Metode Avansate de Programare
Fluxuri în Java. Interfața grafică cu utilizatorul

Arthur Molnar
arthur.molnar@ubbcluj.ro

Universitatea Babeș-Bolyai

2023

Privire de ansamblu

1. Fluxuri Java 8
   - Creare
   - Ordinea de procesare
   - Operații cu fluxuri
2. Interfețe grafice

Fluxuri I

- `java.util.Stream` - o secvență de elemente care acceptă mai multe tipuri de operații care pot și efectuate pe acele elemente.
- Operațiile pe fluxuri pot fi:
  - *intermediare* - acestea returnează un flux, astfel că mai multe operații intermediare pot fi înlănțuite.
  - *terminale* - pot returna ori `void` ori un tip de dată care nu reprezintă un flux.
- Un lanț de operații pe un flux pot fi referite și utilizând termenul de pipeline de operații (eng. *operation pipeline*).
- Toate fluxurile sunt create pe baza unei surse, care este de tipul `java.util.Collection` (o instanță a unei clase de tip `List` sau `Set`, dar nu și `Map`).

Fluxuri II

```
List<String> names =List<String> names =
                Arrays.asList("Barbara", "James", "Brooke",
                              "Emilia", "Boris");

    names.stream()
         .filter(s -> s.startsWith("B"))
         .map(String::toUpperCase)
         .sorted()
         .forEach(System.out::println);

// Rezultat:
// BARBARA
// BORIS
// BROOKE
```

Fluxuri III

- De regulă, operațiile pe fluxuri primesc un parametru de tip expresie lambda ce reprezintă instanța unei interfețe funcționale care specifică comportamentul operației.
- O operație trebuie să fie:
  - *non-interferentă* - (eng. *non-interfering*) nu modifică sursa de date a fluxului (lista sau mulțimea dată).
  - *fără stare* - (eng. *stateless*) execuția sa este deterministă (produce mereu același rezultat, date fiind aceleași date de intrare) și nu depinde de variabile sau stări exterioare care se pot modifica pe durata execuției.
- Fluxurile nu pot fi reutilizate. În momentul apelării unei operații terminale, fluxul este închis.

Creare

- Fluxurile pot fi create:
  - Folosind metoda `stream()`, pornind de la o listă sau mulțime.

```
List<String> names =
    Arrays.asList("Barbara", "James");
names.stream()
        . // restul operatiilor
```

  - Folosind metoda `Stream.of()`, pornind de la o referința unui obiect.

```
Stream.of("Barbara", "James")
        . // restul operatiilor
```

Ordinea de procesare

- Operațiile intermediare vor fi executate doar atunci când o operație terminală este prezentă (evaluare târzie) (eng. *lazy evaluation*).
- Fiecare operație intermediară creează un f lux nou, stochează funcția/oper furnizată și returnează noul flux.
- Pipeline-ul de procesare acumulează aceste fluxuri nou create.
- În momentul apelării unei operații terminale se începe traversarea fluxurilor, funcțiile asociate fiind rulate una câte una.
- Elementele avansează de-a lungul lanțului de procesare în mod vertical; fiecare element este procesat în ordine, și doar după efectuarea tuturor operațiilor necesare pe un element este demarată procesarea elementului următor.

Funcția `map(Function<? super T,? extends R> mapper)`

- Funcția `map(Function<? super T,? extends R> mapper)` - acceptă o expresie lambda ca unic argument și modifică fiecare element al fluxului în concordanță cu această operație.
- Returnează un flux nou ce constă din rezultatele aplicării funcției date pe elementele fluxului.
- Este o operație intermediară.

Funcția `filter(Predicate<? super T> predicate)`

- Funcția `filter(Predicate<? super T> predicate)` acceptă ca unic argument o expresie predicat (trebuie să returneze o valoarea booleană).
- Evaluarea predicatului pe elementul fluxului decide dacă acesta va rămâne asociat fluxului rezultant.
- Este o operație intermediară.

Funcția `sorted(Comparator<? super T> comparator)`

- Funcția `sorted(Comparator<? super T> comparator)` - returnează un flux ce constă din elementele sortate ale fluxului conform cu ordinea lor naturală sau conform instanței `Comparator` transmise ca parametru.
- Este o operație intermediară.

Funcția `reduce(T identity, BinaryOperator<T> accumulator)` I

- Funcția permite calcularea unui rezultat utilizând toate elementele fluxului.
- `reduce(T identity, BinaryOperator<T> accumulator)` - agreghează fluxul într-un rezultat
- Poate avea următorii parametri:
  - Un operator binar ce funcționează ca un acumulator. În cazul unui operator binar numeric, valoarea de început a acestuia va fi 0. Pentru un operator bazat pe un șir de caractere, valoarea de început este șirul vid.
  - O identitate și un acumulator. Identitatea reprezintă valoarea inițială a reducerii și rezultatul implicit în cazul în care fluxul nu conține elemente. Acumulatorul este un operator binar.

Funcția `reduce(T identity, BinaryOperator<T> accumulator)` II

- Operația *reduce()*, apelată cu un singur parametru returnează un obiect de tipul `java.util.Optional<T>`
- Aceasta este o clasă utilizată pentru a reprezenta dacă o valoare este prezentă sau absentă (poate conține sau nu o valoare nenulă).
- Dacă o valoare este prezentă, funcția `isPresent()` va returna `true` iar funcția `get()` va returna valoarea.
- `reduce()` este o operație terminală
- Exemple de operații de reducere predefinite: `average()`, `sum()`, `min()` `max()`, `count()`.

Operațiile din familia `*match`

- Acestea sunt operații terminale.
- Returnează `true` sau `false` depinzând dacă obiectele din flux satisfac criteriul dat.
- `allMatch()`, `anyMatch()`, `noneMatch()`

Operațiile collect (`collect(Collector<? super T,A,R> collector)` și `collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)`)

- Acestea primesc elementele dintr-un flux și le stochează într-o colecție.
- Acestea sunt operații terminale.

JavaFX

- JavaFX este un set de unelte pentru dezvoltarea aplicațiilor cu interfață grafică în Java.
- Integrează posibilități pentru dezvoltarea graficii 2D și 3D, a graficelor, audio - video precum și includerea componentelor web încorporate (eng. *embedded web components*) (script-uri Javascript, cod HTML5).
- Include componente pentru crearea interfeței grafice și permite administrarea aspectului acestora prin utilizarea fișierelor CSS.
- Este portabil, putând fi utilizat pe desktop, browser, dispozitive mobile, TV, console de jocuri.
- Asigură interoperabilitatea cu Swing. Setul de unelte Swing face parte din clasele fundamentale ale platformei Java (eng. *Java Foundation Classes*) și poate fi utilizat pentru crearea aplicațiilor grafice bazate pe interacțiunea cu ferestre

Instalarea și configurarea cu *IntelliJ IDEA*

1. La crearea unui proiect nou JavaFX în IntelliJ (**File** → **New** → **Project** → **JavaFX**), toate bibliotecile necesare vor fi instalate în mod automat pe sistem.

Aplicațiile JavaFX I

- O aplicație JavaFX conține una sau mai multe obiecte de tipul `javafx.stage.Stage`, care corespund ferestrelor pe desktop.
- O aplicație JavaFX are un obiect primar de tipul `javafx.stage.Stage`, obiect creat de runtime-ul JavaFX.
- Fiecare obiect `Stage` are atașat un obiect de tipul `javafx.scene.Scene`.
- Obiectul `Scene` este necesar pentru a afișa conținut pe *Stage*.
- Un *Stage* poate afișa o singură scenă la un moment dat (un singur *Scene*), dar scenele afișate pot fi schimbate în timpul execuției programului.
- Fiecare scenă are atașat un graf al scenei, care menține un graf de controale și moduri de amplasare (eng. *layouts*).

Aplicațiile JavaFX II

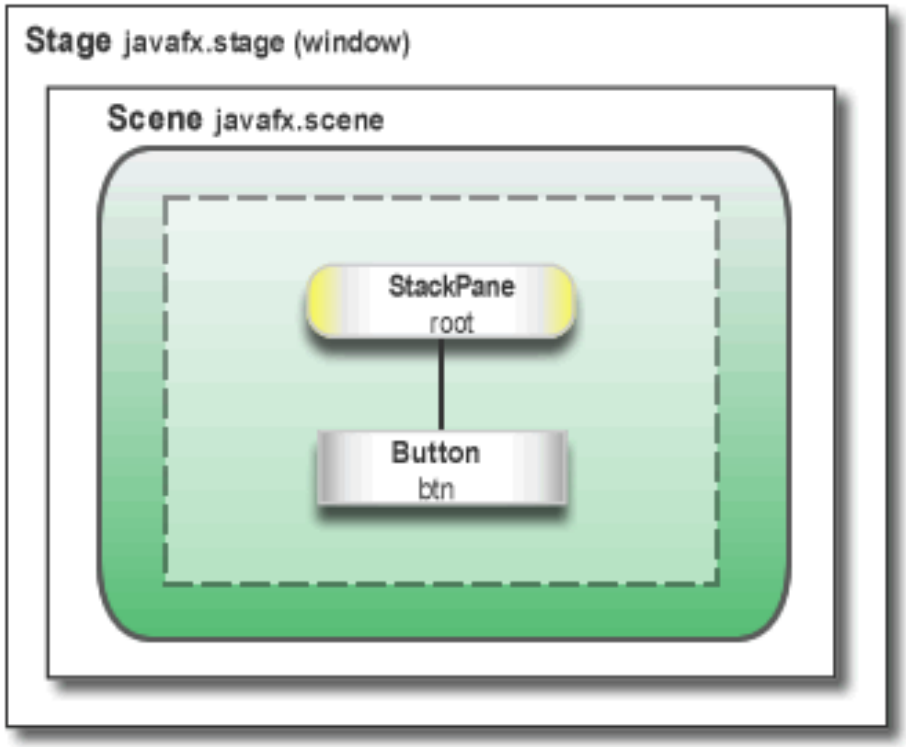

Figure: Sursa figurii: Exemplu pentru graful scenei

Aplicațiile JavaFX III

- Toate componentele atașate grafului scenei se numesc noduri. Acestea sunt clase derivate din clasa `javafx.scene.Node`
- Există două tipuri de noduri:
  - Noduri părinte, care generează o ierarhie de noduri.
  - Noduri frunză, care nu pot fi părinți pentru alte noduri.
- Interfața grafică este organizată conform șablonului de proiectare Composite (mai multe detalii aici - `https://refactoring.guru/design-pa composite`).
- Componente pentru dispunere (eng. *layout*) - acestea conțin alte componente, ele fiind respsonsabile de modul în care componentele conținute vor fi afișate pe ecran (poziționare, dimensiune relativă, distanța dintre componente etc.)
- Controale (eng. *Controls*) - acestea furnizează posibilitatea de a controla interfața grafică (ex: `Button`, `CheckBox`, `Label`, `Spinner`, `TableView`, `TextFields` și multe altele).

Aplicațiile JavaFX IV

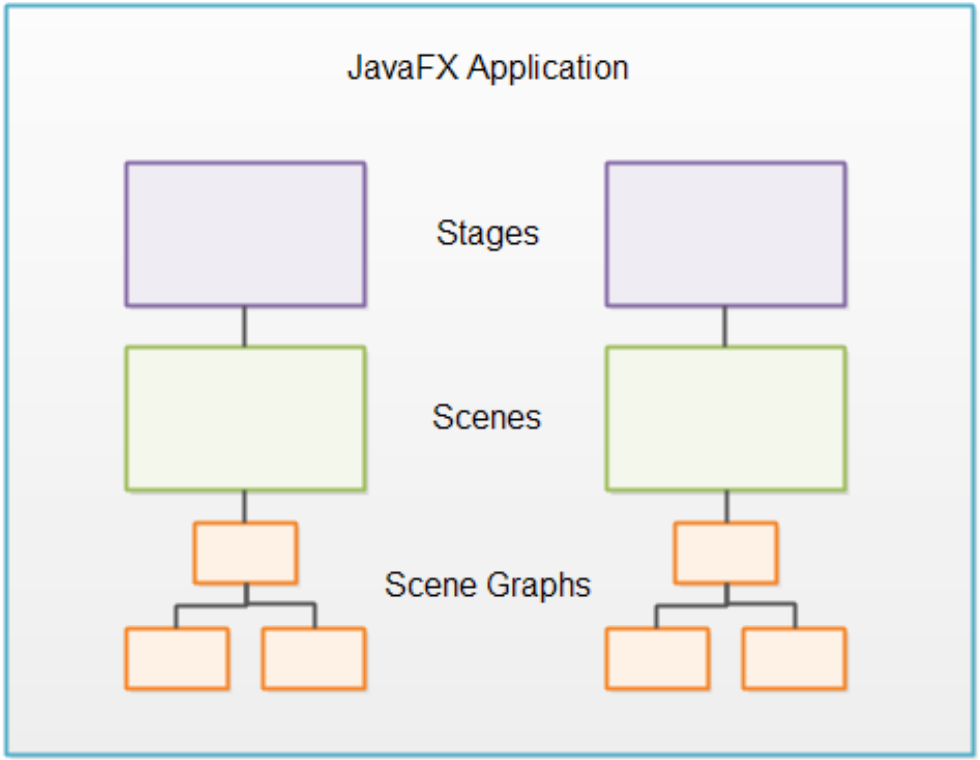

Figure: Sursa figurii: Structura generală a unei aplicații JavaFX.

Aplicațiile JavaFX V

- *Diagrame* - componente pentru desenarea digramelor (eng. charts): `BarChart`, `PieChart`, `ScatterChart` și altele.
- *Grafică 2D și 3D* - pentru desenarea pe ecran.
- *Audio și video* - funcționalități pentru redarea conținutului audio și video în cadrul aplicațiilor JavaFX.
- *WebView* - o componentă care permite afișarea paginilor web. Aceasta permite utilizarea conținutului web în cadrul unei aplicații desktop.

Clasa `javafx.application.Application` I

- Clasa utilizată pentru lansarea în execuție a unei aplicații JavaFX trebuie să fie derivată din clasa `javafx.application.Application`.
- Toate clasele derivate din clasa `Application` vor implementa metoda `start(Stage stage)`.
- Obiectul de tipul *Stage* este creat de biblioteca JavaFX.
- Metoda `show()` trebuie apoi apelată pe obiectul `Stage` pentru a-l face vizibil pe ecran.
- O aplicație JavaFX poate fi pornită fără o metodă `main()`, dar de regulă aceasta este adăugată în cazul în care se utilizează parametri în linia de comandă.
- Metoda statică `launch()` din clasa `Application` lansează aplicația în execuție.

Clasa javafx.application.Application II

```
public class Main extends Application {

    @Override
    public void start(Stage stage) throws Exception{
        stage.setTitle("Hello World");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Stage

- Pentru a putea afișa elemente ale interfeței grafice în cadrul unui `Stage`, trebuie creat un obiect de tipul `javafx.scene.Scene` și setat în cadrul `Stage`-ului dorit.
- Titlul și poziția ferestrei pe ecran pot fi setate folosind metodele `setTitle(...)`, `setX(...)`, `setY(...)`
- Modalitatea setată pe obiectul *Stage* determină dacă fereastra va fi modală (o fereastră modală blochează afișarea altor ferestre ale aceleeași aplicații în fața sa) prin intermediul metodei `initModality()`.
- Un obiect de tip *Stage* poate aparține unui alt `Stage`, legătura fiind făcută prin apelarea metodei `initOwner()`.
- Un `Stage` poate fi decorat prin utilizarea metodei `initStyle()`.

Scena și graful de scenă I

- Un obiect `Scene` este la rădăcina grafului de scenă, acesta conținând în mod tranzitiv toate componentele vizuale ale interfeței grafice.
- Graful de scenă include toate nodurile atașate scenei.
- Graful de scenă poate avea un singur nod rădăcină, toate celelalte noduri fiind atașate nodului rădăcină și formând o structură sub formă de arbore.

Scena și graful de scenă II

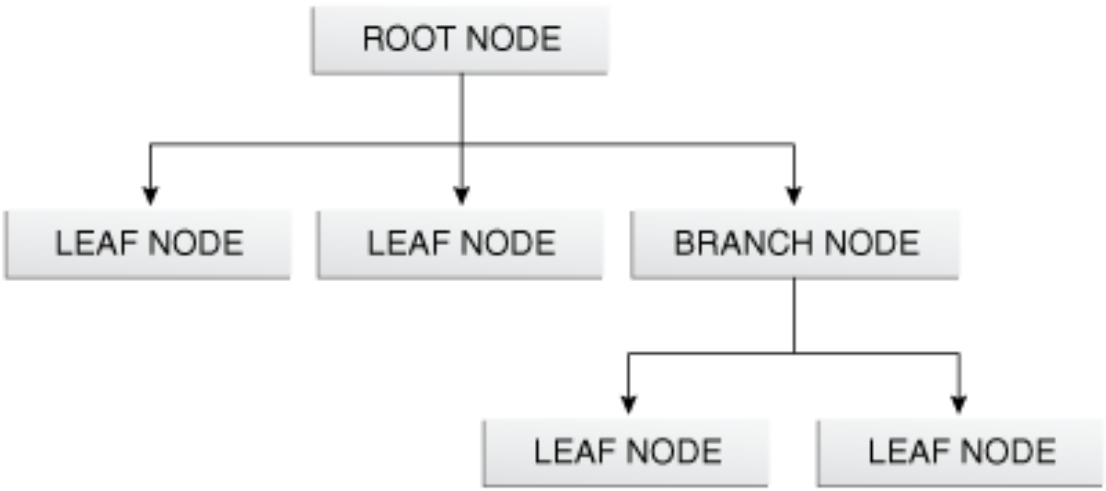

Figure: Sursa figurii: Diagrama compunerii nodurilor unei scene JavaFX.

Scena și graful de scenă III

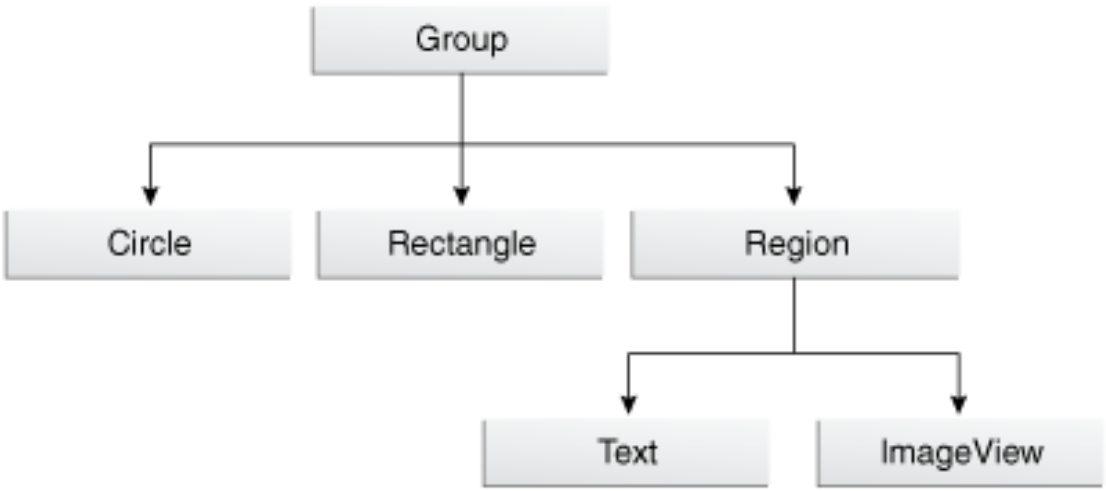

Figure: Sursa figurii: Un exemplu ce utilizează componente specifice.

Scena și graful de scenă IV

```
public void start(Stage stage) throws Exception{
        stage.setTitle("JavaFX application :)");
        stage.setX(400);
        stage.setY(400);

        Group root = new Group();
        Rectangle r = new Rectangle(25,25,100,100);
        r.setFill(Color.BLUE);
        root.getChildren().add(r);

        Circle c = new Circle(200, 200, 40);
        c.setFill(Color.RED);
        root.getChildren().add(c);

        Label label = new Label("Hello World Label!");
        root.getChildren().add(label);

        Scene scene = new Scene(root, 400, 300);
```

Scena și graful de scenă V

```
        stage.setScene(scene);
        stage.show();
    }
```

Dispunerea controalelor vizuale I

- Clasele container care determină dispunerea elementelor interfeței grafice se numesc *layout panes* (ro. panouri de amplasare).
- De exemplu, clasa `javafx.scene.layout.BorderPane` furnizează 5 regiuni pentru amplasarea nodurilor JavaFX (elemente ale interfeței grafice):

Dispunerea controalelor vizuale II

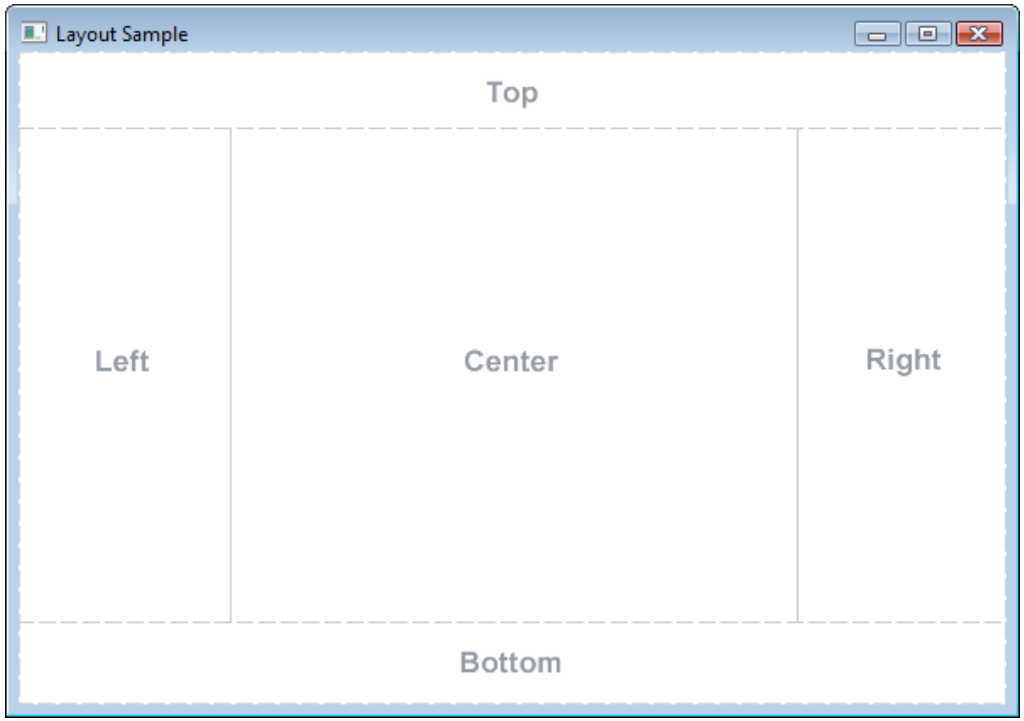

Figure: Sursa figurii: Border Pane.

- Alte clase care pot fi utilizate pentru dispunere elementelor pe ecran:
- `javafx.scene.layout.HBox` - nodurile sunt aranjate pe un singur rând.

Dispunerea controalelor vizuale III

- `javafx.scene.layout.VBox` - nodurile sunt aranjate pe o singură coloană.
- `javafx.scene.layout.StackPane` - nodurile sunt stivuite unul peste celălalt, fiind util atunci când se dorește suprapunerea vizuală a componentelor.

Figure: Sursa figurii: Stack Pane.

- `javafx.scene.layout.GridPane` - permite crearea unei grile de rânduri și coloane în care pot fi amplasate nodurile.
- `javafx.scene.layout.FlowPane` - nodurile sunt amplasate în mod consecutiv.

Dispunerea controalelor vizuale IV

- `javafx.scene.layout.TilePane` - similar cu `FlowPane`, însă fiecare celulă are aceeași dimensiune.

Controalele JavaFX UI

- Reprezintă elementele de bază ale unei aplicații cu interfață grafică.
- Fiecare control UI este un nod al grafului scenei.
- Controalele pot fi manipulate de utilizatorul aplicației.
- Documentația tuturor controalelor este disponibilă aici.