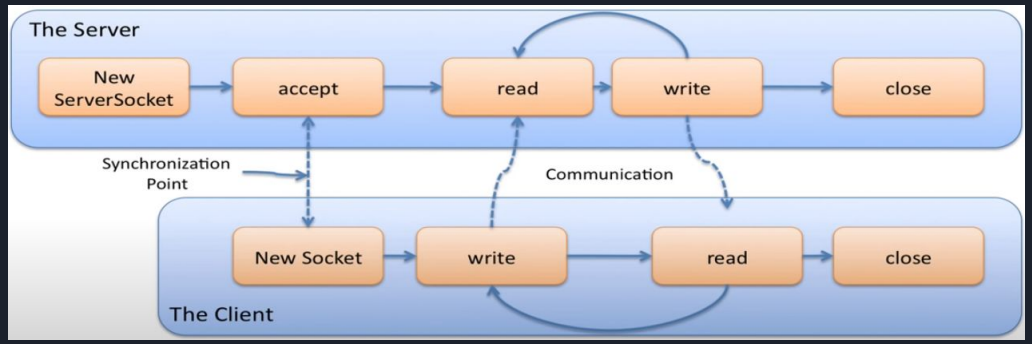
2. Programowanie socketowe

Zacznijmy od tego czym w ogóle jest ten socket?

Socket jest to pewien mechanizm umożliwiający otwarcie kanału komunikacji pomiędzy hostami. Każde gniazdo posiada adres IP oraz numer portu i może przesyłać m.in trzy rodzaje pakietów:

- datagramy (pakiety UDP)
- strumienie (pakiety TCP)
- raw sockets (IP sockets)

W przypadku modelu klient-serwer, gniazdo na serwerze czeka na żądanie klienta. W tym celu serwer musi przede wszystkim ustalić adres, który klienci mogą wykorzystać do znalezienia i połączenia z serwerem. Gdy połączenie zostanie pomyślnie nawiązane, serwer będzie czekał, aż klienci zażądają usługi. Wymiana danych klient-serwer będzie miała miejsce, jeśli klient połączy się z serwerem poprzez gniazdo. Serwer odpowie na prośbę klienta i wyśle odpowiedź.



Interfejs java.io.Serializable

Serializacja służy do konwersji stanu obiektu w strumień(sekwencję) bajtów, które mogą być zapisane w bazy danych lub przesłane przez internet i następnie wiernie odtworzone.

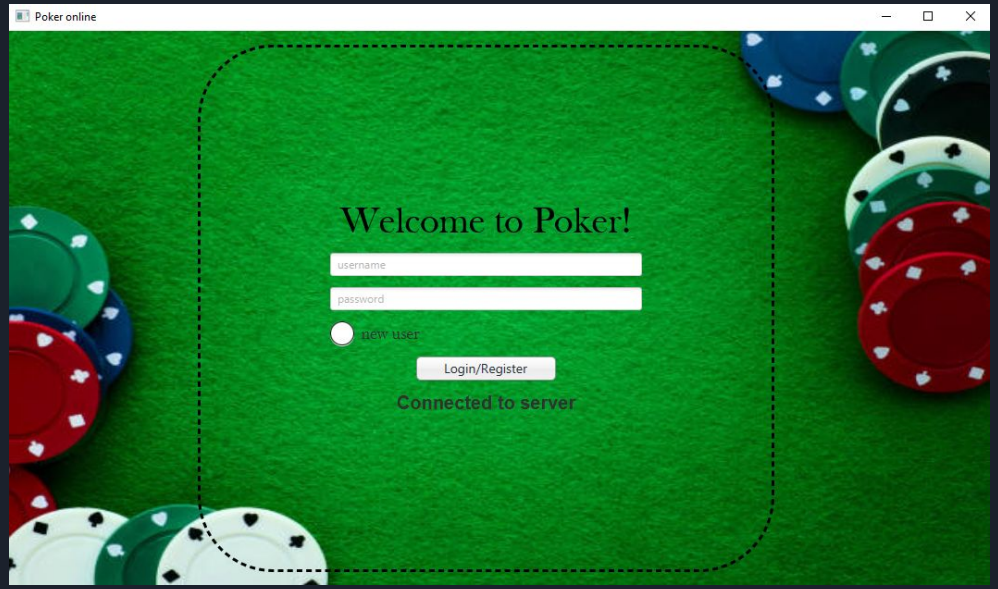
Odpowiednie klasy wyznaczone do serializacji muszą zaimplementować tzw. marker interface.

```
import java.io.*;
class Wezel implements Serializable {
  String nazwa;
  /* ... */
}
public class Serializacja {
  Wezel korzen = new Wezel();
  /* ... */
  void zapiszWezly() throws IOException {
    /* ... utworzenie strumienia wyjściowego ... */
    ObjectOutputStream out =
      new ObjectOutputStream( /*strumień wyj.*/ );
    out.writeObject(korzen);
  }
```

```
public class Message implements Serializable
```

Troszkę o “bezpieczeństwie”

Chcąc zaimplementować możliwość rejestrowania się użytkowników do naszej aplikacji, pojawił się problem z bezpiecznym przechowywaniem haseł na serwerze.



Algorytmy skrótu SHA

SHA-256 – ang. Secure Hash Algorithm 256-bit (256-bitowy bezpieczny algorytm haszujący). Kryptograficzna funkcja skrótu, czyli algorytm używany do ochrony kryptograficznej różnego rodzaju danych. Operacja matematyczna wykonywana na danych cyfrowych; porównująca obliczony hash ze znaną i oczekiwaną wartością poprzedniego hash-u.

MessageDigest class -> provides applications the functionality of a message digest algorithm, such as SHA-1 or SHA-256.

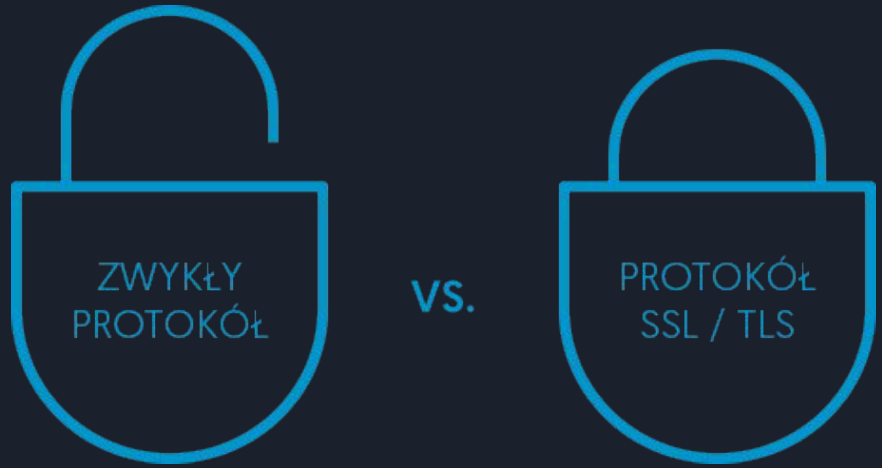
```
private String generateHash(String password, String algorithm) {
    String hashValue = "";

    try {
        MessageDigest messageDigest = MessageDigest.getInstance(algorithm);
        messageDigest.reset();
        byte[] hash = messageDigest.digest(password.getBytes());
        hashValue = bytesToStringHex(hash);
    } catch (NoSuchAlgorithmException e) {
        System.out.println("error:hash: " + e.getStackTrace() + " " + e.getCause());
    }
    return hashValue;
}
```

SSL/TLS

TLS, czyli Transport Layer Security to protokół zapewniający autentyczność, prywatność i integralność danych przesyłanych pomiędzy aplikacjami. Jest to najpopularniejszy protokół bezpieczeństwa w Internecie stosowany przez przeglądarki i przydatny w sytuacjach, gdy konieczna jest ochrona danych sesji, plików, VPN, zdalnego pulpitu czy VoIP.

TSL jest wyższą wersją SSL.



3. Czym jest JavaFX

Java FX jest domyślną biblioteką definiowania graficznego interfejsu użytkownika w języku Java począwszy od wersji 8. Zastąpiła ona starsze biblioteki takie jak Swing i AWT. Charakteryzuje się przede wszystkim dobrą separacją warstw, wsparciem dla architektury MVC (Model View Controller), możliwością stylizacji komponentów przy użyciu CSS .

Istnieją dwa podstawowe sposoby tworzenia GUI wykorzystującego JavaFX:

- z użyciem „czystej” Javy (jak w przypadku naszego projektu),

- z użyciem plików FXML (udostępnia do tego użyteczne narzędzie o nazwie Scene Builder).

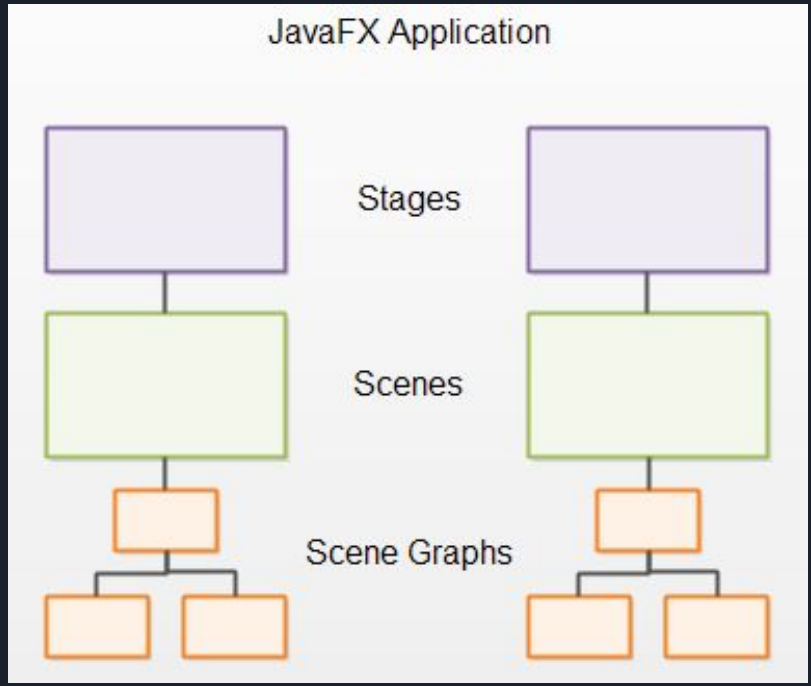
Aplikacje w JavaFX mają charakter hierarchiczny np. obiekt typu Button umieszczamy w obiekcie typu StackPane, ten z kolei w obiekcie typu Scene, który był umieszczony w obiekcie primaryStage. Wszystkie elementy, które umieszczamy w Scene są węzłami (ang. node).

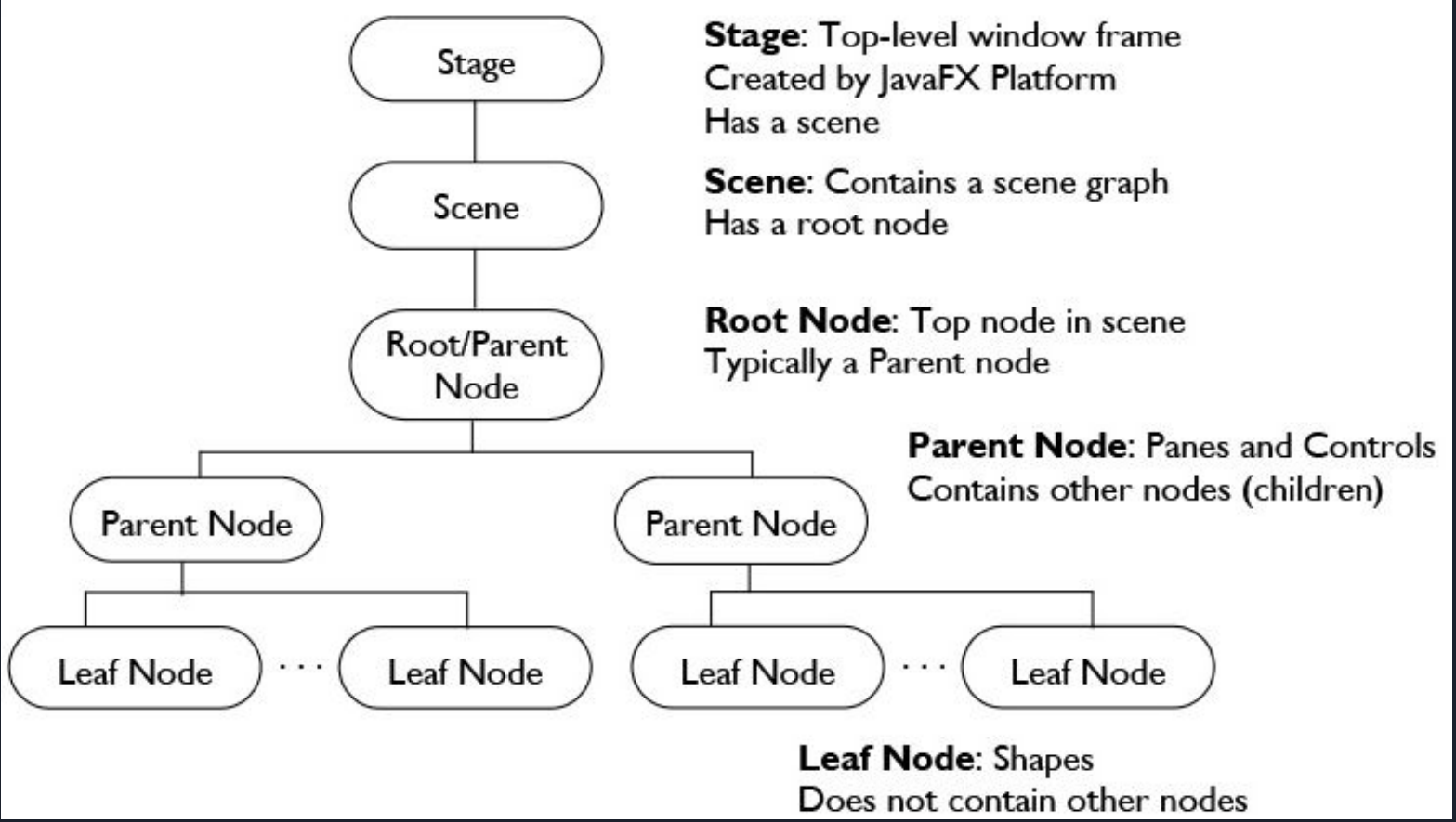
JavaFX - ciąg dalszy

Może to przypominać zależność liniową. Obiekt, który znajduje się wyżej w hierarchii jest „rodzicem” (ang. parent) a ten, który znajduje się niżej jest „dzieckiem” (ang. child). Rzeczywisty obraz zależności w aplikacji jest jednak bardziej złożony, relacje między obiektami przypominają raczej drzewo niż drabinę. Węzeł może być rodzicem dla wielu innych węzłów. Jeśli nie posiada węzłów potomnych, nazywany jest liściem (ang. leaf).

Podstawowym, „szczytowym” elementem aplikacji graficznej w JavaFX jest stage, która określa okno jest „kontenerem” dla wszystkich elementów znajdujących się w oknie aplikacji. Stage zawiera scene, która z kolei zawiera całe drzewo składające się z węzłów. To w obiekcie typu Scene definiujemy rozmiary okna i w niej umieszczamy obiekt root, który znajdował się stopień niżej w hierarchii.

Obiekt root jest przykładem panelu – obiektu służącego do rozmieszczania innych obiektów w aplikacji. W JavaFX wyróżniamy: BorderPane, HBox, VBox, GridPane, StackPane, FlowPane ...




Stage
Scene
Root/Parent Node
Parent Node
Parent Node
Leaf Node
Leaf Node
Leaf Node
Leaf Node
Stage: Top-level window frame
Created by JavaFX Platform
Has a scene
Scene: Contains a scene graph
Has a root node
Root Node: Top node in scene
Typically a Parent node
Parent Node: Panes and Controls
Contains other nodes (children)
Leaf Node: Shapes
Does not contain other nodes

4. Wielowątkowość w JavaFX - Platform.runLater()

JavaFX używa modelu jednowątkowego(single-threaded rendering design), oznacza to, że tylko jeden wątek ma niejako dostęp do wyświetlanej sceny. Takim głównym wątkiem aplikacji jest JavaFX application thread i tylko on posiada możliwość zmiany JavaFX Scene Graph.

Czasami potrzebujemy wykonać jakieś dłuższe działanie w JavaFX application, ale przy tym nie chcemy zostawiać nierosponsywnego GUI, z tego powodu niektórym taskom tworzymy własne wątki. Jednakże, w tym samym momencie chcielibyśmy aktualizować nasze GUI na bieżąco po zakończonym task'u lub nawet w jego trakcie. Jak już wiemy, osobne wątki tasków nie mogą bezpośrednio zaktualizować GUI (od tego jest tylko JavaFX application thread) - a więc jak rozwiązać ten problem?

Platform.runLater() - rozwiązanie naszego problemu

JavaFX posiada klasę Platform zawierającą metodę runLater(), która pozwala aktualizować składnik GUI przez tzw. non-GUI thread, umożliwia jak najszybsze wprowadzenie uaktualnienia przez GUI thread(główny wątek).

Platform.runLater() - rozwiązanie naszego problemu

```
public void displayHand(String[] handCards){
    Platform.runLater(() -> {
        for(int i = 0; i < handCards.length; i++){
            Node handChange = myHand.getChildren().get(i);
            if (handChange instanceof ImageView){
                ((ImageView) handChange).setImage(new Image("images/"+handCards[i]+".png"));
            }
        }
        GridPane.setHalignment(myHand, HPos.CENTER);
    });
}
```

```
public void actualizePoolMoney(int i){
    Platform.runLater(() -> {
        String money = String.valueOf(i);
        pool.setText("Pool: " + money + " $");
    });
}
```

```
public void resetCards(){
    Platform.runLater(() -> {
        for(int i = 0; i < 5; i++){
            Node cardReset = crHand.getChildren().get(i);
            if (cardReset instanceof ImageView){
                ((ImageView) cardReset).setImage(null);
            }
        }
        for(int i = 0; i < 2; i++){
            Node cardReset = myHand.getChildren().get(i);
            if (cardReset instanceof ImageView){
                ((ImageView) cardReset).setImage(null);
            }
        }
    });
}
```

Zadanie 1 - Server-Client

W metodzie main() klienta i serwera użyć odpowiednich gotowych metod tak, aby:

- nawiązać połączenie
- klient wysłał wiadomość message i odczytał wiadomość od serwera
- serwer odebrał wiadomość i wysłał swoją

Zadanie 2 - przesyłanie obiektów.

Podobnie jak w zadaniu 1, w metodzie main() klienta i serwera użyć odpowiednich gotowych metod tak, aby:

- nawiązać połączenie
- klient wysłał wiadomość message i odczytał wiadomość od serwera
- serwer odebrał wiadomość i wysłał swoją

tylko tym razem zamiast String'a przesyłamy obiekt klasy Message2.

Zadanie 3 - wielowątkowość

Server3 dla każdego połączonego klienta startuje wątek ClientHandler do obsługi tego klienta i dodaje go do arrayListy clientHandlers. ClientHandler ciągle nasłuchuje na wiadomość od swojego klienta i jeśli jakąś dostanie rozsyła ją do pozostałych klientów.

W klasie ClientHandler uzupełnić metodę sendToAll() tak, aby każdy element arrayListy clientHandlers wysłał podany jako argument obiekt message. Zwróć uwagę, że element, który odebrał wiadomość nie powinien tego robić.

Bibliografia

- https://javastart.pl/b/java/dlaczego-javafx-jest-lepsza-od-swinga/
- https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm
- http://tutorials.jenkov.com/javafx/concurrency.html
- https://www.speedcheck.org/pl/wiki/socket/
- https://sekurak.pl/tworzenie-narzedzi-sieciowych-w-pythonie-z-uzyciem-socketow/
- http://architektura-oprogramowania.blogspot.com/p/architektura-klient-serwer.html
- https://vavatech.pl/technologie/architektura/Klient-Server
- https://www.informit.com/articles/article.aspx?p=2251210&seqNum=2
- http://tutorials.jenkov.com/javafx/overview.html
- https://docs.oracle.com/javase/9/docs/api/java/security/MessageDigest.html
- https://slideplayer.pl/slide/440309/
- https://certyfikatyssl.pl/news/protokol-ssl-a-tls.html
- https://www.ssl.certum.pl/cert/certy_informacje_co_to_jest_certyfikat_ssl/

Dziękujemy za uwagę :)