**TABLE OF CONTENTS**

1. **INTRODUCTION** **2**
2. **PROGRAM DOCUMENTATION 3**
   1. *Getting Started 3*
   2. *Windows and Usage 6*
3. **CODE DOCUMENTATION 12**
   1. *FXML Files ----------------------------------------------------------- 13*
      1. window-view.fxml 13
      2. powerset-window.fxml 14
      3. complement-window.fxml 14
      4. learn-set.fxml 14
      5. learn-set-operations.fxml 15
      6. learn-set-relations.fxml 15
   2. *Controllers ----------------------------------------------------------- 16*
      1. Controller 16
      2. PowerSetController 26
      3. ComplementController 28
      4. LearnSetOperationsController 30
      5. LearnSetRelationsController 32
   3. *Set Class ------------------------------------------------------------- 35*
      1. Utility Methods 36
      2. Set Operation Methods 37
      3. Set Relation Methods 39

**INTRODUCTION**

We commonly use the terms like ‘a complete set of novels’ or ‘a set of cutlery’ in day-to-day life. What do we mean by the term ‘set’ here? It simply defines a collection of objects or things of the same type. Sets in math are also defined in the similar context.

In mathematics, a **set** is defined as a collection of distinct, well-defined objects forming a group. There can be any number of items, be it a collection of whole numbers, months of a year, types of birds, and so on. Each item in the set is known as an element of the set. We use curly brackets while writing a set.

Sets can be **related** to one another in a number of ways. They can be completely different, have some elements in common, be exactly the same, have the same number of elements, or one set can be part of another. Two sets are *equal* if they contain exactly the same elements, and they are *equivalent* if they have the same number of elements. A set is a *subset* of another if all its elements are in the other set, while *disjoint sets* have no elements in common. The *universal set* contains all possible elements in a context, the *empty set* has no elements, and the *power set* includes all possible subsets of a set.

**Set operations** can be defined as the operations that are performed on two or more sets to obtain a single set containing a combination of elements from all the sets being operated upon. The *complement* of a set includes all the elements not in the set but in the universal set. The *union* of two sets combines all the elements from both sets. The *intersection* of two sets contains only the elements that are common to both sets. The *difference* between two sets consists of the elements in one set that are not in the other set.

This project makes use of programming to make an application that utilizes the topics mentioned above. The main language we used is the **Java** programming language. We also implemented a GUI for better usability. The GUI type we used is **JavaFX**, with the help of a drag-and-drop application called **Scene Builder** to help simplify the building process of the GUI. For the logic of the program, we implemented a custom **Set** class that will handle most of the calculations for the Set Relations and Operations. Lastly, the project makes heavy use of the **MVC Design Pattern**. The *Model* being our custom Set class, the *View* being our GUI, and the not-mentioned-before *Controller* classes will be awaiting input from the user while interacting with the GUI.

**PROGRAM DOCUMENTATION**

from README.md file of <https://github.com/lirrnaiad/Set-Relations-and-Operations>

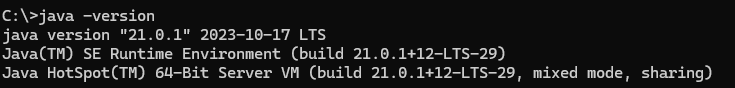
**Getting Started**

**Dependencies:**

* Windows 10/11
* atleast JDK 21
* atleast JavaFX SDK 17

Make sure that the command line's Java version is atleast JDK 21. You can check it using the command ***java -version***.

Here's what it looks like (if you downloaded the JDK from Oracle; if it's from OpenJDK it'll be different, the important thing is that it's version 21 of Java):



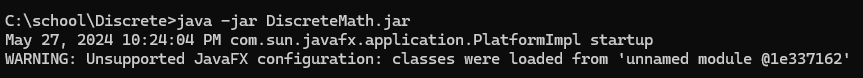
**Installing:**

* Download JDK 21 (or newer) here: https://www.oracle.com/java/technologies/downloads/#java21.
* Download JavaFX 17 SDK (or newer) here: <https://gluonhq.com/products/javafx/>.
* Download **DiscreteMath.jar** from <https://github.com/lirrnaiad/Set-Relations-and-Operations/releases>.

**Executing Program:**

* Make sure your command line is in the same repository using the ***cd*** command. For more info, check the *Help* section.
* Make sure your JDK is of version 21 or newer.
* Once everything has been set up, enter the command ***java -jar DiscreteMath.jar****.*

If the program has started successfully, it will show the following lines of code (disregard the warning):



**Help**

**Error: Unable to access jarfile DiscreteMath.jar**

Your command line is currently on the wrong directory. Make sure to move to the directory where the **DiscreteMath.jar** file is.

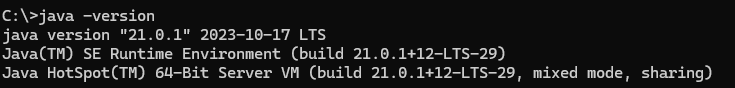
If the .jar file is in the Downloads folder, then it is located at **C:\Users\[username]\Downloads**, with **[username]** being the name of your user folder. For the example, I'll use **Sean** as the user folder.



Your command line will then show what repository it is currently, you can then run ***java -jar DiscreteMath.jar***.

**Error: LinkageError occurred while loading main class com.sets.discremathsets.Main**

The JDK version your command line is currently using is not of JDK 21 or newer.

Run ***java -version***. It should look something similar to this:

To fix this, make sure to download the appropriate JDK from the Installing section and install it. The **jdk-21 folder** (or similar)'s default location will be *at* ***C:\Users\Program Files\Java.***

After this, make sure to change the **PATH** to the jdk-21 folder (or similar). You can do this in two ways:

**Setting Temporary Path**

* Copy the path of jdk/bin directory where Java is located (***C:\Program Files\Java\jdk-21 \bin***)
* Write the command: ***SET PATH=C:\Program Files\Java\jdk-21\bin*** and hit enter.
* Run ***java -version*** again to check if it was successfully changed.

**Setting Permanent Path**

* Go to **This PC** and right click on it.
* Click on **Properties**.
* Under Device Specifications, there should be a related links section, click on **Advanced system settings**.
* Click on **Environment Variables**.
* Click on New tab of User variables, assign value ***JAVA\_HOME*** to Variable Name
* ***java\jdk-21\bin*** path (copied path) to Variable Value and click on OK Button
* Finally, click on OK button.

**Windows and their Usage**

**Main Window**

***A screenshot of a computer

Description automatically generated***A screenshot of a computer

Description automatically generated

1. Click “Get Set Relations” to get results
2. Enter elements of sets A and B

***A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated*Power Set Window**

1. Toggle between using notations or not by checking the "Toggle Notation” button.
2. Toggle between Set Relations and Set Operations usng the “Switch to Set Operations” or “Switch to Set Relations” Button

**A screenshot of a computer

Description automatically generated**

***A screenshot of a computer

Description automatically generated***

1. Input elements of the set and click on “Get Power Set” button to get all possible subsets
2. Open Power Set window by clicking on its button in the main window

**Complement Window *A screenshot of a computer

Description automatically generated***

1. Open Complement window by clicking on its button in the main window (switch to Set Operations first)

***A screenshot of a computer

Description automatically generated***

1. Input elements of the universal set and the set to perform operation on and click on “Get Complement” button to get the complement set.

**Learn Menu and Corresponding Menu Items**

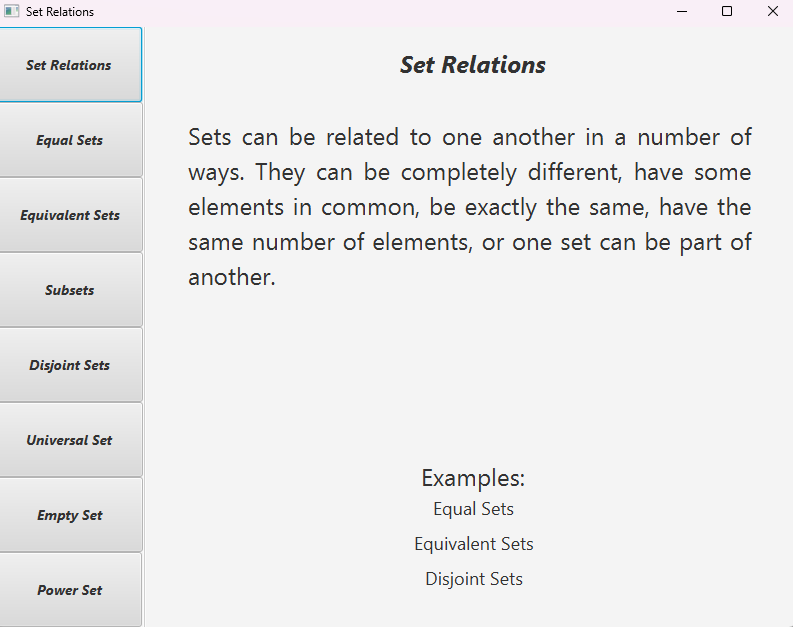
***A screenshot of a computer

Description automatically generatedScreens screenshot of a computer

Description automatically generated***

1. Open the Learn Menu to show different menu items

***A screenshot of a computer

Description automatically generated***

1. Navigate through the descriptions using the buttons on the left

**CODE DOCUMENTATION**

package com.sets.discremathsets;  
  
public class Main {  
 public static void main(String[] args) {  
 Application.*main*(args);  
 }  
}

The **Main** class will be what our package runs first when opening the .jar file using the Terminal. It calls the main() method of the **Application** class, shown below:

package com.sets.discremathsets;  
  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
  
public class Application extends javafx.application.Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Application.class.getResource("window-view.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 800, 650);  
 stage.setTitle("Set Relations and Operations");  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

This class is where the initial window shows up when running the program. It loads the **window-view.fxml** file from the resources folder and creates a new window with a resolution of 800x650 with a title of “Set Relations and Operations”.

**FXML Files**

The .fxml files are what makes the design and layout of the windows. Using Scene Builder, these .fxml files are automatically generated and are very lengthy (up to 300 lines of code) due to the syntax of XML files itself. Because of this, the screenshots of the windows in Scene Builder will be shown instead.

A screenshot of a computer

Description automatically generated

window-view.fxml

This window is what will open after launching the program. The window is contained in a **BorderPane** and inside it are a **MenuBar** at the top which contains the Learn menu item which then contains the windows for the different set relation and operation descriptions.

At the bottom (the rest of the window) is a **GridPane** which stores the different functionalities of the window:

The top left portion are the **TextField**s for the input of the sets, on the bottom left are the button to get the results, a label to notify the user to use commas to separate the elements, and the toggle to use notation form for the results. At the top right are labels which confirm the set you just inputted after pressing the results button, at the center right are the different results which change from Set Relations to Set Operations depending on the button on the bottom right, which is to toggle between Set Operations and Relations. The bottom right portion of the GridPane also contains the functionalities for Power Set and Complement, depending on the current toggle.

A screenshot of a computer

Description automatically generatedThis window will run when the “Power Set” button in the main window is pressed. It contains a **TextField** for inputting the main set and a **Button** to get the results. The results label is particularly large since the results of a power set can get pretty numerous since it exponentially grows.

powerset-window.fxml

A screenshot of a computer

Description automatically generated

This window will run when the “Complement” button in the main window is pressed. It contains two **TextField**s for inputting the universal set and the set to operate on. Once the sets have been inputted, clicking on the button will display the results for the complement of A.

complement-window.fxml

A screenshot of a computer

Description automatically generated

This window will run when the “What is a set?” **MenuItem** is clicked from the Learn **MenuBar**. It contains a description of what a set is and some examples on how to write one.

learn-set.fxml

A screenshot of a computer

Description automatically generated

learn-set-operations-window.fxml

This window wil run when the “Set Operations” **MenuItem** is clicked from the Learn **MenuBar**. It contains a description of set operations and the different descriptions of the said set operations with examples. You can traverse through the different descriptions using the buttons found in the left side of the window.

A screenshot of a computer

Description automatically generated

learn-set-relations-window.fxml

Likewise, this window wil run when the “Set Relations” **MenuItem** is clicked from the Learn **MenuBar**. It contains a description of set relations and the different descriptions of the said set operations with examples. Like the previous window, you can traverse the different descriptions using the buttons found in the left side of the window.

**Controllers**

The Controller classes are what connects the GUI to the logic of the program. These classes are where **Event Listeners** are coded, adding different functionalities to different interactible elements in the program.

**Controller** (main controller class)

package com.sets.discremathsets.logic;  
  
import com.sets.discremathsets.Application;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
  
import com.sets.discremathsets.domain.Set;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.HashSet;  
  
public class Controller {  
 @FXML  
 public TextField setAField;  
 public TextField setBField;  
 public Button resultsButton;  
 public Button toggleButton;  
 public Label setAOutput;  
 public Label setBOutput;  
 public Label isEqualOrUnionLabel;  
 public Label isEquivalentOrUnionResultLabel;  
 public Label isASubsetOfBOrIntersectionLabel;  
 public Label isBSubsetOfAOrIntersectionResultLabel;  
 public Label isDisjointOrDifferenceAMinusBLabel;  
 public Label DifferenceAMinusBResultLabel;  
 public Label DifferenceBMinusALabel;  
 public Label DifferenceBMinusAResultLabel;  
 public CheckBox notationToggle;  
 public Button powerSetOrComplementButton;  
 public MenuItem learnSet;  
 public MenuItem learnSetRelations;  
 public MenuItem learnSetOperations;

This is the controller for the initial window. First, we initialize all the interactible elements into a variable so that we can call them in our methods.

// Instance variable to check whether current mode is set relations or not. True by default  
private boolean modeIsSetRelations = true;  
  
// Instance variables to check the set relations of the two current sets. False by default.  
private boolean notationToggled = false;  
private boolean setsAreEqual = false;  
private boolean setsAreEquivalent = false;  
private boolean setAIsSubsetOfB = false;  
private boolean setBIsSubsetOfA = false;  
private boolean setsAreDisjoint = false;

These are extra instance variables to check the current state of the program. I added these to avoid undefined behavior and so that results have default values.

// Get set from the text field  
private HashSet<String> getSetFromTextField(TextField setField) {  
 if (setField.getText().isEmpty()) {  
 return new HashSet<>();  
 }  
  
 String[] elements = setField.getText().split(",");  
 HashSet<String> set = new HashSet<>();  
  
 // Remove any leading or trailing whitespaces from the elements  
 for (String element : elements) {  
 set.add(element.trim());  
 }  
  
 return set;  
}

This is the method used to get the text from the **TextField**s and convert them into a HashSet of type String to be used in making a new Set object.

// Runs when "Get Set Relations" or "Get Set Operations" button is clicked  
@FXML  
protected void onResultsButtonClick() {  
 // Create sets from the text field  
 Set setA = new Set(getSetFromTextField(setAField));  
 Set setB = new Set(getSetFromTextField(setBField));  
  
 // Output text from text field  
 if (setA.isEmpty()) {  
 setAOutput.setText("∅");  
 } else {  
 setAOutput.setText(String.*valueOf*(setA));  
 }  
  
 if (setB.isEmpty()) {  
 setBOutput.setText("∅");  
 } else {  
 setBOutput.setText(String.*valueOf*(setB));  
 }  
  
 if (modeIsSetRelations) {  
 // SET RELATIONS  
 setsAreEqual = setA.isEqualTo(setB);  
 setsAreEquivalent = setA.isEquivalentTo(setB);  
 setAIsSubsetOfB = setA.isSubsetOf(setB);  
 setBIsSubsetOfA = setB.isSubsetOf(setA);  
 setsAreDisjoint = setA.isDisjoint(setB);  
 updateSetRelations();  
 } else {  
 // SET OPERATIONS  
 updateSetOperations();  
 Set setUnion = new Set(setA.union(setB));  
 Set setIntersection = new Set(setA.intersection(setB));  
 Set setDifferenceAMinusB = new Set(setA.difference(setB));  
 Set setDifferenceBMinusA = new Set(setB.difference(setA));  
  
 if (setUnion.isEmpty()) {  
 isEquivalentOrUnionResultLabel.setText("= ∅");  
 } else {  
 isEquivalentOrUnionResultLabel.setText("= " + setUnion);  
 }  
  
 if (setIntersection.isEmpty()) {  
 isBSubsetOfAOrIntersectionResultLabel.setText("= ∅");  
 } else {  
 isBSubsetOfAOrIntersectionResultLabel.setText("= " + setIntersection);  
 }  
  
 if (setDifferenceAMinusB.isEmpty()) {  
 DifferenceAMinusBResultLabel.setText("= ∅");  
 } else {  
 DifferenceAMinusBResultLabel.setText("= " + setDifferenceAMinusB);  
 }  
  
 if (setDifferenceBMinusA.isEmpty()) {  
 DifferenceBMinusAResultLabel.setText("= ∅");  
 } else {  
 DifferenceBMinusAResultLabel.setText("= " + setDifferenceBMinusA);  
 }  
 }  
}

This method is activated when the “Get Set Relations” or “Get Set Operations” button is clicked. How it works is:

1. It calls the getSetFromTextField() method and creates new Set objects setA and setB.
2. It outputs the inputted sets into the labels on the right of the text fields to confirm what the user has inputted.
3. Checks whether the current mode is Set Relations or Set Operations and updates the text of the diffrent result labels accordingly.

// Toggles between Set Relations and Set Operations  
@FXML  
protected void onToggleButtonClick() {  
 if (modeIsSetRelations) {  
 modeIsSetRelations = false;  
 resultsButton.setText("Get Set Operations");  
 toggleButton.setText("Switch to Set Relations");  
 powerSetOrComplementButton.setText("Complement");  
  
 onResultsButtonClick();  
  
 DifferenceAMinusBResultLabel.setVisible(true);  
 DifferenceBMinusALabel.setVisible(true);  
 DifferenceBMinusAResultLabel.setVisible(true);  
  
 } else {  
 modeIsSetRelations = true;  
 resultsButton.setText("Get Set Relations");  
 toggleButton.setText("Switch to Set Operations");  
 powerSetOrComplementButton.setText("Power Set");  
  
 onResultsButtonClick();  
  
 DifferenceAMinusBResultLabel.setVisible(false);  
 DifferenceBMinusALabel.setVisible(false);  
 DifferenceBMinusAResultLabel.setVisible(false);  
 }  
}

This method activates when the **“Switch to Set Operations”** or **“Switch to Set Relations”** button is clicked. How it works is:

1. Changes the instance variable modelsSetRelations, indicating that the current mode is no longer Set Relations but rather Set Operations. The same happens in reverse when Set Operations is the current mode when clicked.
2. Makes extra labels visible or disappear depending on the current mode.
3. This method calls the same method when clicking the results button, making it so that the results are automatically calculated when the mode is changed.

// Toggle between words and notation  
@FXML  
protected void onNotationToggle() {  
 notationToggled = notationToggle.isSelected();  
  
 if (modeIsSetRelations) {  
 updateSetRelations();  
 } else {  
 updateSetOperations();  
 }  
}

This method activates when the checkbox “Toggle Notation” is checked. It updates the notationToggled variable to true or false depending if the checkbox is checked or not. It then calls the helper methods updateSetRelations() or updateSetOperations() depending on the current mode.

// Updates labels of set operations depending on whether notation is toggled  
private void updateSetOperations() {  
 if (!notationToggled) {  
 isEqualOrUnionLabel.setText("A Union B");  
 isASubsetOfBOrIntersectionLabel.setText("A Intersection B");  
 isDisjointOrDifferenceAMinusBLabel.setText("A Minus B");  
 DifferenceBMinusALabel.setText("B Minus A");  
 } else {  
 isEqualOrUnionLabel.setText("A ∪ B");  
 isASubsetOfBOrIntersectionLabel.setText("A ∩ B");  
 isDisjointOrDifferenceAMinusBLabel.setText("A - B");  
 DifferenceBMinusALabel.setText("B - A");  
 }  
}

This method changes the labels for the results on Set Operations mode and changes them to notation form.

(PS. It looks broken in MS Word but it is shown just fine in the program itself.)

// Updates labels of set relations depending on whether notation is toggled  
private void updateSetRelations() {  
 if (!notationToggled) {  
 if (setsAreEqual) {  
 isEqualOrUnionLabel.setText("Equal Sets: ✔");  
 } else {  
 isEqualOrUnionLabel.setText("Equal Sets: ✘");  
 }  
  
 if (setsAreEquivalent) {  
 isEquivalentOrUnionResultLabel.setText("Equivalent Sets: ✔");  
 } else {  
 isEquivalentOrUnionResultLabel.setText("Equivalent Sets: ✘");  
 }  
  
 if (setAIsSubsetOfB) {  
 isASubsetOfBOrIntersectionLabel.setText("A is a subset of B: ✔");  
 } else {  
 isASubsetOfBOrIntersectionLabel.setText("A is a subset of B: ✘");  
 }  
  
 if (setBIsSubsetOfA) {  
 isBSubsetOfAOrIntersectionResultLabel.setText("B is a subset of A: ✔");  
 } else {  
 isBSubsetOfAOrIntersectionResultLabel.setText("B is a subset of A: ✘");  
 }  
  
 if (setsAreDisjoint) {  
 isDisjointOrDifferenceAMinusBLabel.setText("Disjoint Sets: ✔");  
 } else {  
 isDisjointOrDifferenceAMinusBLabel.setText("Disjoint Sets: ✘");  
 }  
 } else {  
 if (setsAreEqual) {  
 isEqualOrUnionLabel.setText("A = B");  
 } else {  
 isEqualOrUnionLabel.setText("A ≠ B");  
 }  
  
 if (setsAreEquivalent) {  
 isEquivalentOrUnionResultLabel.setText("A ≈ B");  
 } else {  
 isEquivalentOrUnionResultLabel.setText("A ≉ B");  
 }  
  
 if (setAIsSubsetOfB) {  
 isASubsetOfBOrIntersectionLabel.setText("A ⊂ B");  
 } else {  
 isASubsetOfBOrIntersectionLabel.setText("A ⊄ B");  
 }  
  
 if (setBIsSubsetOfA) {  
 isBSubsetOfAOrIntersectionResultLabel.setText("B ⊂ A");  
 } else {  
 isBSubsetOfAOrIntersectionResultLabel.setText("B ⊄ A");  
 }  
  
 if (setsAreDisjoint) {  
 isDisjointOrDifferenceAMinusBLabel.setText("A ∩ B = ∅");  
 } else {  
 isDisjointOrDifferenceAMinusBLabel.setText("A ∩ B ≠ ∅");  
 }  
 }  
}

Likewise, this method changes the labels for the results on Set Relations mode and changes them to notation form.

(PS. It looks broken in MS Word but it is shown just fine in the program itself.)

// Power Set or Complement Windows  
@FXML  
protected void calculatePowerSetOrComplement() {  
 if (powerSetOrComplementButton.getText().equals("Power Set")) {  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader(Application.class.getResource("powerset-window.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 600, 400);  
 Stage stage = new Stage();  
 stage.setTitle("Power Set");  
 stage.setScene(scene);  
 stage.show();  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*err*.println("Failed to open new window: " + e.getMessage());  
 }  
 } else {  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader(Application.class.getResource("complement-window.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 600, 400);  
 Stage stage = new Stage();  
 stage.setTitle("Complement");  
 stage.setScene(scene);  
 stage.show();  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*err*.println("Failed to open new window: " + e.getMessage());  
 }  
 }  
}

This method activates when the **“Power Set”** or **“Complement”** button is clicked, depending on the current mode. It opens a new window and calls their respective .fxml files.

// Menu Items  
@FXML  
protected void onLearnSetClick() {  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader(Application.class.getResource("learn-set.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 600, 400);  
 Stage stage = new Stage();  
 stage.setTitle("Set Definition");  
 stage.setScene(scene);  
 stage.show();  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*err*.println("Failed to open new window: " + e.getMessage());  
 }  
}  
  
@FXML  
protected void onLearnSetRelationsClick() {  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader(Application.class.getResource("learn-set-relations.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 800, 600);  
 Stage stage = new Stage();  
 stage.setTitle("Set Relations");  
 stage.setScene(scene);  
 stage.show();  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*err*.println("Failed to open new window: " + e.getMessage());  
 }  
}  
  
@FXML  
protected void onLearnSetOperationsClick() {  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader(Application.class.getResource("learn-set-operations.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 800, 600);  
 Stage stage = new Stage();  
 stage.setTitle("Set Operations");  
 stage.setScene(scene);  
 stage.show();  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*err*.println("Failed to open new window: " + e.getMessage());  
 }  
}

These are the different methods for when the different **MenuItems** are clicked in the Learn **MenuBar**. These will open their respective .fxml files.

**PowerSetController** (Controller for powerset-window.fxml)

package com.sets.discremathsets.logic;  
  
import com.sets.discremathsets.domain.Set;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.text.Font;  
  
import java.util.HashSet;  
  
public class PowerSetController {  
 public TextField powerSetTextField;  
 public Button resultsButton;  
 public Label resultsLabel;  
 public Label subsetNumberLabel;  
  
 private HashSet<String> getSetFromTextField(TextField setField) {  
 if (setField.getText().isEmpty()) {  
 return new HashSet<>();  
 }  
  
 String[] elements = setField.getText().split(",");  
 HashSet<String> set = new HashSet<>();  
  
 // Remove any leading or trailing whitespaces from the elements  
 for (String element : elements) {  
 set.add(element.trim());  
 }  
  
 return set;  
 }

Once again, initializing the interactible elements used in the power set window, and reusing the getSetFromTextField() method for getting the text inside the TextField.

@FXML  
protected void onResultsButtonClick() {  
 Set setA = new Set(getSetFromTextField(powerSetTextField));  
 int numberOfSubsets = (int) Math.*pow*(2, setA.size());  
 subsetNumberLabel.setText("Total subsets: " + numberOfSubsets);  
  
 // Change font size depending on number of subsets  
 if (numberOfSubsets <= 32) {  
 resultsLabel.setFont(new Font(20.0));  
 } else if (numberOfSubsets <= 128) {  
 resultsLabel.setFont(new Font(12.0));  
 } else if (numberOfSubsets <= 256) {  
 resultsLabel.setFont(new Font(10.0));  
 } else {  
 resultsLabel.setFont(new Font(5.0));  
 }  
  
 resultsLabel.setText(String.*valueOf*(setA.powerSet()));  
}

This method activates when the results button is clicked. How it works is:

1. Creates a new Set object from the TextField using the getSetFromTextField() method.
2. Counts the number of subsets using the 2^n formula and updates this into a Label.
3. It then changes the font size of the results Label first using the number of subsets as a baseline. The more the subsets, the smaller the font.
4. Calls the powerset() method for the created Set object and prints it into the result Label.

**ComplementController** (Controller for complement-window.fxml)

package com.sets.discremathsets.logic;  
  
import com.sets.discremathsets.domain.Set;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
  
import java.util.HashSet;  
  
public class ComplementController {  
 public TextField universalSetTextField;  
 public TextField setATextField;  
 public Button getComplementButton;  
 public Label complementLabel;  
  
 private HashSet<String> getSetFromTextField(TextField setField) {  
 if (setField.getText().isEmpty()) {  
 return new HashSet<>();  
 }  
  
 String[] elements = setField.getText().split(",");  
 HashSet<String> set = new HashSet<>();  
  
 // Remove any leading or trailing whitespaces from the elements  
 for (String element : elements) {  
 set.add(element.trim());  
 }  
  
 return set;  
 }

Once again, initializing the interactible elements used in the complement window, and reusing the getSetFromTextField() method for getting the text inside the TextFields.

@FXML  
protected void onComplementButtonClicked() {  
 Set universalSet = new Set(getSetFromTextField(universalSetTextField));  
 Set toBeComplemented = new Set(getSetFromTextField(setATextField));  
  
 complementLabel.setText(String.*valueOf*(new Set(toBeComplemented.complement(universalSet))));  
}

This method activates when the results button is clicked. How it works is:

1. Gets the text from both text fields and creates them into Set objects.
2. The complement() method is then used for the two sets.
3. A new Set object is created using the results from the complement method().
4. The results label is then updated using the String.valueOf() method of the created Set object.

**LearnSetOperationsController** (Controller for learn-set-operations-window.fxml)

package com.sets.discremathsets.logic;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
  
public class LearnSetOperationsController {  
 public Label titleLabel;  
 public Label notationLabel;  
 public Label descriptionLabel;  
 public Label exampleOneLabel;  
 public Label exampleTwoLabel;  
 public Label exampleThreeLabel;

Initializition of variables.

@FXML  
protected void onSetOperationsClick() {  
 titleLabel.setText("Set Operations");  
 notationLabel.setOpacity(0);  
 descriptionLabel.setText("Set operations can be defined as the operations that are performed on two or more sets to obtain a single set containing a combination of elements from both all the sets being operated upon.");  
 exampleOneLabel.setText("Union of Sets");  
 exampleTwoLabel.setText("Intersection of Sets");  
 exampleThreeLabel.setText("Difference of Sets");  
}  
  
@FXML  
protected void onComplementClick() {  
 titleLabel.setText("Set Complement");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A'");  
 descriptionLabel.setText("The complement of set A is the set of elements in the universal set which are not in set A. That is, the set of all elements in the universal set outside of set A.");  
 exampleOneLabel.setText("U = {1, 2, 3, 4, 5}");  
 exampleTwoLabel.setText("A = {1, 3, 5}");  
 exampleThreeLabel.setText("A' = {2, 4}");  
}  
  
  
@FXML  
protected void onUnionClick() {  
 titleLabel.setText("Set Union");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A ∪ B");  
 descriptionLabel.setText("The union of the sets A and B is the set of elements that belong to A or to B or both, that is, it is the set made up combining all the elements of set A with all the elements of set B.");  
 exampleOneLabel.setText("A = {1, 2, 3}");  
 exampleTwoLabel.setText("B = {3, 4, 5}");  
 exampleThreeLabel.setText("A ∪ B = {1, 2, 3, 4, 5}");  
}  
  
@FXML  
protected void onIntersectionClick() {  
 titleLabel.setText("Set Intersection");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A ∩ B");  
 descriptionLabel.setText("The intersection of the sets A and B is the set of all elements which belong to both A and B, that is, the set made up of the elements common to A and B.");  
 exampleOneLabel.setText("A = {1, 2, 3, 4, 5}");  
 exampleTwoLabel.setText("B = {4, 5, 6, 7, 8}");  
 exampleThreeLabel.setText("A ∩ B = {4, 5}");  
}  
  
@FXML  
protected void onDifferenceClick() {  
 titleLabel.setText("Set Difference");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A - B");  
 descriptionLabel.setText("The difference of set A minus set B is the set of elements which belong to A but which do not belong to B, that is, the set made up of the elements of A except those that can be found in B as well.");  
 exampleOneLabel.setText("A = {1, 2, 3, 4, 5}");  
 exampleTwoLabel.setText("B = {2, 4}");  
 exampleThreeLabel.setText("A - B = {1, 3, 5}");  
}

These methods will activate when their respective buttons are clicked. These will change the text displayed in the labels.

**LearnSetRelationsController** (Controller for learn-set-relations-window.fxml)

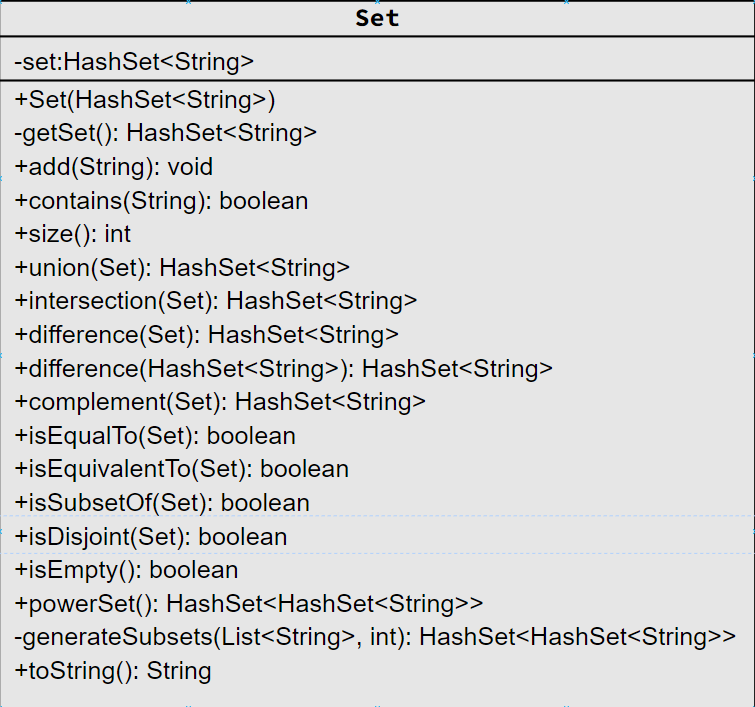
package com.sets.discremathsets.logic;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
  
public class LearnSetRelationsController {  
 public Label titleLabel;  
 public Label notationLabel;  
 public Label descriptionLabel;  
 public Label exampleOneLabel;  
 public Label exampleTwoLabel;  
 public Label exampleThreeLabel;

Initialization of variables

@FXML  
protected void onSetRelationsClick() {  
 titleLabel.setText("Set Relations");  
 notationLabel.setOpacity(0);  
 descriptionLabel.setText("Sets can be related to one another in a number of ways. They can be completely different, have some elements in common, be exactly the same, have the same number of elements, or one set can be part of another.");  
 exampleOneLabel.setText("Equal Sets");  
 exampleTwoLabel.setText("Equivalent Sets");  
 exampleThreeLabel.setText("Disjoint Sets");  
}  
  
@FXML  
protected void onEqualSetsClick() {  
 titleLabel.setText("Equal Sets");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A = B");  
 descriptionLabel.setText("Two sets A and B are equal (A = B) if and only if every element of set A is an element of set B is an element of set A. That is, two sets have exactly the same elements. If two sets A and B are not equal, we write A ≠ B.");  
 exampleOneLabel.setText("{1, 3, 5, 7} = {1, 3, 5, 7}");  
 exampleTwoLabel.setText("{2, 2, 4, 6, 8} = {2, 4, 4, 6, 8}");  
 exampleThreeLabel.setText("{1, 5, 7} ≠ {1, 3, 7}");  
}  
  
@FXML  
protected void onEquivalentSetsClick() {  
 titleLabel.setText("Equivalent Sets");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A ≈ B");  
 descriptionLabel.setText("Two sets A and B are equivalent (A ≈ B) if and only if for each element in A there is exactly one element in B and for each element in B there is exactly one element in A. If two sets A and B are not equivalent, we write A ≉ B.");  
 exampleOneLabel.setText("{2, 9, 4, 6} ≈ {3, 7, 1, 5}");  
 exampleTwoLabel.setText("{1, 2, 3, 4, 5} ≈ {6, 7, 8, 9, 10}");  
 exampleThreeLabel.setText("{1, 5, 7} ≉ {1, 3, 7, 4}");  
}  
  
@FXML  
protected void onSubsetsClick() {  
 titleLabel.setText("Subsets");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A ⊂ B");  
 descriptionLabel.setText("Set A is a subset of B if and only if each element of set A is an element of set B. That is, if x ∈ A, then x ∈ B. To indicate that A is a subset of B, we write A ⊂ B. This is read as 'A is a subset of B'. The negation of this idea, or 'A is not a subset of B' is written as A ⊄ B. From this definition, we see that every set is a subset of itself, that is for any set A, we have A ⊂ A.");  
 exampleOneLabel.setText("{1, 2, 4} ⊂ {1, 2, 3, 4, 5}");  
 exampleTwoLabel.setText("{a, b, c} ⊂ {a, b, c}");  
 exampleThreeLabel.setText("{1, 3, 5} ⊄ {1, 2, 3, 4}");  
}  
  
@FXML  
protected void onDisjointSetsClick() {  
 titleLabel.setText("Disjoint Sets");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("A ∩ B = ∅");  
 descriptionLabel.setText("Two sets A and B are disjoint if and only if no element of set A is an element of set B and no element of set B is an element of set A; that is, sets A and B have no elements in common.");  
 exampleOneLabel.setText("{1, 3, 5} and {2, 4, 6} are disjoint.");  
 exampleTwoLabel.setText("{a, b, c} and {A, B, C} are disjoint.");  
 exampleThreeLabel.setText("{1, 2, 4} and {2, 4, 5} are not disjoint.");  
}  
  
@FXML  
protected void onUniversalSetClick() {  
 titleLabel.setText("Universal Set");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("U");  
 descriptionLabel.setText("The universal set, denoted by U is the set containing all the elements under discussion. All the sets under consideration are subsets of the universal set U.");  
 exampleOneLabel.setText("U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}");  
 exampleTwoLabel.setText("U = {a, b, c, ..., z}");  
 exampleThreeLabel.setText("U = {x|x is a student in UEP}");  
}  
  
@FXML  
protected void onEmptySetClick() {  
 titleLabel.setText("Empty Set");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("{} or ∅");  
 descriptionLabel.setText("The null or empty set denoted by ∅ or {} is the set containing no elements.");  
 exampleOneLabel.setText("{}");  
 exampleTwoLabel.setText("∅");  
 exampleThreeLabel.setText("");  
}  
  
@FXML  
protected void onPowerSetClick() {  
 titleLabel.setText("Power Set");  
 notationLabel.setOpacity(1);  
 notationLabel.setText("P(A)");  
 descriptionLabel.setText("By the power set of a given set A, we mean the set of all possible subsets of Set A. If set A contain n number of elements, then 2^n number of subsets of A.");  
 exampleOneLabel.setText("A = {a, b, c}");  
 exampleTwoLabel.setText("P(A) =");  
 exampleThreeLabel.setText("{{}, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}");  
}

These methods will activate when their respective buttons are clicked. These will change the text displayed in the labels.

**Set Class**



UML Class Diagram of the Set Class

The **Set** class is the backbone of this program. This class is used for calculating the different set relations and operations for the sets provided by the user in the GUI. We used a **HashSet** of type **String** as the instance variable for Set objects as this provides us with built-in methods that make this implementation easier.

**Constructor**

public Set(HashSet<String> set) {  
 this.set = new HashSet<>(set);  
}

To create a Set object, we pass in a HashSet of type String as this will be what will be modified and updated while using the different methods in this class.

**Utility Methods**

private HashSet<String> getSet() {  
 return this.set;  
}

The getSet() method returns the HashSet that is inside the Set class. This method is used because the different Set Relation and Operation methods take in a Set object as a parameter, not a HashSet.

public void add(String element) {  
 this.set.add(element);  
}

Unused. Adds an element of type String by the given parameter to the HashSet of the current Set object. Not used because the GUI makes use of TextFields.

public boolean contains(String element) {  
 return this.set.contains(element);  
}

Unused. Checks whether the HashSet of a Set object contains the String of the given parameter.

public int size() {  
 return set.size();  
}

Returns the number of elements present in the HashSet of the Set object.

**Set Operation Methods**

public HashSet<String> union(Set setTwo) {  
 HashSet<String> setUnion = new HashSet<>(this.set);  
 setUnion.addAll(setTwo.getSet());  
 return setUnion;  
}

**Method Identifier**: union

**Method Declaration**: public HashSet<String> union(Set setTwo)

**Parameter:** Set setTwo

**Return Type**: The HashSet<String> returned by the union method will contain the elements that are present both in the original (this.set) and the setTwo, and are stored as strings.

**Method Description**: The method takes a setTwo as a parameter and calculates the union of two sets. It adds all the elements from setTwo.getSet() to setUnion.

public HashSet<String> intersection(Set setTwo) {  
 HashSet<String> setIntersection = new HashSet<>(this.set);  
 setIntersection.retainAll(setTwo.getSet());  
 return setIntersection;  
}

**Method Identifier**: intersection

**Method Declaration**: public Hashset<String> intersection(Set setTwo)

**Parameter**: Set setTwo

**Return type**: The HashSet<String> returned by the intersection method will contain the elements that are common to both the original set(this.set) and the setTwo set, and are stored as strings.

**Method Description**: This method takes a setTwo as a parameter and calculates the intersection of the two sets. It retains only the elements that are common to both setIntersection and setTwo.getSet() using the retain all method. Finally, it returns the setIntersection which contains the intersection of the two sets.

public HashSet<String> difference(Set setTwo) {  
 HashSet<String> setDifference = new HashSet<>(this.set);  
 setDifference.removeAll(setTwo.getSet());  
 return setDifference;  
}

**Method Identifier**: difference

**Method Declaration**: public HashSet<String> difference(Set setTwo)

**Parameter**: Set setTwo

**Return Type**: The HashSet returned by the difference method will contain the elements that are present in the original set (this.set) but not in the setTwo set. The elements are stored as strings.

**Method Description**: This method takes a setTwo as a parameter and calculates the difference between two sets. It removes all the elements that are present in setTwo.getSet() from setDifference using the removeAll method. Finally, it returns the setDifference which contains the elements that are in this.set but not in setTwo.

public HashSet<String> complement(Set universalSet) {  
 return universalSet.difference(this.set);  
}

**Method Identifier**: complement

**Method Declaration**: public HashSet<String> complement(Set universalSet)

**Parameter**: Set universalSet

**Return Type**: The HashSet<String> returned by the complement method will contain the elements that are present in the universalSet but not in the set this.set. The elements are stored as string.

**Method Description**: This method takes a universalSet as a parameter and calculates the complement set by finding the difference between the universalSet and the set (this.set) on which the method is called.

**Set Relation Methods**

public boolean isEqualTo(Set secondSet) {  
 return this.set.equals(secondSet.getSet());  
}

**Method Identifier**: isEqualTo

**Method Declaration**: public Boolean isEqualTo(Set secondSet)

**Parameter**: Set secondSet

**Return Type**: Boolean- when the isEqualTo method is called, it will return a Boolean value indicating whether the two sets are equal or not.

**Method Description**: This method takes a secondSet as a parameter and checks if it is equal to the set (this.set) on which the method is called. It returns true if the two sets are equal, and false otherwise.

public boolean isEquivalentTo(Set secondSet) {  
 return this.set.size() == secondSet.getSet().size();  
}

**Method Identifier**: isEquivalentTo

**Method Declaration**: public Boolean isEquivalentTo(Set secondSet)

**Parameter**: Set secondSet

**Return Type**: Boolean- when the isEqualTo method is called, it will return a Boolean value indicating whether the two sets are equivalent or not.

**Method Description**: This method takes a secondSet as a parameter and checks if it is equivalent (the two sets have the same number of elements) to the set (this.set) on which the method is called. It returns true if the two sets are equivalent, and false otherwise.

public boolean isSubsetOf(Set secondSet) {  
 if (this.set.size() > secondSet.getSet().size()) {  
 return false;  
 }  
  
 return secondSet.getSet().containsAll(this.set);  
}

**Method Identifier**: isSubsetOf

**Method Declaration**: public Boolean isSubsetOf(Set secondSet)

**Parameter**: Set secondSet

**Return Type**: Boolean- when the isSubsetOf method is called, it will return a boolen value indicating whether (this.set) is a subset of secondSet or not.

**Method Description**: This method takes a secondSet as a parameter and checks if the set (this.set) on which the method is called is a subset of secondSet. It returns true if (this.set) is a subset of secondSet, and false otherwise.

public boolean isDisjoint(Set secondSet) {  
 return intersection(secondSet).isEmpty();  
}

**Method Identifier**: isDisjoint

**Method Declaration**: public Boolean isDisjoint(Set secondSet)

**Parameter**: Set secondSet

**Return Type**: Boolean - true if the current set has no elements in common with the other set, and false otherwise.

**Method Description**: The method checks whether the calling set and the (secondSet) are disjoint, meaning they have no elements in common. If they are disjoint, it returns true; otherwise, it returns false.

public boolean isEmpty() {  
 return this.set.isEmpty();  
}

**Method Identifier**: isEmpty

**Method Declaration**: public Boolean isEmpty()

**Parameter**: N/A

**Return Type**: Boolean - true if the current set has no elements, and false otherwise.

**Method Description**: The method checks whether the current set has 0 or no elements.

public HashSet<HashSet<String>> powerSet() {  
 return generateSubsets(new ArrayList<>(this.set), 0);  
}

**Method Identifier**: powerSet

**Method Declaration**: public HashSet<HashSet<String>> powerSet()

**Parameter**: N/A

**Return type**: HashSet<HashSet<String>> - returns the complete power set created by the helper method ‘generateSubsets’.

**Method description**: This method generates the power set of a given set by calling the private method 'generateSubsets'. It creates a new ArrayList from the "set" attribute of the current object using the constructor ArrayList<>(this.set), and passes it along with an integer value of 0 as parameters to the 'generateSubsets' method. It returns the result obtained from the 'generateSubsets' method, which is a HashSet of Strings representing the power set.

private HashSet<HashSet<String>> generateSubsets(List<String> elements, int index) {  
 HashSet<HashSet<String>> allSubsets;  
  
 // Base Case  
 if (index == elements.size()) {  
 allSubsets = new HashSet<>();  
 allSubsets.add(new HashSet<>()); // add empty set  
 } else {  
 allSubsets = generateSubsets(elements, index + 1); // Recursive Case  
 String element = elements.get(index);  
 HashSet<HashSet<String>> moreSubsets = new HashSet<>();  
  
 for (HashSet<String> subset : allSubsets) {  
 HashSet<String> newSubset = new HashSet<>(subset);  
 newSubset.add(element);  
 moreSubsets.add(newSubset);  
 }  
  
 allSubsets.addAll(moreSubsets);  
 }  
  
 return allSubsets;  
}

**Method Identifier**: generateSubsets

**Method Declaration**: private HashSet<HashSet<String>> generateSubsets(List<String> elements, int index)

**Parameters**: List<String> elements and int index

**Return type**: HashSet<HashSet<String>> - recursively adds and returns subsets every time this method is called.

**Method description**: This method is a helper method used by the 'powerSet' method to generate all the subsets of a given list of elements. It takes two parameters; a List of Strings called 'elements' and an integer value called 'index'. The 'elements' parameter represents the list of elements for which the subsets need to be generated, and the 'index' parameter keeps track of the current position in the list while generating subsets repeatedly. The method returns a HashSet containing a HashSet of strings, which represents all the subsets of the given list of elements.

@Override  
public String toString() {  
 StringBuilder setFormat = new StringBuilder();  
 setFormat.append("{");  
 setFormat.append(String.*join*(", ", set));  
 setFormat.append("}");  
 return setFormat.toString();  
}

**Method Identifier**: toString

**Method Declaration**: public String toString()

**Parameter:** N/A

**Return type**: String – the string representation of the set.

**Method description**: This method is what will be called when a Set object is used as an argument for System.out.println(). It separates each element with a comma followed with a space and encloses them in braces.