NetBeans Debugging

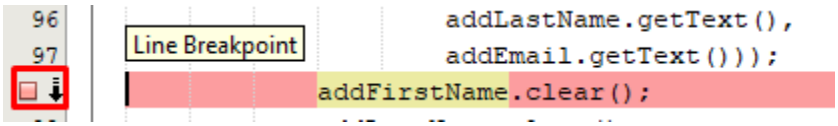
Depanarea(debugging-ul) reprezintă procesul de identificare și înlăturare a erorilor dintr-un program la runtime.

Pentru diminuarea comportamentului eronat al unui program, se recomandă tratarea corespunzătoare a situațiilor excepționale (semnalate prin mecanisme bazate pe exceptii, în cazul Java). În programele mari se pune problmea localizării erorilor. De exemplu, dacă la sfârșitul unei interacțiuni complexe cu o interfață grafică, se obține un comportament eronat, se va încerca reproducerea situației repetând doar anumiți pași din cei anteriori.

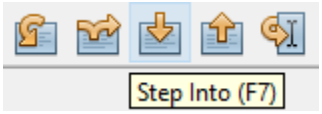
Sarcina este cu atât mai greu de îndeplinit cu cât anumite bug-uri au o comportare nedeterministă, putând apărea în momente diferite de la o rulare la alta a programului, sau chiar deloc. Cel mai adesea, în această categorie se încadrează problemele de sincronizare (ex. deadlocks).

Terminologie:

- *Breakpoint* – punctul în care execuția programului va fi suspendată, pentru analiza detaliată. Pot fi:
  - ✓ *Instruction breakpoint* - instructiune indicată de utilizator;
  - ✓ *Data breakpoint* – eveniment din execuția programului, precum modificarea unei locații de memorie.

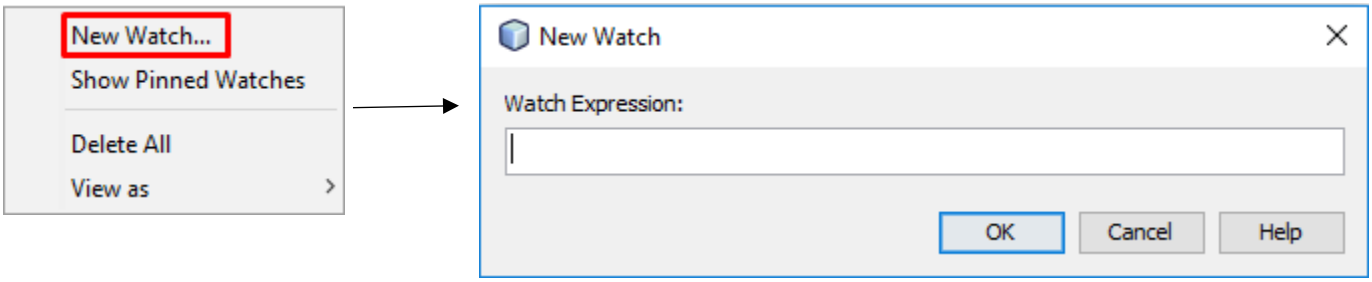


- *Stepping* – execuția instrucțiunii curente, în scopul evaluării individuale. Dacă acesta este un apel de funcție, se poate face:



  - ✓ *Step into* – se va sări la prima instrucțiune din corpul funcției;
  - ✓ *Step over* – se va considera apelul ca fiind atomic, și se va trece la următoarea.

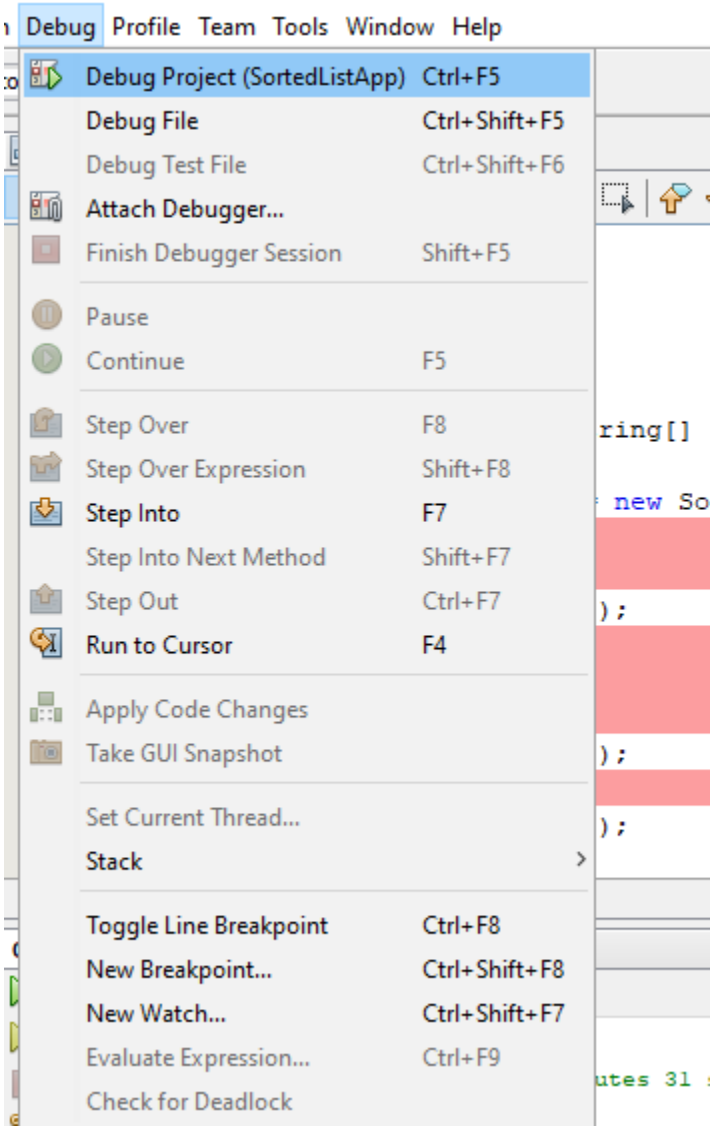
- *Watch* – mecanismul de urmărire a valorii unor variabile/expresii, pe măsură ce acestea se modifică. Pentru a adăuga în fereastra de watch variabile, putem fie să:
  - ✓ Click dreapta -> *New watch;* în fereastra apărută, vom tasta numele varibilei.



- ✓ Sau, selectăm varibila, și o aducem în fereastra de watch (drag and drop).

IDE-ul NetBeans oferă modul de debug, ce îți permite să plasezi breakpoint-uri in codul sursă, să adaugi variabile în watch, să parcurgi codul pas cu pas etc.

Pentru a intra în modul debug:



Sau, putem apăsa combinația de taste ***Ctrl+F5***.

Aplicație: Proiectul ***SortedListApp*** implementează o listă înlănțuită sortată:

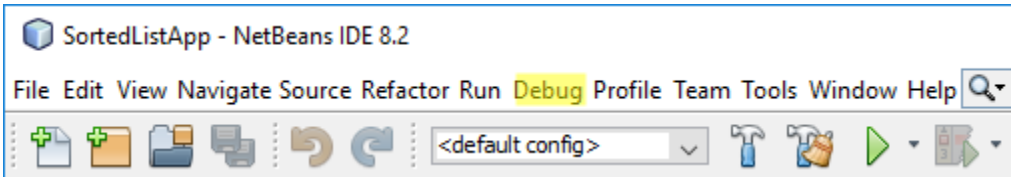
```
import java.io.*;
class Link
```

```
{ public double dData;
 public Link next;
public Link(double dd)
{ dData = dd; }
public void displayLink()
{ System.out.print(dData + " "); }
}
class SortedList
{ private Link first;
public SortedList()
{ first = null; }
public boolean isEmpty()
{ return (first==null); }
public void insert(double key)
{ Link newLink = new Link(key);
 Link previous = null;
 Link current = first;
 while(current != null && key > current.dData)
 { previous = current;
 current = current.next;
 }
 if(previous==null)
 first = newLink;
 else
 previous.next = newLink;
 newLink.next = current;
}
public Link remove()
{ Link temp = first;
first = first.next;
return temp;
}
public void displayList()
{ System.out.print("List (first-->last): ");
Link current = first;
while(current != null)
 { current.displayLink();
 current = current.next;
 }
System.out.println("");
}
}
class SortedListApp
```

```
{ public static void main(String[] args)
{SortedList theSortedList = new SortedList();
 theSortedList.insert(20);
 theSortedList.insert(40);
 theSortedList.displayList();
 theSortedList.insert(10);
 theSortedList.insert(30);
 theSortedList.insert(50);
 theSortedList.displayList();
 theSortedList.remove();
 theSortedList.displayList();
}
}
```

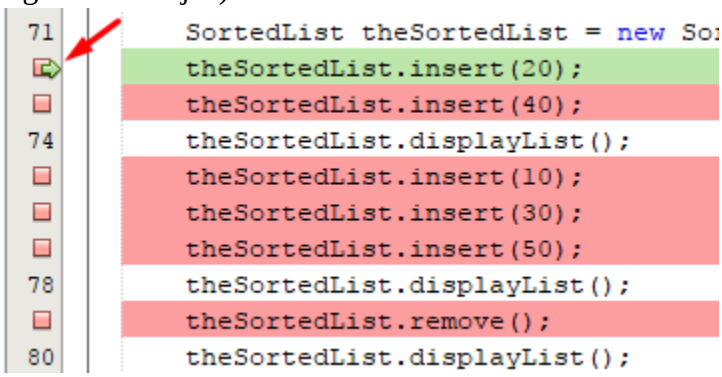
Debug:

Pentru a intra în modul debug selectăm meniul *Debug* din partea de sus:

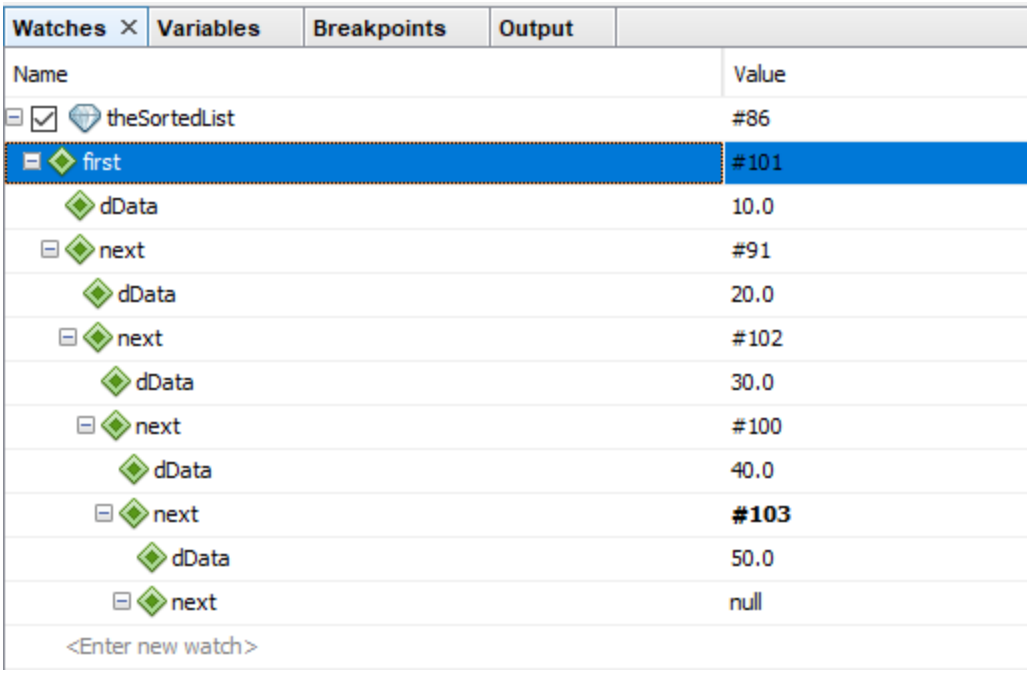


După deschiderea interfeței de debug:

1. Setăm breakpoint-uri, făcând click pe linia de cod la care vrem să se oprească (ex. Vezi figura de mai jos).

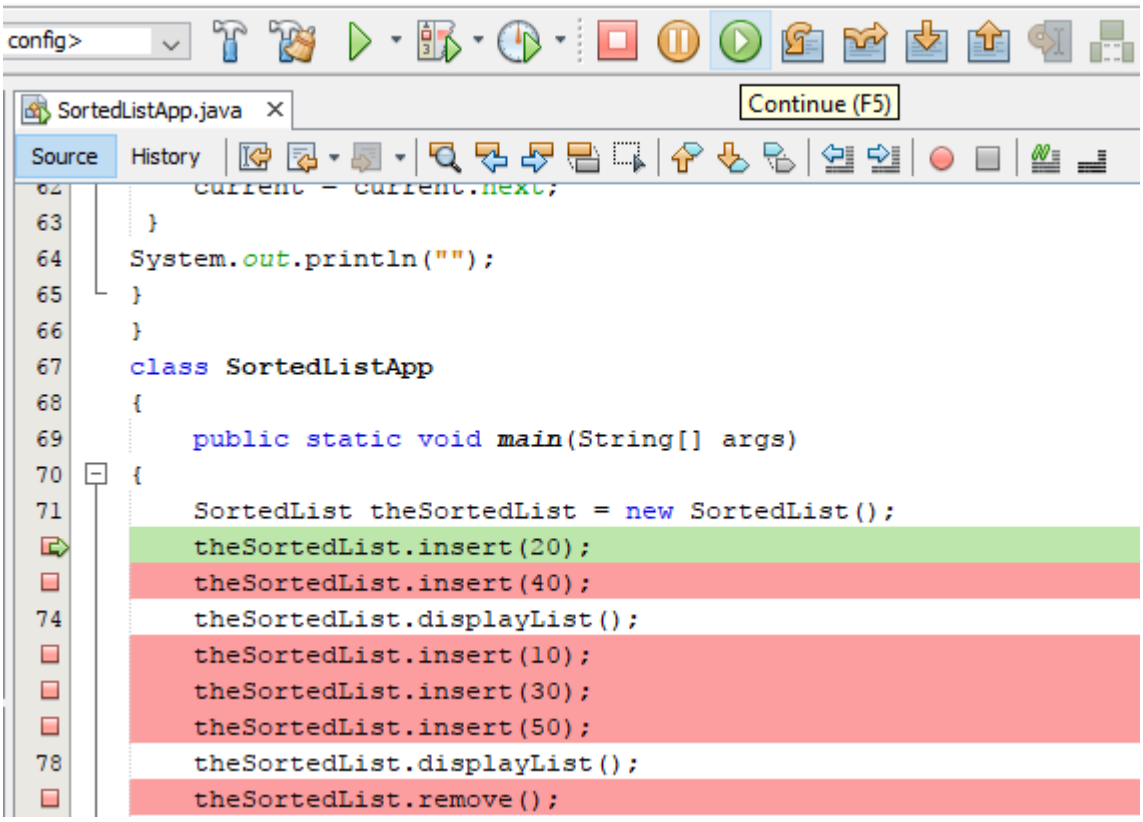


2. Adăugăm în fereastra de watch variabilele pe care dorim să le urmărim evoluția (ex. theSortedList).

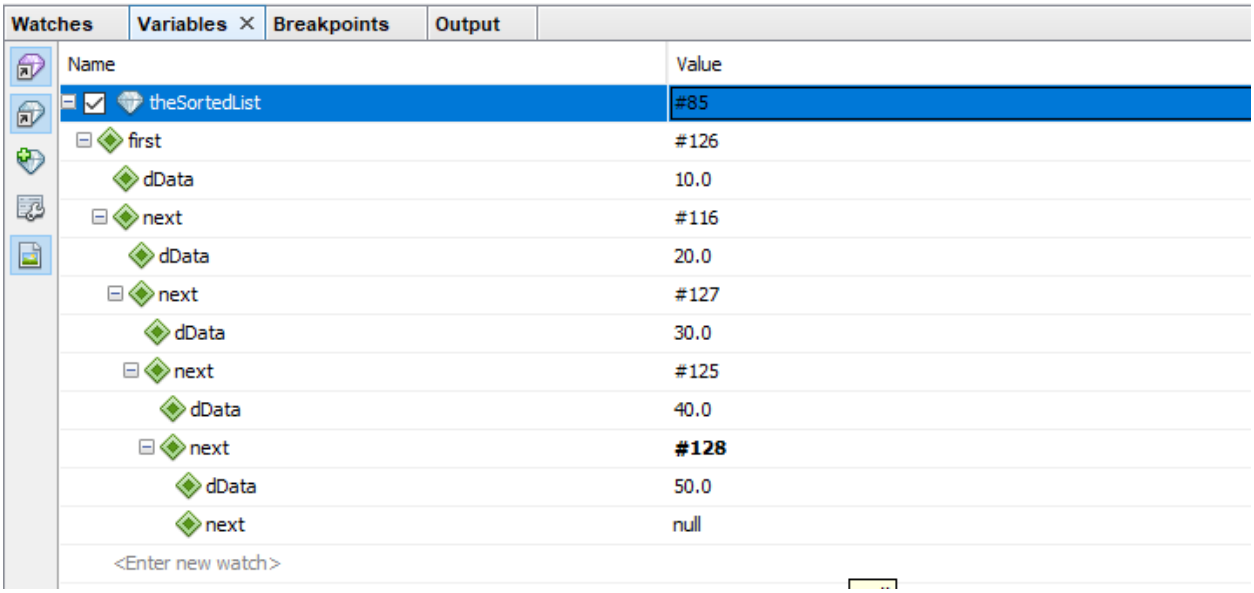


În partea din dreapta a ferestrei, în coloana *Value*, sunt afișate valorile variabilelor puse în watch, la momentul când programul s-a oprit.

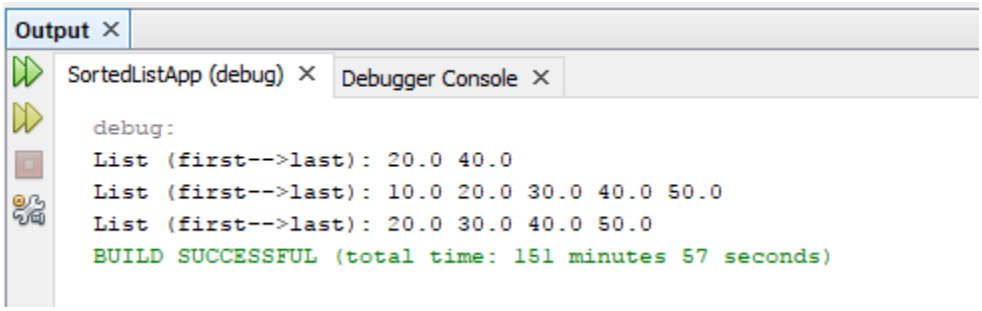
Pentru a sări la următorul breakpoint:



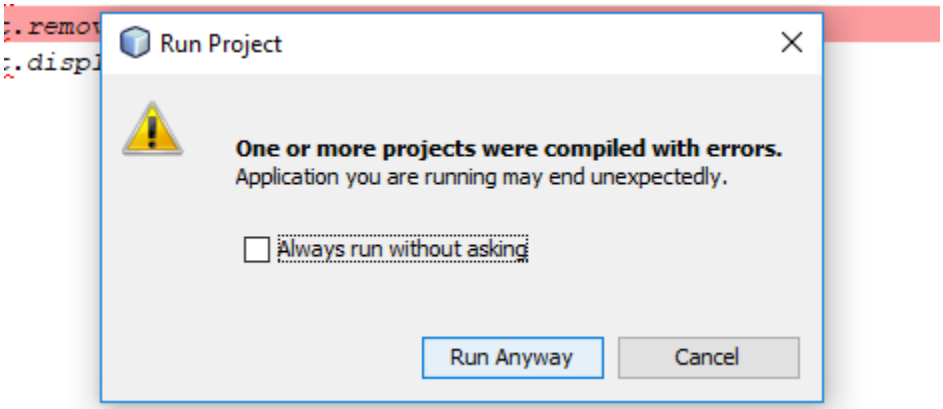
În fereastra de *Watch/Variables* putem urmări inserarea unui nou număr în lista, precum și sortarea:



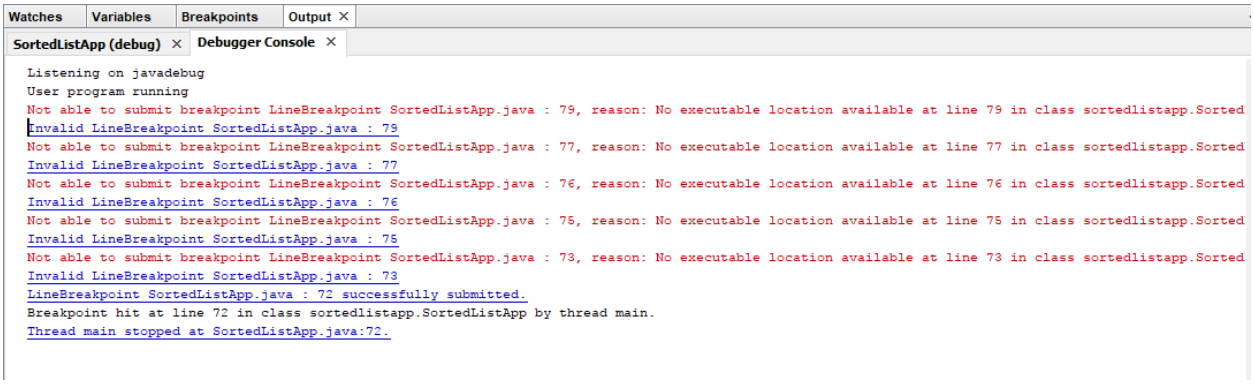
După terminarea execuției programului sau după executarea ultimului breakpoint, fereastra de debug se va inchide și va afișa în mod automat output-ul:



Dacă avem erori în program, următoarea fereastră va apărea înainte să putem intra în modul debug:



În funcție de eroare, apăsând butonul "*Run Anyway*", ne arată erorile în consola de debug:



Aplicație: Lucrul cu TableView (**JavaFX**). TableView este corespondetul JTable din Swing, care conține linii și coloane. Mai jos este prezentat un exemplu care prezintă cum se populeaza un TableView.

```
package javaFx;

import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;



public class TableViewSample extends Application {


    private final TableView<Person> table = new TableView<>();

    //populare lista
    private final ObservableList<Person> data =
            FXCollections.observableArrayList(
```

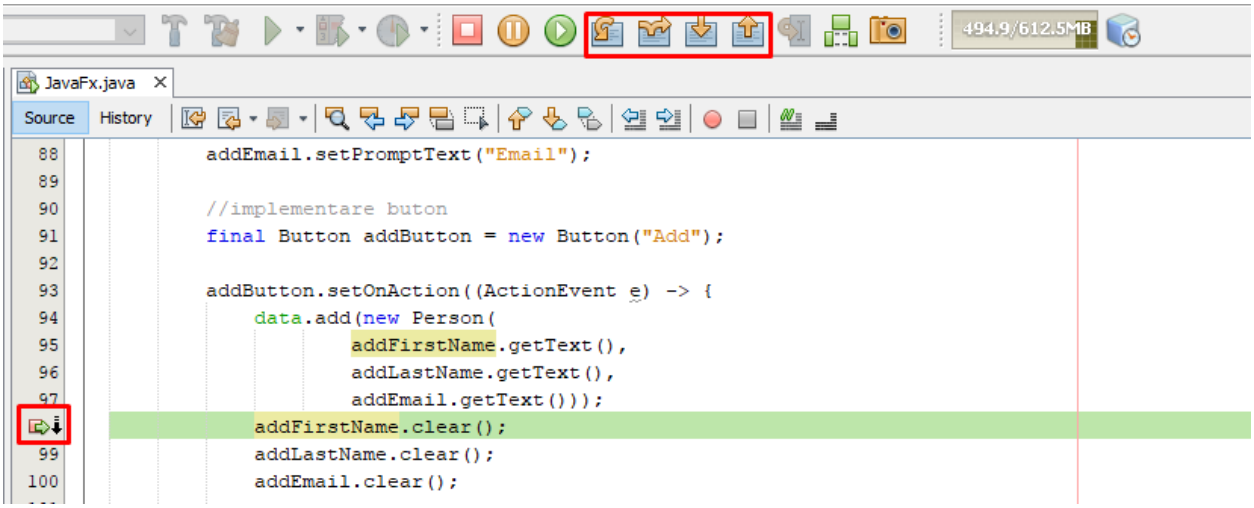
```
        new Person("Jacob", "Smith", "jacob.smith@example.com"),
        new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
        new Person("Ethan", "Williams", "ethan.williams@example.com"),
        new Person("Emma", "Jones", "emma.jones@example.com"),
        new Person("Michael", "Brown", "michael.brown@example.com"));

final HBox hb = new HBox();

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

@Override
public void start(Stage stage) {
    Scene scene = new Scene(new Group());
    stage.setTitle("Table View Sample");
    stage.setWidth(450);
    stage.setHeight(550);

    //adaugare eticheta
    final Label label = new Label("Address Book");
    label.setFont(new Font("Arial", 20));

    table.setEditable(true);

    //creare coloane
    TableColumn firstNameCol = new TableColumn("First Name");
    firstNameCol.setMinWidth(100);
    firstNameCol.setCellValueFactory(
            new PropertyValueFactory<>("firstName"));

    TableColumn lastNameCol = new TableColumn("Last Name");
    lastNameCol.setMinWidth(100);
    lastNameCol.setCellValueFactory(
            new PropertyValueFactory<>("lastName"));

    TableColumn emailCol = new TableColumn("Email");
    emailCol.setMinWidth(200);
    emailCol.setCellValueFactory(
            new PropertyValueFactory<>("email"));

    table.setItems(data);
    table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

    //creare texte
    final TextField addFirstName = new TextField();
    addFirstName.setPromptText("First Name");
    addFirstName.setMaxWidth(firstNameCol.getPrefWidth());
    final TextField addLastName = new TextField();
    addLastName.setMaxWidth(lastNameCol.getPrefWidth());
    addLastName.setPromptText("Last Name");
```

```
    final TextField addEmail = new TextField();
    addEmail.setMaxWidth(emailCol.getPrefWidth());
    addEmail.setPromptText("Email");

    //implementare buton
    final Button addButton = new Button("Add");

    addButton.setOnAction((ActionEvent e) -> {
        data.add(new Person(
                addFirstName.getText(),
                addLastName.getText(),
                addEmail.getText()));
        addFirstName.clear();
        addLastName.clear();
        addEmail.clear();
    });

    hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton);
    hb.setSpacing(3);

    final VBox vbox = new VBox();
    vbox.setSpacing(5);
    vbox.setPadding(new Insets(10, 0, 0, 10));
    vbox.getChildren().addAll(label, table, hb);

    ((Group) scene.getRoot()).getChildren().addAll(vbox);

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

//clasa imbricata pentru adaugarea de noi date
public static class Person {


    private final SimpleStringProperty firstName;
    private final SimpleStringProperty lastName;
    private final SimpleStringProperty email;

    private Person(String fName, String lName, String email) {
        this.firstName = new SimpleStringProperty(fName);
        this.lastName = new SimpleStringProperty(lName);
        this.email = new SimpleStringProperty(email);       }

    public String getFirstName() {
        return firstName.get();
    }

    public void setFirstName(String fName) {
        firstName.set(fName);
    }
```

```
    public String getLastName() {
        return lastName.get();
    }

    public void setLastName(String fName) {
        lastName.set(fName);
    }

    public String getEmail() {
        return email.get();
    }


    public void setEmail(String fName) {
        email.set(fName);
    }
}
}
```

Debug:

1. Adăugăm breakpoint-uri în codul sursă pentru a observa variabilele care ne interesează, sau pentru a observa evoluția unor funcții cu ajutorul butoanelor *Step In, Step Out* etc. (ex. În funcția de adăugare persoane în listă)



2. În fereastra *Variables* putem observa evoluția variabilelor odată cu apăsarea butonului: