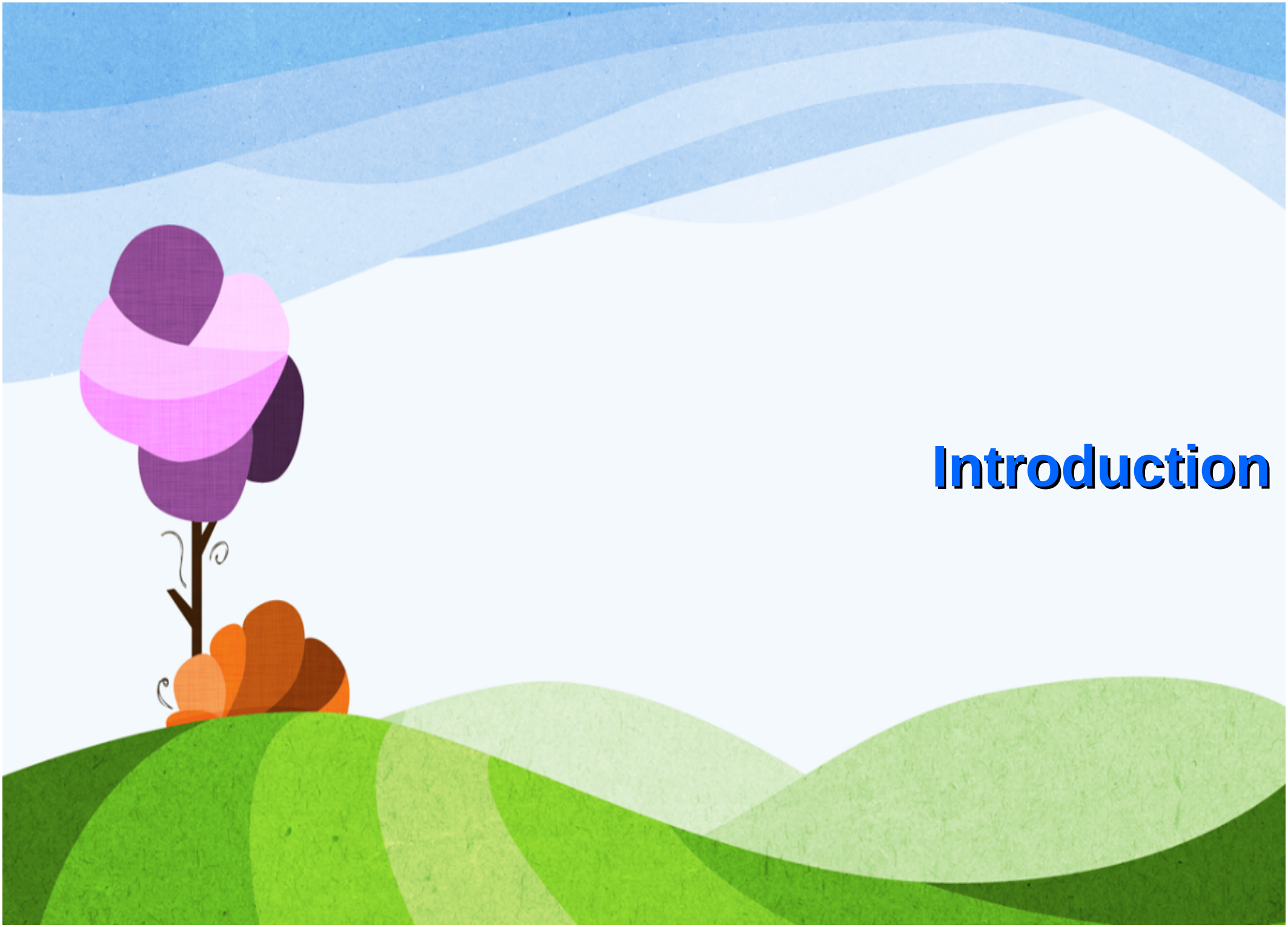
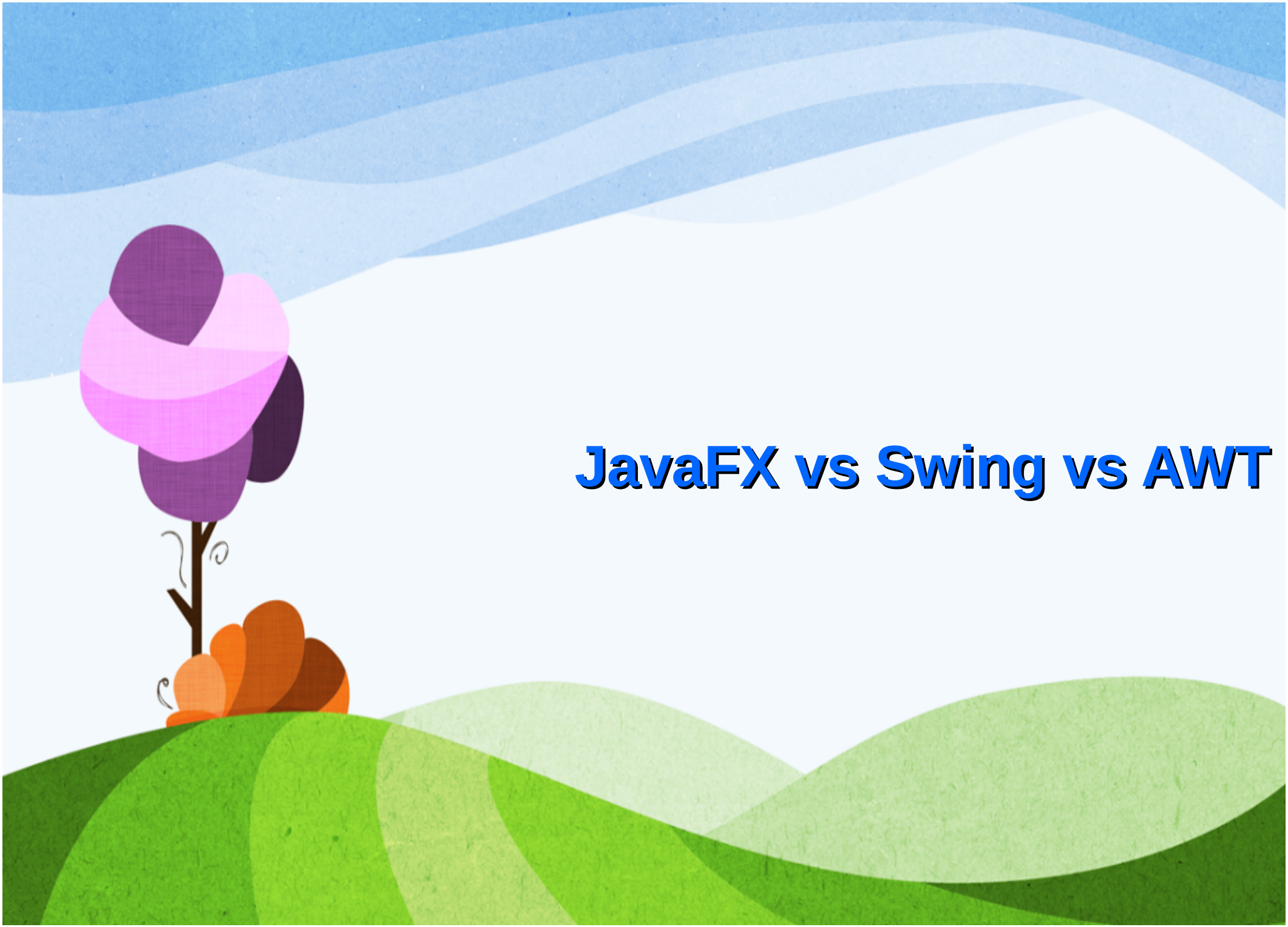

Introduction

Introduction

- JavaFX is a new framework for developing GUI (Graphical User Interface) Applications.
- The JavaFX API is a great example of how object oriented principles can be applied.
- This week we will learn how to develop GUI projects using layout panes, buttons, labels, text fields, colors, fonts, images, image views, and shapes.

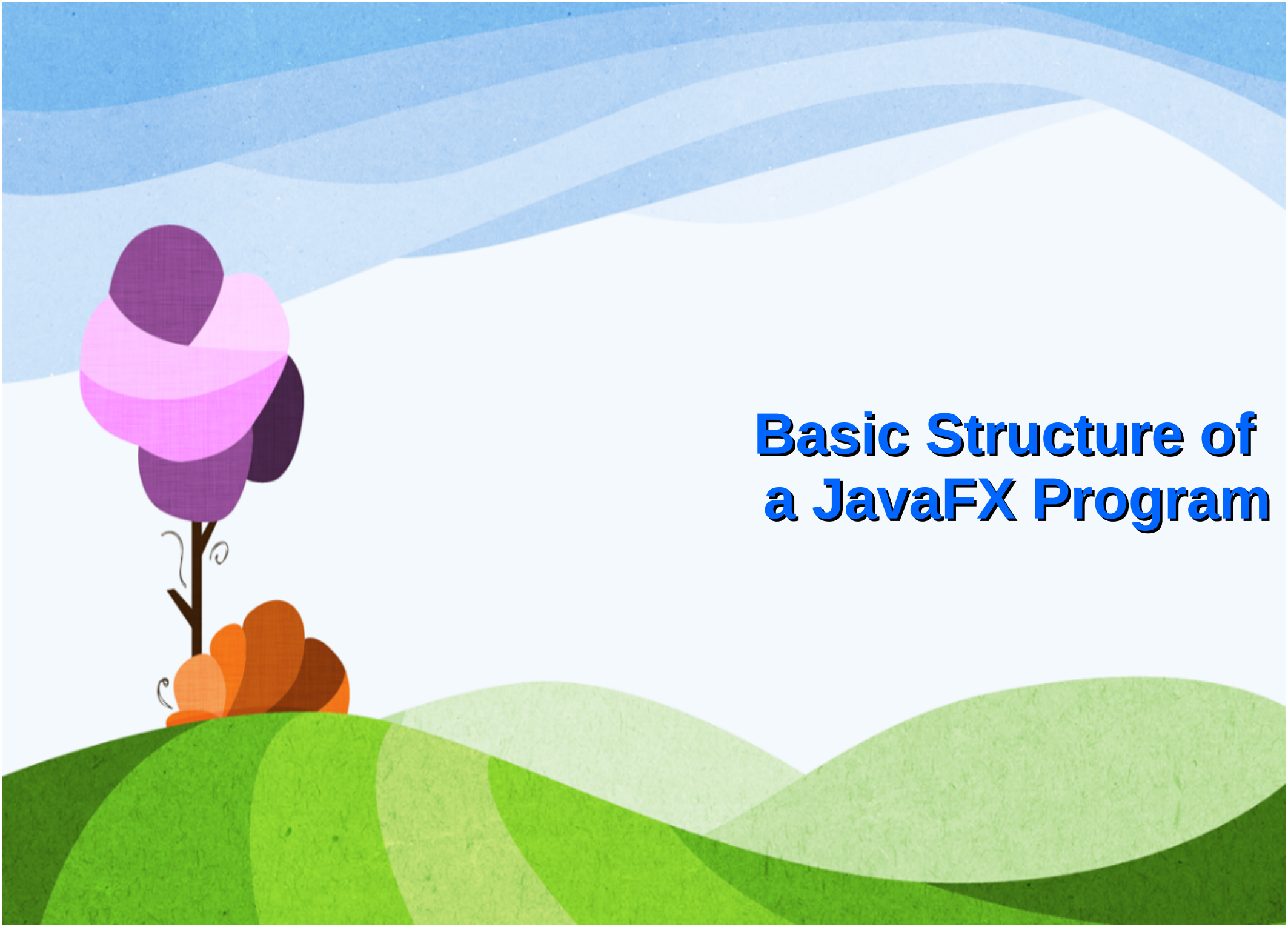

JavaFX vs Swing vs AWT

JavaFX vs Swing vs AWT

- AWT (Abstract Windows Toolkit)
  - first framework created for developing GUI apps in Java
  - only good for simple interfaces
  - prone to platform specific bugs
  - replaced by Swing
- Swing
  - more robust, versatile, and flexible
  - components are painted directly on canvases using Java code
  - less dependent on platform specific operations
  - will be completely replaced by JavaFX
  - Swing is essentially dead, no further updates.

JavaFX vs Swing vs AWT

- JavaFX
  - incorporates modern GUI technologies to enable the development of rich Internet Applications (RIA)
  - RIAs are web apps designed to deliver the same features and functions normally associated with desktop applications.
  - JavaFX apps can run seemlessly on a desktop or from a Web browser
  - provides multi-touch support for touch-enabled devices such as tablets and smartphones
  - has built in 2D and 3D animation support, video and audo playback, runs as a stand-alone application or from a browser.
  - Simpler to learn than AWT or Swing



Basic Structure of a JavaFX Program

Basic Structure of a JavaFX Program

- **Every** JavaFX program is defined in a class which extends **`javafx.application.Application`**

- Just extend the **`Application`** class and you can start making GUI programs.

LISTING 14.1 MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");
11     Scene scene = new Scene(btOK, 200, 250);
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title
13     primaryStage.setScene(scene); // Place the scene in the stage
14     primaryStage.show(); // Display the stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {
22     Application.launch(args);
23   }
24 }
```

MyJavaFX.java

- This program displays a window with a simple button on it.
- line 22: the **`Application.launch(args)`** method is a static method defined in the **`Application`** class.
  - if you run this program from the command line main is not needed.
  - some IDEs may need main to launch the application due to limited JavaFX support (such as Eclipse)

MyJavaFX.java

- line 8: the Driver class of your program needs to override the **`start`** method of Application.

```
public void start(Stage primaryStage)
```

- **`start()`** method
  - usually used to place UI controls in a scene, and display the scene in a stage

- Line 10: creates a **`Button`** object and places it in a **`Scene`** object
- A **`Scene`** object can be created using

```
Scene(node, width, height)
```

MyJavaFX.java

- A Stage object is a window
  - ***primary stage*** a Stage object automatically created by the JVM when the application is launched

FIGURE 14.2 (a) Stage is a window for displaying a scene that contains nodes. (b) Multiple stages can be displayed in a JavaFX program.

MultipleStageDemo.java

LISTING 14.2 MultipleStageDemo.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MultipleStageDemo extends Application {
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {
9      // Create a scene and place a button in the scene
10     Scene scene = new Scene(new Button("OK"), 200, 250);
11     primaryStage.setTitle("MyJavaFX"); // Set the stage title
12     primaryStage.setScene(scene); // Place the scene in the stage
13     primaryStage.show(); // Display the stage
14
15     Stage stage = new Stage(); // Create a new stage
16     stage.setTitle("Second Stage"); // Set the stage title
17     // Set a scene with a button in the stage
18     stage.setScene(new Scene(new Button("New Stage"), 100, 100));
19     stage.show(); // Display the stage
20   }
21 }
```


Panes, UI Controls and Shapes

Panes, UI Controls, and Shapes

- Panes, UI controls, and shapes, are all subtypes of the **Node** class.
- ***node***: visual components such as shapes, images, UI controls, or even a pane
- ***pane***: a container class which helps to automatically layout a node in a desired location and size.
  - nodes are placed inside of a pane and then the pane is placed into a scene
- ***shape***: text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.
- ***UI control***: a label, button, checkbox, radio button, text field, text area, etc.

JavaFX Class Hierarchy

- The relationship between a **`Stage`**, **`Scene`**, **`Node`**, **`Control`**, and **`Pane`** is shown on the next slide.
- A **`Scene`** can contain a **`Control`** or **`Pane`**, but not a **`Shape`** or **`ImageView`**
- A **`Pane`** can contain any subtype of **`Node`**
- A **`Scene`** can be created with:
  - **`Scene(Parent, width, height)`**
  - **`Scene(Parent)`** (dimensions are automatically decided)
- Every subclass of **`Node`** has a no-arg constructor for creating a default node.

JavaFX Class Hierarcy

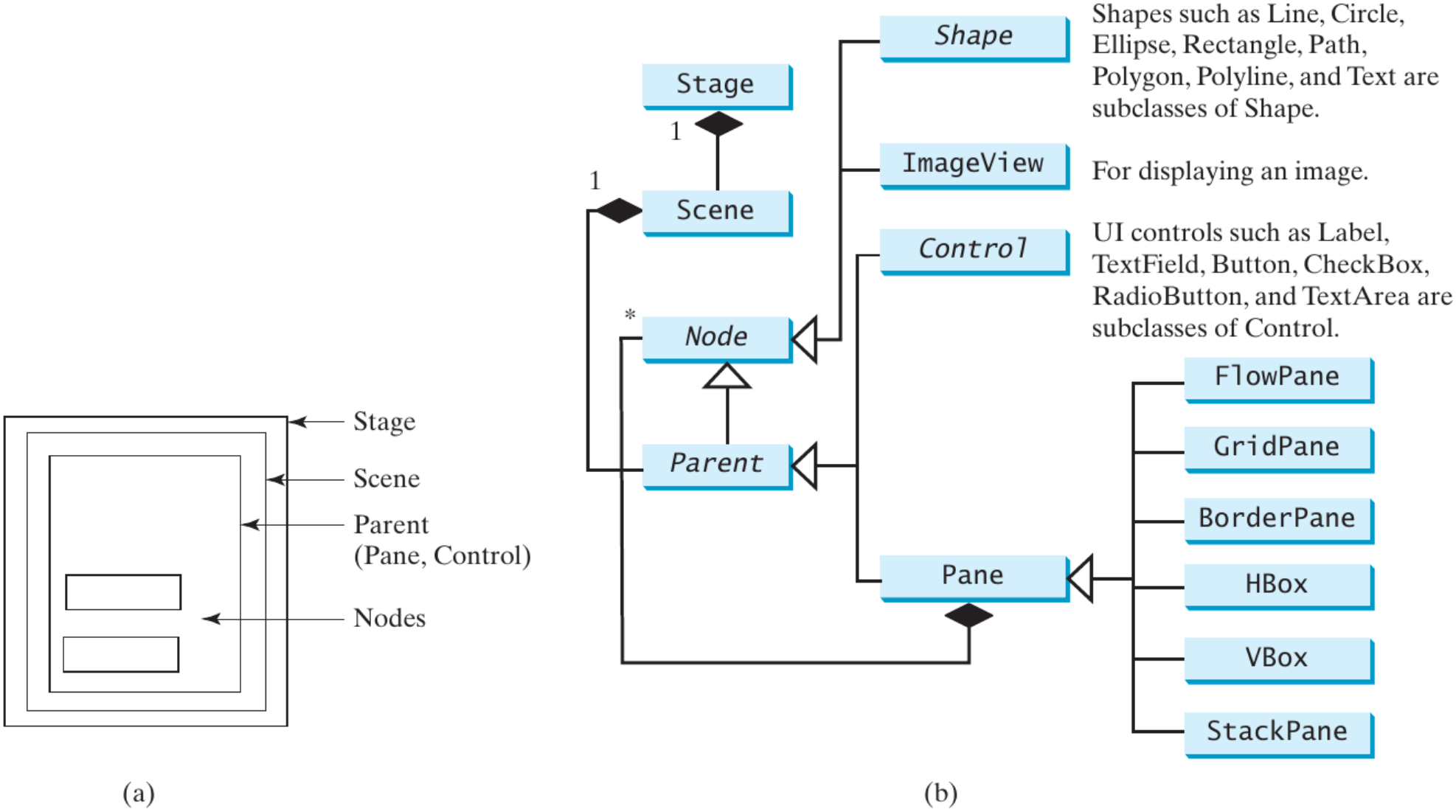


FIGURE 14.3 (a) Panes are used to hold nodes. (b) Nodes can be shapes, image views, UI controls, and panes.

ButtonInPane.java

LISTING 14.3 ButtonInPane.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5  import javafx.scene.layout.StackPane;
6
7  public class ButtonInPane extends Application {
8    @Override // Override the start method in the Application class
9    public void start(Stage primaryStage) {
10     // Create a scene and place a button in the scene
11     StackPane pane = new StackPane();
12     pane.getChildren().add(new Button("OK"));
13     Scene scene = new Scene(pane, 200, 50);
14     primaryStage.setTitle("Button in a pane"); // Set the stage title
15     primaryStage.setScene(scene); // Place the scene in the stage
16     primaryStage.show(); // Display the stage
17   }
18 }
```

ButtonInPane.java

- line 11: program creates a **`StackPane`**
  - **`StackPane`** places nodes in the center of the pane on top of each other.
  - **`StackPane`** respects a node's preferred size (it does not change the size of the components to fit the window)
- line 12: adds a button as a child of the pane
  - **`getChildren()`** returns an instance of **`ObservableList`**
  - **`ObservableList`** is a lot like an **`ArrayList`** for storing collections of elements
  - **`add(e)`** adds an element to the list

ShowInCircle.java

LISTING 14.4 ShowCircle.java

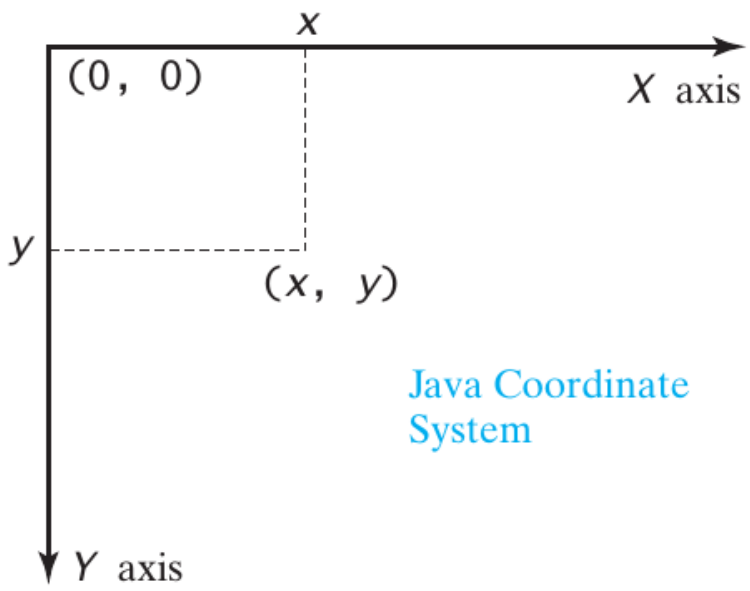
```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.stage.Stage;
7
8  public class ShowCircle extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a circle and set its properties
12     Circle circle = new Circle();
13     circle.setCenterX(100);
14     circle.setCenterY(100);
15     circle.setRadius(50);
16     circle.setStroke(Color.BLACK);
17     circle.setFill(Color.WHITE);
18
19     // Create a pane to hold the circle
20     Pane pane = new Pane();
21     pane.getChildren().add(circle);
22
23     // Create a scene and place it in the stage
24     Scene scene = new Scene(pane, 200, 200);
25     primaryStage.setTitle("ShowCircle"); // Set the stage title
26     primaryStage.setScene(scene); // Place the scene in the stage
27     primaryStage.show(); // Display the stage
28   }
29 }
```

ShowInCircle.java

- line 12: creates a **`Circle`**
- lines 13-14: set the center to be (100, 100)
  - also center of **`Scene`**, since size of **Scene** is 200 x 200
  - all measurements are in ***pixels***
- line 16: stroke color set to black (line color)
- line 17: fill color set to white (color to fill the shape)
  - color could be **`null`**
- line 20: creates a **`Pane`**, puts circle in pane
- line 24: **`Pane`** is placed in the **`Scene`**
- line 26: **`Scene`** is set in the **`Stage`**
- NOTE: circle is centered as long as you do not resize the window. The next section will show how to fix this

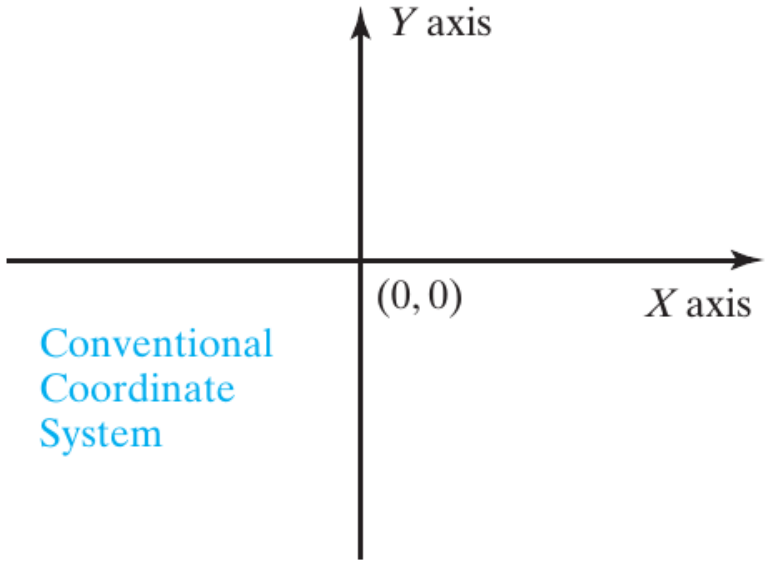
JavaFX Coordinate System

- the upper left corner of a Pane or Scene is used as (0,0) in the Java coordinate system.
- does not use the conventional coordinate system where (0,0) is the center of the Pane or Scene



Java Coordinate System

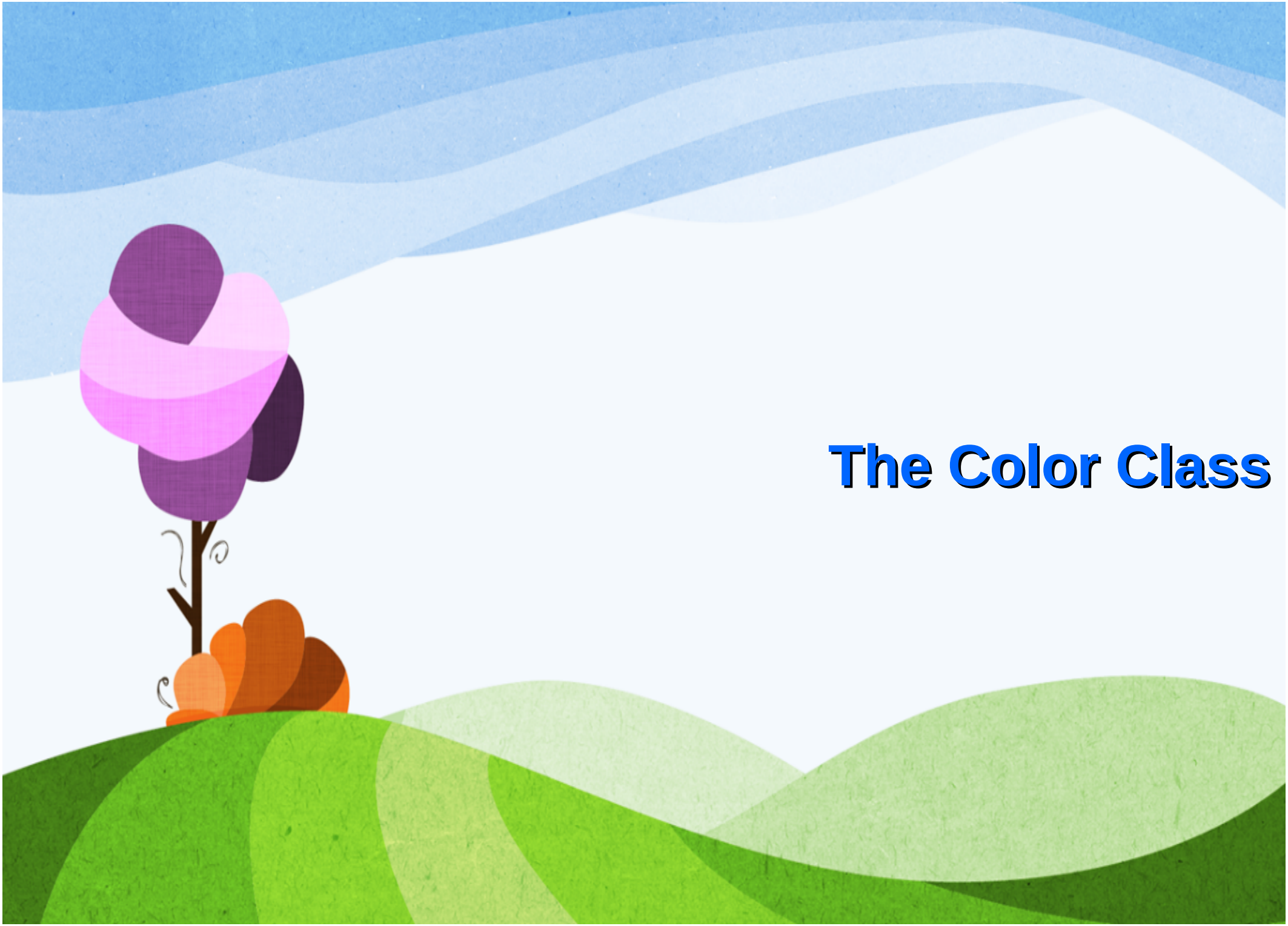
(a)



Conventional Coordinate System

(b)

FIGURE 14.6 The Java coordinate system is measured in pixels, with **(0, 0)** at its upper-left corner.


The Color Class

The Color Class

- used to create colors
- The abstract Paint class is used for painting a Node
- **`javafx.scene.paint.Color`** is a concrete subclass of paint used to encapsulate colors

The Color Class

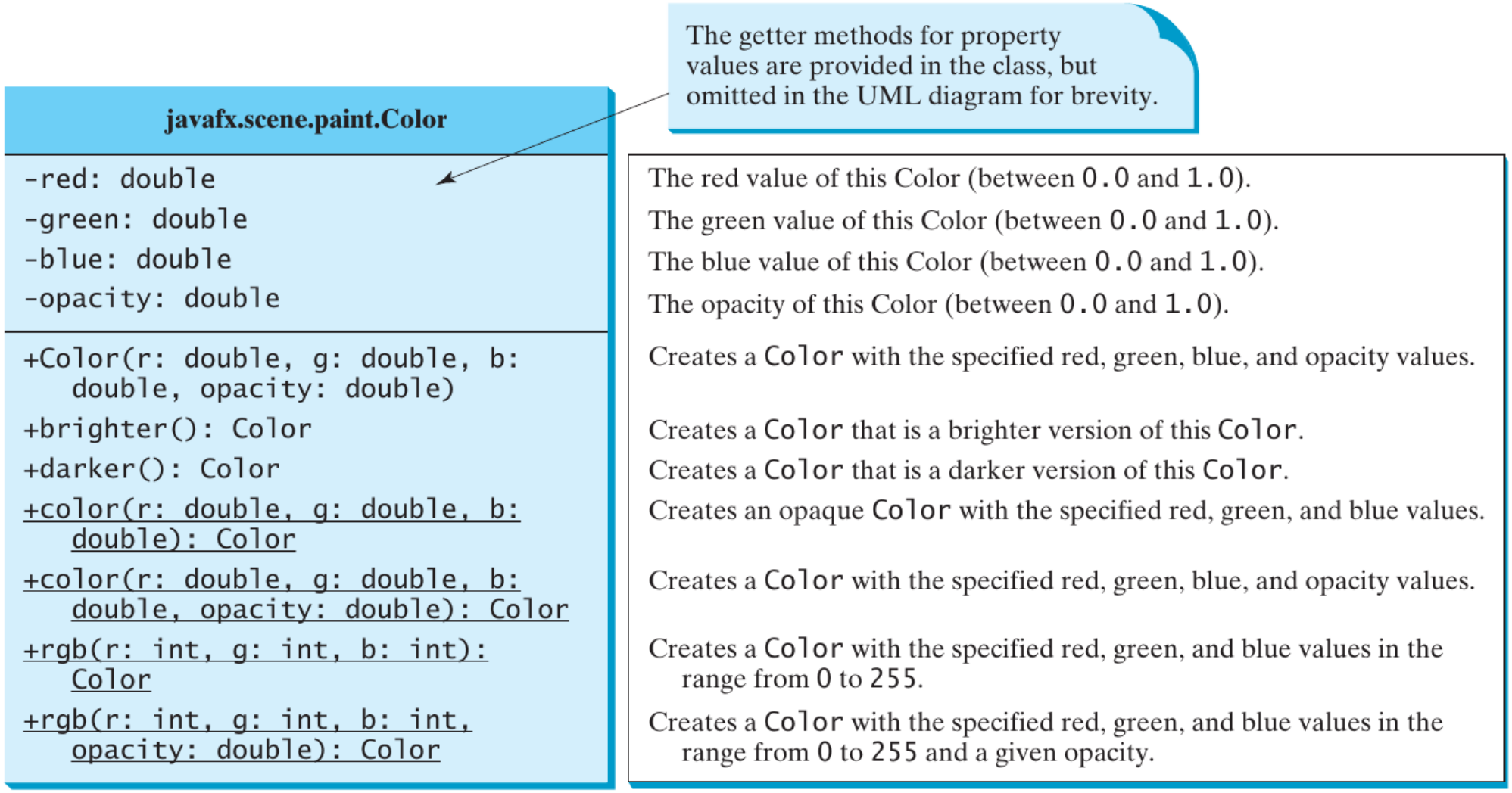


FIGURE 14.9 `Color` encapsulates information about colors.

The Color Class

- Creating a color:

```
public Color(double r, double g, double b, double opacity);
```

  - r, g, b are the red, green, blue components of a color, value range from 0.0 (darkest) to 1.0 (lightest shade)
  - opacity is the color transparency between 0.0 (completely transparent) to 1.0 (completely opaque)
  - This follows the RGBA (red, green, blue, alpha) color model.

- Example:

```
Color color = new Color(0.25, 0.14, 0.333, 0.51);
```

The Color Class

- **`Color`** is immutable
  - any methods which "alter" the color actually create a new object and return that object.
- You can also create a color using some of the static methods of the Color class.
- You can also use some of the standard colors which are defined as constants in the Color class.
  - See the API for Color
  - Example: **`circle.setFill(Color.RED);`**


The Font Class

The Font Class

- used to create and set fonts and change their properties.
- Font has a name, weight, posture, and size.
- You can see a list of available font family names by calling getFamilies().
- FontPosture is for italics or no italics:
  - FontPosture.ITALIC
  - FontPosture.REGULAR

```
Font font1 = new Font("SansSerif", 16);
Font font2 = Font.font("Times New Roman", FontWeight.BOLD,
  FontPosture.ITALIC, 12);
```

The Font Class

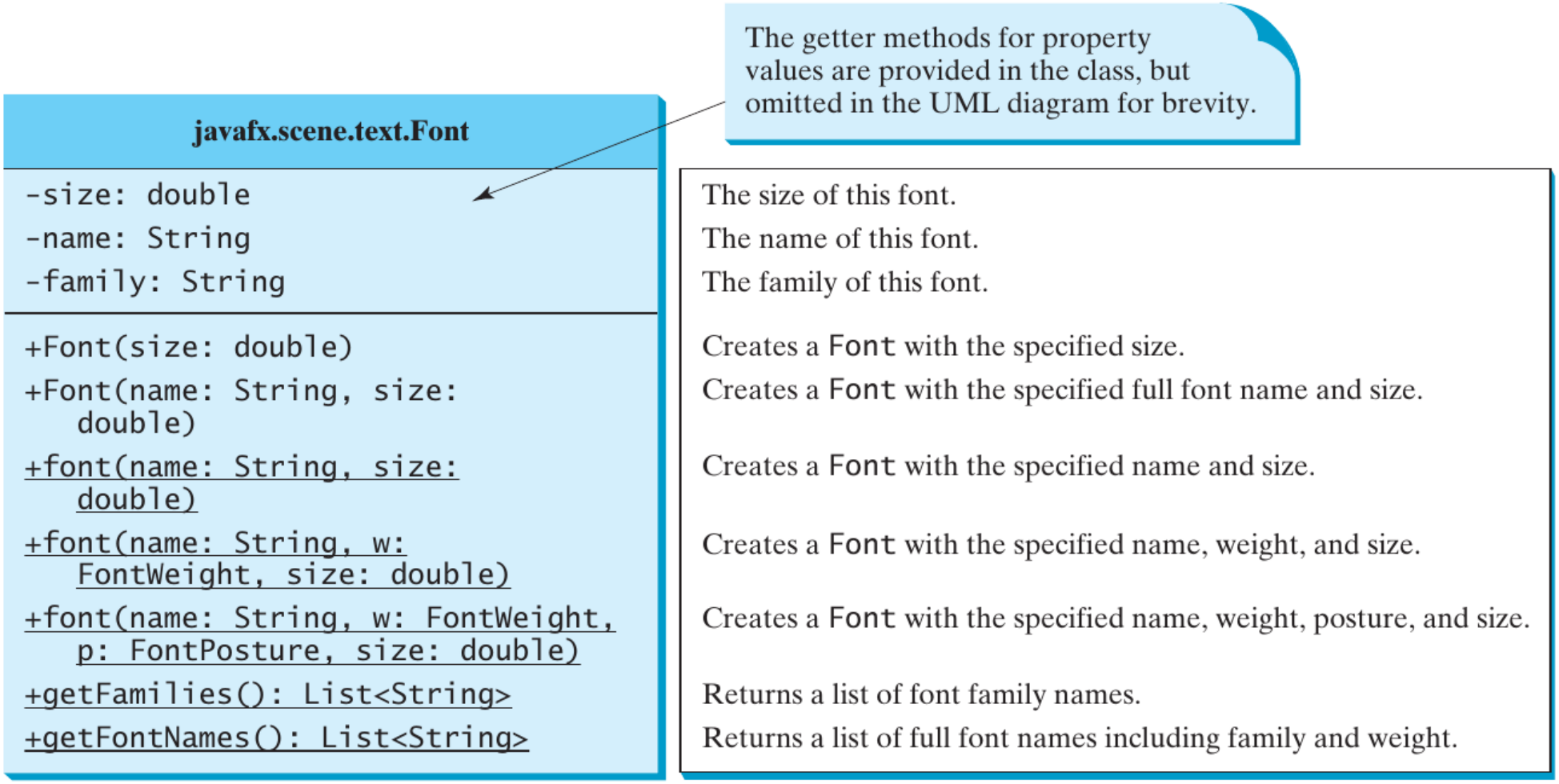


FIGURE 14.10 `Font` encapsulates information about fonts.