

## Platformy Technologiczne

### Laboratorium nr 3 – Java: Wielowątkowość w API Stream

W ramach zadania laboratoryjnego należy przygotować aplikację, która pozwala użytkownikowi na wybranie zbioru plików graficznych z dysku i konwertuje je do postaci obrazów w skali szarości. Pliki graficzne powinny być przetwarzane w sposób równoległy z użyciem API Stream.

Aplikacja powinna wykorzystywać komponent tabeli (`TableView`) prezentujący w kolejnych wierszach informacje o wybranych przez użytkownika plikach. Każdy wiersz tabeli powinien składać się z trzech kolumn, które zawierają:

- nazwę pliku,
- pasek postępu (`ProgressBar`), który wypełnia się w miarę przetwarzania danego pliku do postaci czarno-białej,
- status przetwarzania:
  - oczekuje – zanim rozpocznie się przetwarzanie danego pliku,
  - przetwarzanie... – w trakcie konwersji danego pliku,
  - zakończone – gdy dany plik zostanie przekształcony na postać czarno-białą i zapisany na dysku.

Miejsce zapisu przekonwertowanych plików powinno być wybierane przez użytkownika za pośrednictwem komponentu `DirectoryChooser`. Do wyboru plików podlegających konwersji należy wykorzystać komponent `FileChooser`. Metoda `showOpenMultipleDialog` klasy `FileChooser` pozwala na wyświetlenie okna wyboru umożliwiającego zaznaczenie wielu plików (z użyciem klawiszy Shift lub Control). Aby ograniczyć wybór tylko do plików graficznych z rozszerzeniem `.jpg` należy dodatkowo zastosować filtr, jak w przykładzie poniżej:

```
FileChooser fileChooser = new FileChooser();
fileChooser.getExtensionFilters().add(
    new FileChooser.ExtensionFilter("JPG images", "*.jpg"));
List<File> selectedFiles = fileChooser.showOpenMultipleDialog(null);
```

Należy przygotować klasę modelu danych opisującą zadanie przekonwertowania pojedynczego pliku, np. `ImageProcessingJob`. Powinna ona zawierać następujące pola:

- `file` – typu `File` – obiekt wskazujący na plik graficzny na dysku,
- `status` – typu `SimpleStringProperty` – o przedstawionym wcześniej zbiorze możliwych wartości: `oczekuje/przetwarzanie.../zakończony` w zależności od stanu przetwarzania danego zadania; pole powinno zostać zbindowane z wartością w kolumnie statusu w tabeli,
- `progress` – typu `DoubleProperty` – o wartościach z przedziału `[0, 1]`; pole powinno zostać zbindowane z wartością w kolumnie postępu przetwarzania w tabeli.

Po wybraniu plików graficznych przez użytkownika należy przygotować listę obiektów klasy `ImageProcessingJob`, które posłużą do wypełnienia tabeli. Powiązanie pól klasy `ImageProcessingJob` z kolumnami tabeli może przebiegać, jak niżej:

```
//pola klasy kontrolera:
@FXML TableColumn<ImageProcessingJob, String> imageNameColumn;
@FXML TableColumn<ImageProcessingJob, Double> progressColumn;
```

```

@FXML TableColumn<ImageProcessingJob, String> statusColumn;

@Override
public void initialize(URL url, ResourceBundle rb) {
    imageNameColumn.setCellValueFactory( //nazwa pliku
        p -> new SimpleStringProperty(p.getValue().getFile().getName()));

    statusColumn.setCellValueFactory( //status przetwarzania
        p -> p.getValue().getStatusProperty());

    progressColumn.setCellFactory( //wykorzystanie paska postępu
        ProgressBarTableCell.<ImageProcessingJob>forTableColumn());

    progressColumn.setCellValueFactory( //postęp przetwarzania
        p -> p.getValue().getProgressProperty().asObject());

    //...dalsze inicjalizacje...
}

```

Przykładowy wygląd tabeli po uruchomieniu programu widoczny poniżej (po lewej stronie w czasie przetwarzania sekwencyjnego, po prawej – równoległego z użyciem czterech wątków):

Image	Progress	Status	Image	Progress	Status
3d_dna-2560x1600.jpg	<div><div></div></div>	completed	3d_dna-2560x1600.jpg	<div><div></div></div>	waiting
avengers_age_of_ultron_team-2560x1440.jpg	<div><div></div></div>	completed	avengers_age_of_ultron_team-2560x1440.jpg	<div><div></div></div>	processing...
dawneys targaryen_season_5-2880x1800.jpg	<div><div></div></div>	completed	dawneys targaryen_season_5-2880x1800.jpg	<div><div></div></div>	processing...
fantastic_four_2015_movie-2880x1800.jpg	<div><div></div></div>	processing...	fantastic_four_2015_movie-2880x1800.jpg	<div><div></div></div>	waiting
green_seascape-2560x1600.jpg	<div><div></div></div>	waiting	green_seascape-2560x1600.jpg	<div><div></div></div>	processing...
hunger_games_mockingjay_part_2-2880x1800.jpg	<div><div></div></div>	waiting	hunger_games_mockingjay_part_2-2880x1800.jpg	<div><div></div></div>	waiting
purple-1920x1200.jpg	<div><div></div></div>	waiting	purple-1920x1200.jpg	<div><div></div></div>	processing...
sea_sunset_cosmos-1920x1080.jpg	<div><div></div></div>	waiting	sea_sunset_cosmos-1920x1080.jpg	<div><div></div></div>	waiting
tomorrowland_2015_movie-2880x1800.jpg	<div><div></div></div>	waiting	tomorrowland_2015_movie-2880x1800.jpg	<div><div></div></div>	waiting
valley_house-2880x1800.jpg	<div><div></div></div>	waiting	valley_house-2880x1800.jpg	<div><div></div></div>	waiting

Przetwarzanie plików powinno odbywać się poza wątkiem interfejsu użytkownika, aby nie został on zablokowany. Wykorzystując wyrażenia lambda, nowy wątek w tle można uruchomić następująco:

```

//metoda obsługująca kliknięcie przycisku rozpoczynającego przetwarzanie
@FXML
void processFiles(ActionEvent event) {
    new Thread(this::backgroundJob).start();
}

//metoda uruchamiana w tle (w tej samej klasie)
private void backgroundJob(){
    //...operacje w tle...
    jobs.stream().forEach(...);
}

```

Konwersja pojedynczego obrazu na postać czarno-białą może zostać zrealizowana przy użyciu przedstawionej poniżej metody, która implementuje przekształcenie postaci:

$$\text{jasność} = 0.21 * \text{kanał\_czerwony} + 0.72 * \text{kanał\_zielony} + 0.07 * \text{kanał\_niebieski}$$

Metoda wczytuje oryginalny plik z dysku do bufora w pamięci operacyjnej, a następnie alokuje drugi bufor o takim samym rozmiarze jak oryginalny obraz przeznaczony na

wersję w skali szarości. Kolejne piksele są przetwarzane sekwencyjnie według powyższego wzoru aż do przekonwertowania całego obrazu.

```
private void convertToGrayscale(
    File originalFile, //oryginalny plik graficzny
    File outputDir, //katalog docelowy
    DoubleProperty progressProp//własność określająca postęp operacji
) {
    try {
        //wczytanie oryginalnego pliku do pamięci
        BufferedImage original = ImageIO.read(originalFile);

        //przygotowanie bufora na grafikę w skali szarości
        BufferedImage grayscale = new BufferedImage(
            original.getWidth(), original.getHeight(), original.getType());

        //przetwarzanie piksel po pikselu
        for (int i = 0; i < original.getWidth(); i++) {
            for (int j = 0; j < original.getHeight(); j++) {

                //pobranie składowych RGB
                int red = new Color(original.getRGB(i, j)).getRed();
                int green = new Color(original.getRGB(i, j)).getGreen();
                int blue = new Color(original.getRGB(i, j)).getBlue();

                //obliczenie jasności piksela dla obrazu w skali szarości
                int luminosity = (int) (0.21*red + 0.71*green + 0.07*blue);
                //przygotowanie wartości koloru w oparciu o obliczoną jasność
                int newPixel =
                    new Color(luminosity, luminosity, luminosity).getRGB();

                //zapisanie nowego piksela w buforze
                grayscale.setRGB(i, j, newPixel);
            }

            //obliczenie postępu przetwarzania jako liczby z przedziału [0, 1]
            double progress = (1.0 + i) / original.getWidth();
            //aktualizacja własności zbindowanej z paskiem postępu w tabeli
            Platform.runLater(() -> progressProp.set(progress));
        }

        //przygotowanie ścieżki wskazującej na plik wynikowy
        Path outputPath =
            Paths.get(outputDir.getAbsolutePath(), originalFile.getName());

        //zapisanie zawartości bufora do pliku na dysku
        ImageIO.write(grayscale, "jpg", outputPath.toFile());
    } catch (IOException ex) {
        //translacja wyjątku
        throw new RuntimeException(ex);
    }
}
```

Należy wykorzystać API Stream, aby wykonać powyższą metodę dla wszystkich plików wybranych przez użytkownika. W czasie konwertowania obrazów postęp dla każdego z nich powinien być widoczny w tabeli w interfejsie użytkownika (jak w przykładowych tabelach przedstawionych wcześniej). Po zakończeniu konwersji plików aplikacja powinna wyświetlić czas przetwarzania obrazów, który można zmierzyć następująco:

```
long start = System.currentTimeMillis(); //zwraca aktualny czas [ms]
//...
//przetwarzanie obrazów
//...
long end = System.currentTimeMillis(); //czas po zakończeniu operacji [ms]
long duration = end-start; //czas przetwarzania [ms]
```

Użytkownik powinien mieć możliwość wyboru pomiędzy następującymi wariantami przetwarzania plików:

- przetwarzanie sekwencyjne,
- przetwarzanie współbieżne z użyciem domyślnej puli wątków (*commonPool*),
- przetwarzanie współbieżne, w którym liczba wątków roboczych jest określona przez użytkownika.

Aby skorzystać z własnej puli wątków o wybranym rozmiarze, należy wykorzystać wywołanie, jak niżej:

```
ForkJoinPool pool = new ForkJoinPool(2); //pożądana liczba wątków
pool.submit(this::backgroundJob);
```

Strumienie współbieżne uruchamiane w ramach metody `backgroundJob()` będą wykorzystywały wątki należące do puli, w której ta metoda została wywołana, a nie w ramach domyślnej puli, jak miało to miejsce wcześniej.

### Punktacja

Na potrzeby prezentacji zadania należy przygotować zbiór testowych plików graficznych, które będą przetwarzane przez program. Pliki powinny posiadać rozdzielczość przynajmniej 1920x1080, aby czas przetwarzania pojedynczego pliku był zauważalny dla użytkownika.

Za poszczególne etapy zadania przysługuje następująca liczba punktów:

- wybór wielu plików z dysku i wyświetlenie ich w tabeli – 1 pkt,
- sekwencyjne przetwarzanie plików z aktualizacją postępu operacji dla każdego obrazu w tabeli – 2 pkt,
- równoległe przetwarzanie plików – 1 pkt,
- porównanie czasów przetwarzania dla strumienia sekwencyjnego oraz strumieni współbieżnych z różnymi liczbami wątków – 1 pkt.