Techniki Komponentowe
Dokumentacja Techniczna
Komponenty 4 i 12

Mateusz Biedak
Mateusz Zawiliński

1. **Opis komponentów.**

Komponent 4 odpowiada za wizualizację aktualnej wartości otrzymywanej stale z wybranego urządzenia. Aktualne wartości przedstawione są w formie termometru. Z lewej strony widoczna jest skala wartości, która ustalana jest przy tworzeniu danego termometru. Z prawej strony widać kolejne poziomy wartości, które po przekroczeniu zmieniają kolor wypełnienia termometru by dać użytkownikowi znać w jakim przedziale znajduje się aktualna wartość. U dołu widać aktualną wartość w postaci liczby całkowitej z ustalonego wcześniej przedziału.

Komponent 12 odpowiada za graficzne przedstawienie danych liczbowych, otrzymanych z wybranego urządzenia, w postaci zbioru kół. Każde koło odpowiada innej danej, która identyfikowana jest po unikalnym ID podanym przez użytkownika przy przesyłaniu jej do komponentu. Dana oprócz ID posiada też wartość liczbową. Promień koła zależny jest od procentowego udziału jego wartości liczbowej w sumie wartości wszystkich obecnych kół. Każde koło oprócz unikalnego ID posiada unikalny kolor w celu łatwego zidentyfikowania go. Po najechaniu myszką na wybrane koło wyświetli nam się popup zawierający: ID, wartość liczbową i procentowy udział wybranego obiektu.

2. **Wymagania funkcjonalne komponentów**

Dla komponentu 4:

- wyświetlanie ostatnio odebranej wartości
- podział skali na progi, od których zależy kolor wypełnienia termometru
- przyjmowanie danych w formie double i wyświetlanie ich w postaci najbliższej liczby naturalnej

Dla komponentu 12:

- przyjmowanie danych w postaci pary: id(String) oraz value(float)
- wyświetlanie każdej otrzymanej danej w postaci koła o unikalnym kolorze do momentu usunięcia tej danej
- udostępnienie możliwości dodawania nowych kół lub modyfikowania i usuwania istniejących
- stałe odświeżanie widoku po operacjach dodania/usunięcia/modyfikacji danych

3. Cechy techniczne komponentów

Komponent 12 składa się z 3 klas (podejście typu MVC):

- *CircleData* – (można określić jako Model) To klasa przechowująca dane dotyczące pojedynczego koła na naszym wykresie. Używana w pozostałych klasach w celu sprawnego zarządzania danymi kołami. Przechowuje takie informacje jak: id, wartość, wartość procentowa. Dodatkowo zawiera informacje niezbędne do poprawnego wyświetlania danego koła: położenie na wykresie, kolor, początkowy kierunek przy kolizji. Przechowywane jest tu także odwołanie do obiektu Circle (javafx), który reprezentuje grafikę koła na wykresie. Dzięki przechowywaniu tych danych zgrupowanych w tej klasie mamy do nich łatwy dostęp. Wykorzystywana do obliczania kolizji kół. Obiekty tej klasy tworzone są w metodzie addNewCircle klasy CircleDataController.
- *CircleDataController* – (można określić jako Controller) klasa sterująca naszymi informacjami o kołach. Przetrzymywana tu jest aktualna kolekcja danych w postaci mapy o parach ID : CircleData. Znajduje się tu aktualna suma wartości wszystkich kół, używana do obliczenia ich procentowych udziałów. Jest tu także kolejka kolorów obliczana przy inicjalizacji klasy by później przypisywać unikalny kolor każdemu obiektowi. Znajdziemy tu metody faktycznie dodające/usuwające/modyfikujące koła na naszym wykresie.
- *BubbleBandwidth* – (można określić jako View i nieco Controller) Klasa w głównej mierze zajmująca się graficznym przedstawieniem danych. Rozszerza klasę Application (javafx), definiuje całą scenę naszego wykresu (Pane, Stage). Zawiera listę 8 kierunków, w które koła mogą się przepychać (dla uproszczenia kolizji kół). To ta klasa tak naprawdę odpowiada za wystartowanie komponentu, dlatego to ona mus zostać zainstancjonowana i następnie uruchomiona za pomocą metody startApplication. Jednak ponieważ opiera się o javafx wątek uruchamiający ten komponent ulega „zawieszeniu”(stałe odświeżanie

obrazu) i wszelkie inne operacje z nim związane muszą toczyć się w innym wątku. Klasa ta zawiera CircleDataController i korzysta z jej funkcji oraz udostępnionych danych (mapa kół) i na ich podstawie aktualizuje widok. Metoda odpowiadająca za aktualizację sprawdza stale flagę oznajmująca nadejście zmian i dopiero w momencie odczytania potrzeby aktualizacji dokonuje jej (tak więc widok nie jest niepotrzebnie aktualizowany co ustalony przedział czasu). Każdy update to narysowanie kół w ich nowych pozycjach obliczonych poprzez ich kolizję między sobą. Kolizja może nastąpić w przypadku dodania nowego koła lub zmiany rozmiarów istniejącego i odbywa się gdy 2 koła stykają się, wtedy to o niższym ID(większy priorytet) odpycha to o wyższym. Dzięki takiemu podejściu użytkownik nie zgubi z pola widzenia kół które obserwował.
Po otrzymaniu danych sprawdzamy je w celu wybrania jaką operację należy dokonać.

- Jeżeli otrzymamy wartość większą od 0 to sprawdzamy czy koło związane z dostarczonym id isntieje. Jeśli tak to modyfikujemy jego wartość, jeśli nie to tworzymy nowę z podaną wartością.
- Jeżeli otrzymujemy wartość mniejszą od zerato spradzamy czy jest równa -1 i czy związane z dostarczonym id koło istnieje. Jeśli tak to usuwamy dane koło, jeśli nie to zwracamy wyjątek z odpowiednią informacją.

Nad każdym kołem po najechaniu na nie myszką wyświetli się tooltip z informacjami takimi jaki: id, wartość i wartość procentowa danego koła.
Widok można zoomować poprzez użycie scrolla myszki oraz przeciągać za pomocą lewego przycisku myszki gdy chwycimy obszar kół i przeciągniemu.

Komponent 4 składa się z jednej klasy TermometerPanel, która rozszerza klasę biblioteki Swing - JPanel oraz implementuje interfejs Runnable.
Właściwości JPanelu sprawiają, że łatwo jest dołączyć ten komponent do głównej sceny w postaci JFrame a także łatwo jest do niego dodawać komponenty z biblioteki JFreeChart.
Interfejs Runnable przydał się z tego względu, że metoda run() co 100 milisekund odświeża wnętrze panelu tak aby wyświetlana w nim była akutalna temperatura.
Klasa ta posiada wbudowany obiekt klasy ThermometerPlot, któy jest właściwym termometrem podłączonym do tego panelu, i do którego są przekazywane kolejne temperatury.

Termometr posiada minimalną i maksymalną temperaturę jaką ma wyświetlać, którą podaje się w jego konstruktorze, a także podzakresy, które definiują:

- niską temperaturę: 0 - 50 % maksymalnej temperatury
- średnią temperaturę: 50% - 75% maksymalnej temperatury
- wysoką temperaturę: 75% - 100% maksymalnej temperatury

Konstruktor tej klasy przyjmuje jako parametry:

- tytuł wykresu
- minimalną temperaturę
- maksymalną temperaturę

W jego wnętrzu ustawiane są temperatury oraz odpalana jest metoda thermometerInit() ustawiająca odpowiednio wszystkie parametry początkowe termometru takie jak:

- podzasięgi
- kolory odpowiednich podzasięgów
- czcionki wyświetlonych napisów
- rozmiar termometru w wyświetlanym panelu

Metoda run() tak jak już zostało to wspomniane, co 100 milisekund odświeża wnętrze panelu metodą repant().
Jedyną publiczną metodą potrzebną do ustawiania temperatury z zewnątrz jest metoda setTemperature(double temperature).
Sprawdza ona czy podana temperatura nie przekroczyła zakresu z jednej lub drugiej strony i jeśli tak to sprowadza ją ona do minimalnej lub maksymalnej wartości.
Temperatury wyświetlane są w liczbach całkowitych dlatego podana wartość zmiennoprzecinkowa jest rzutowana na typ podstawowy "int".
Temperatura zostaje ustawiona obiektowi klasy ThermometerPlot i przy następnym odświeżeniu panelu jest już widoczna.

4. Instrukcja użytkowania komponentów

W celu użycia **komponentu 4** należy zainicjować obiekt klasy Jframe z odpowiednimi ustawieniami layoutu. Następnie stworzyć porządaną liczbę przedstawicieli klasy TermometerPanel z odpowiednimi danymi (tytuł, minimum, maximum). Do zawartości utworzonego Jframe'a dodajemy utworzone panele, ustawiamy rozmiary okna według upodobania oraz 2 dodatkowe parametry: visible oraz resizable dotyczące widoczności i skalowalności danego Jframe'a.

Tak skonfigurowany Jframe po uruchomieniu aplikacji pokaże nam nasze termometry, aby aktualizować ich dane potrzeba tylko użyć metody setTemperature której przekazujemy nową wartość do ustawienia.

W celu użycia **komponentu 12** należy utworzyć przedstawiciela klasy BubbleBandwidth i utworzyć jeden wątek. Wątek będzie odpowiedzialny za proces rysujący, gdyż javafx działa na tej zasadzie, że wątek uruchamiający klasę typu Application działa już tylko w zakresie działań tej klasy (odpowiada za aktualizację grafiki), zostaje w pewien sposób zawieszony. Następnie możemy już w naszym wątku głównym aktualizować informacje wykresu poprzez metodę newData przyjmującą 2 parametry: id oraz wartość. Należy pamiętać że id to String a wartość to liczba typu float.

Jeżeli chcemy dodać nowe koło do wykresu wystarczy przekazać nieistniejące jeszcze id oraz wartość liczbową większą od 0, po niedługiej chwili koło pojawi się na wykresie. Jeżeli chcemy zmodyfikować już istniejące koło podajemy następujące dane: istniejące id oraz wartość liczbową większą od 0. Usunięcie wskazanego koła odbywa się po przesłaniu jego id wraz z liczbą -1. Jeżeli prześlemy wartość liczbową mniejszą od zera i różną od -1 to rzucony zostanie Exception (który należy obsłużyć) z odpowiednią wiadomością. Dodatkowo jeżeli podamy wartość -1 ale nieistniejące id także otrzymamy Exception z informacją mówiącą jaki błąd został popełniony.

5. Przykładowe kody dla użycia komponentów

– Zapoczątkowanie graficznego przedstawienia termometru – wyświetlenie gotowego Jframe'a z termometrem:

```
final JFrame frame = new JFrame()
final TermometerPanel panel = new TermometerPanel("Temperature 1", 0.0, 200.0);
frame.getContentPane().setLayout(new BorderLayout(5, 5));
frame.setDefaultCloseOperation(3);
frame.setTitle("Thermometer");
frame.getContentPane().add(panel, BorderLayout.WEST);
frame.setSize(1400, 490);
frame.setVisible(true);
frame.setResizable(false);
```

– Przesłanie nowych danych do termometru:

```
panel.setTemperature(random.nextDouble() * 200.0);
```

- Inicjalizacja wykresu z kołami:

```
final BubbleBandwidth bubbleBandwidth = new BubbleBandwidth();
final Thread thread3 = new Thread(new Runnable() {
    @Override
    public void run() {
        bubbleBandwidth.startApplication();
    }
});
thread3.start();
```

- Przesłanie danych do wykresu z kołami:

```
    String id = String.valueOf(random.nextInt(20));
    float value = (float)((random.nextFloat() * 200.0) + 1.0);
    bubbleBandwidth.newData(id, value);
```