Platformy Technologiczne

Laboratorium nr 2 – Java: Gniazda sieciowe

Należy zaimplementować aplikację serwerową oraz aplikację kliencką, które umożliwią przesyłanie plików z wykorzystaniem gniazd sieciowych. Serwer powinien obsługiwać wielu klientów równocześnie z wykorzystaniem wątków.

Aplikacja nadawcy musi być zrealizowana jako aplikacja JavaFX. Wybieranie pliku do wysłania powinno być zrealizowane za pomocą komponentu `FileChooser`. Samo wysyłanie pliku musi odbywać się w osobnym wątku, aby nie blokować głównego wątku obsługi zdarzeń aplikacji. Zadanie do wykonania w osobnym wątku należy zdefiniować implementując klasę, która dziedziczy po klasie pomocniczej `Task`. Klasa `Task` implementuje interfejs `Runnable` i zawiera pomocnicze metody przydatne w kontekście aplikacji okienkowych JavaFX.

Postęp wysyłania pliku musi być prezentowany za pomocą kontrolki `ProgressBar`, której własność `progressProperty` należy powiązać z odpowiednią własnością zaimplementowanego zadania (`progressProperty`). Informacja na temat aktualnego statusu zadania w tle (np. inicjowanie/wysyłanie/zakończono) powinna być wyświetlana za pomocą komponentu `Label`, którego własność `textProperty` należy powiązać z własnością `messageProperty` zadania. Zadanie aktualizuje informacje o swoim statusie i postępie poprzez wywołania metod `updateMessage()` oraz `updateProgress()`. Przekazane wartości są automatycznie propagowane do powiązanych komponentów w interfejsie aplikacji.

Przykład powiązania własności zadania z komponentami interfejsu

```
//pola klasy kontrolera:
@FXML private Label statusLabel;
@FXML private ProgressBar progressBar;
private ExecutorService executor = Executors.newSingleThreadExecutor();

//w metodzie obsługującej wybór pliku:
Task<Void> sendFileTask = new SendFileTask(file); //klasa zadania
statusLabel.textProperty().bind(sendFileTask.messageProperty());
progressBar.progressProperty().bind(sendFileTask.progressProperty());
executor.submit(sendFileTask); //uruchomienie zadania w tle
```

Szablon klasy zadania do wykonania w tle

```
public class SendFileTask extends Task<Void> {
    File file; //plik do wysłania
    public SendFileTask(File file) { this.file = file; }

    @Override protected Void call() throws Exception {
        updateMessage("Initiating...");

        //...wysyłanie pliku...
        updateProgress(/*liczba przesłanych bajtów*/, file.length());

        return null;
    }
}
```

Serwer może być zrealizowany jako aplikacja konsolowa w trybie tekstowym. Nasłuchiwanie na połączenia od klientów odbywa się z wykorzystaniem klasy `ServerSocket`. Metoda `accept()` przyjmuje połączenie od klienta i zwraca instancję klasy `Socket` do komunikacji z nim. Obsługa klienta powinna odbywać się w osobnym wątku aplikacji.

Nasłuchiwanie na połączenia od klientów

```
try (ServerSocket serverSocket = new ServerSocket(port)) {
    while (true) {
        final Socket socket = serverSocket.accept();
        //...obsługa klienta w osobnym wątku...
    }
}
```

Do wczytywania/zapisywania plików oraz komunikacji przez gniazda sieciowe należy wykorzystać strumienie bajtowe (hierarchie `InputStream` oraz `OutputStream`). Należy również wykorzystać buforowanie stosując odpowiednie dekoratory strumieni.

Przykład odczytu/zapisu dla strumieni binarnych

```
//in – strumień wejściowy (np. z gniazda sieciowego)
//out – strumień wyjściowy (np. do pliku)
byte[] buffer = new byte[4096]; //bufor 4KB
int readSize;

while ((readSize = in.read(buffer)) != -1) {
    out.write(buffer, 0, readSize);
}
```

Wszystkie strumienie wejścia/wyjścia powiązane z plikami na dysku oraz gniazdami sieciowymi powinny być poprawnie zamykane. Należy w tym celu wykorzystać konstrukcję *try-with-resources*. Dotyczy to zarówno aplikacji serwerowej jak i aplikacji klienckiej.

Punktacja:

- odbieranie pliku od jednego klienta na raz: 2 pkt,
- odbieranie plików od wielu klientów równocześnie z wykorzystaniem kilku wątków w jednym z dwóch wariantów:
  - dla każdego klienta uruchamiany jest nowy wątek: 1 pkt,
    LUB
  - z wykorzystaniem puli wątków do obsługi klientów: 2 pkt,
- poprawna aktualizacja kontrolek `ProgressBar` oraz `Label` w aplikacji klienckiej: 1 pkt.