Platformy Technologiczne

Laboratorium 6

Java: Wyrażenia lambda, API Stream i wielowątkowość

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 powinien składać się z trzech kolumn, zawierających:

- 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.ExtensionFilter filter =
                  new FileChooser.ExtensionFilter("JPG images", "*.jpg");
fileChooser.getExtensionFilters().add(filter);
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.../zakończone 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 z kolumnami tabeli może przebierać następująco (wykorzystano wyrażenia

lambda zamiast wcześniejszych anonimowych klas wewnętrznych dla skrócenia zapisu):

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

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

	//...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
3d_dna-2560x1600.jpg		completed
avengers_age_of_ultron_team-2560x1440.jpg		completed
daenerys_targaryen_season_5-2880x1800.jpg		completed
fantastic_four_2015_movie-2880x1800.jpg		processing...
green_seascape-2560x1600.jpg		waiting
hunger_games_mockingjay_part_2-2880x1800.jpg		waiting
purple-1920x1200.jpg		waiting
sea_sunset_cosmos-1920x1080.jpg		waiting
tomorrowland_2015_movie-2880x1800.jpg		waiting
valley_house-2880x1800.jpg		waiting

Image	Progress	Status
3d_dna-2560x1600.jpg		waiting
avengers_age_of_ultron_team-2560x1440.jpg		processing...
daenerys_targaryen_season_5-2880x1800.jpg		processing...
fantastic_four_2015_movie-2880x1800.jpg		waiting
green_seascape-2560x1600.jpg		processing...
hunger_games_mockingjay_part_2-2880x1800.jpg		waiting
purple-1920x1200.jpg		processing...
sea_sunset_cosmos-1920x1080.jpg		waiting
tomorrowland_2015_movie-2880x1800.jpg		waiting
valley_house-2880x1800.jpg		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
}
```

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

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

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 przetwarzanie 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 jaskość
            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). W pierwszej wersji należy

zastosować strumień sekwencyjny (wywołanie metody `stream()`) i zmierzyć łączny czas przetwarzania obrazów, przykładowo:

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

Zmierzony czas należy wyświetlić użytkownikowi. Następnie należy wykorzystać strumień współbieżny (wywołanie metody `parallelStream()`) i porównać czas przetwarzania z uzyskanym wcześniej. Należy określić, ile wątków jest domyślnie wykorzystywanych do konwersji obrazów przez współbieżny strumień. Aby skorzystać z własnej puli wątków, o wybranym rozmiarze, należy dotychczasowe wywołanie:

```
new Thread(this::backgroundJob).start();
```

zastąpić następującym:

```
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 przynajmniej rozdzielczość FullHD (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.