UNIVERSITÉ Concordia UNIVERSITY

CENTRE FOR
CONTINUING EDUCATION

DESKTOP APPLICATION DEVELOPMENT WITH JAVA – CEJV569

Lecture #5

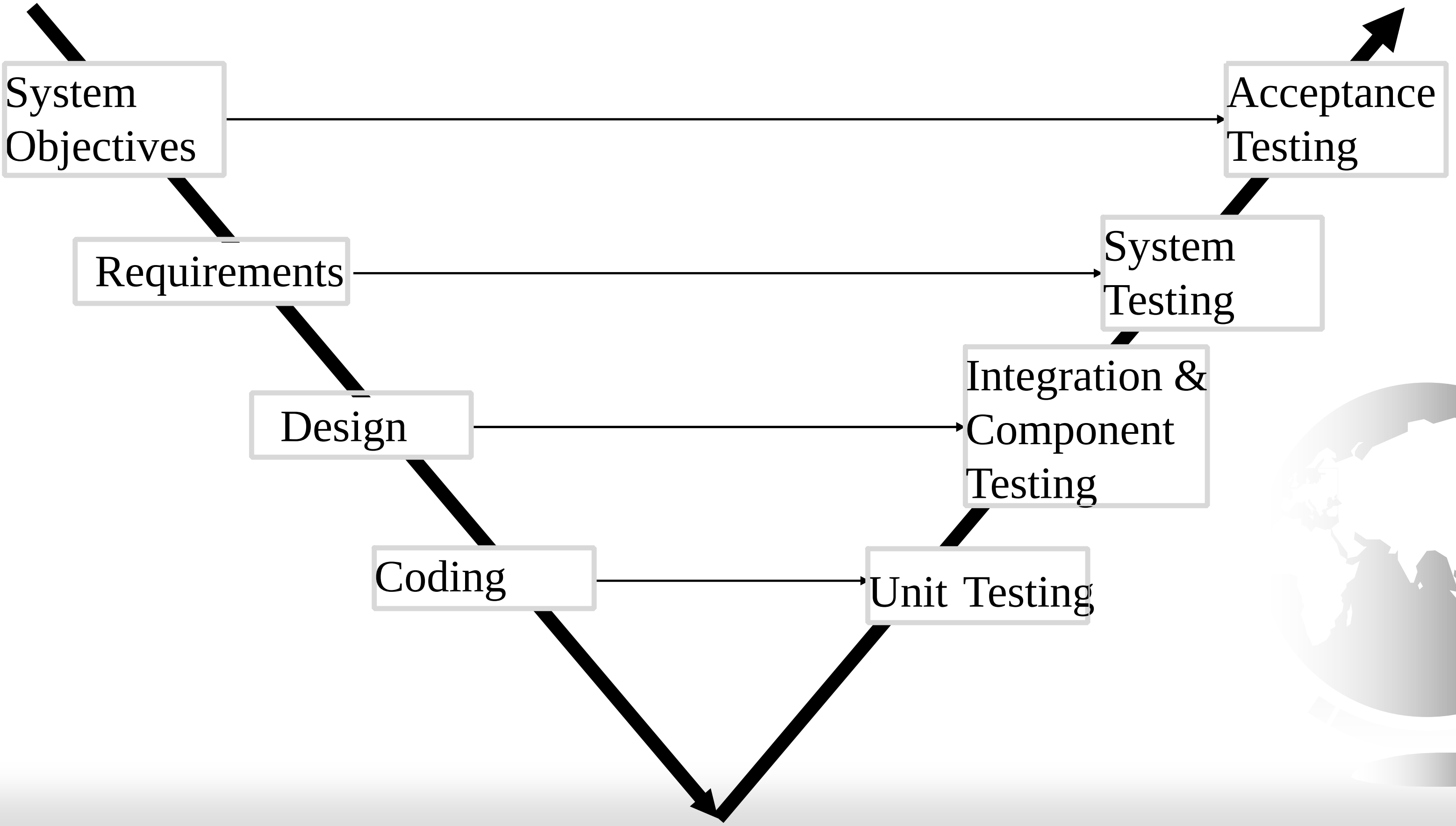
Testing

JavaFX

Test Driven Development

Traditional Development

- Testing happens after the code is written
- Tests are designed to prove what has been written will work



System Objectives

Requirements

Design

Coding

Acceptance Testing

System Testing

Integration & Component Testing

Unit Testing

An Important Message

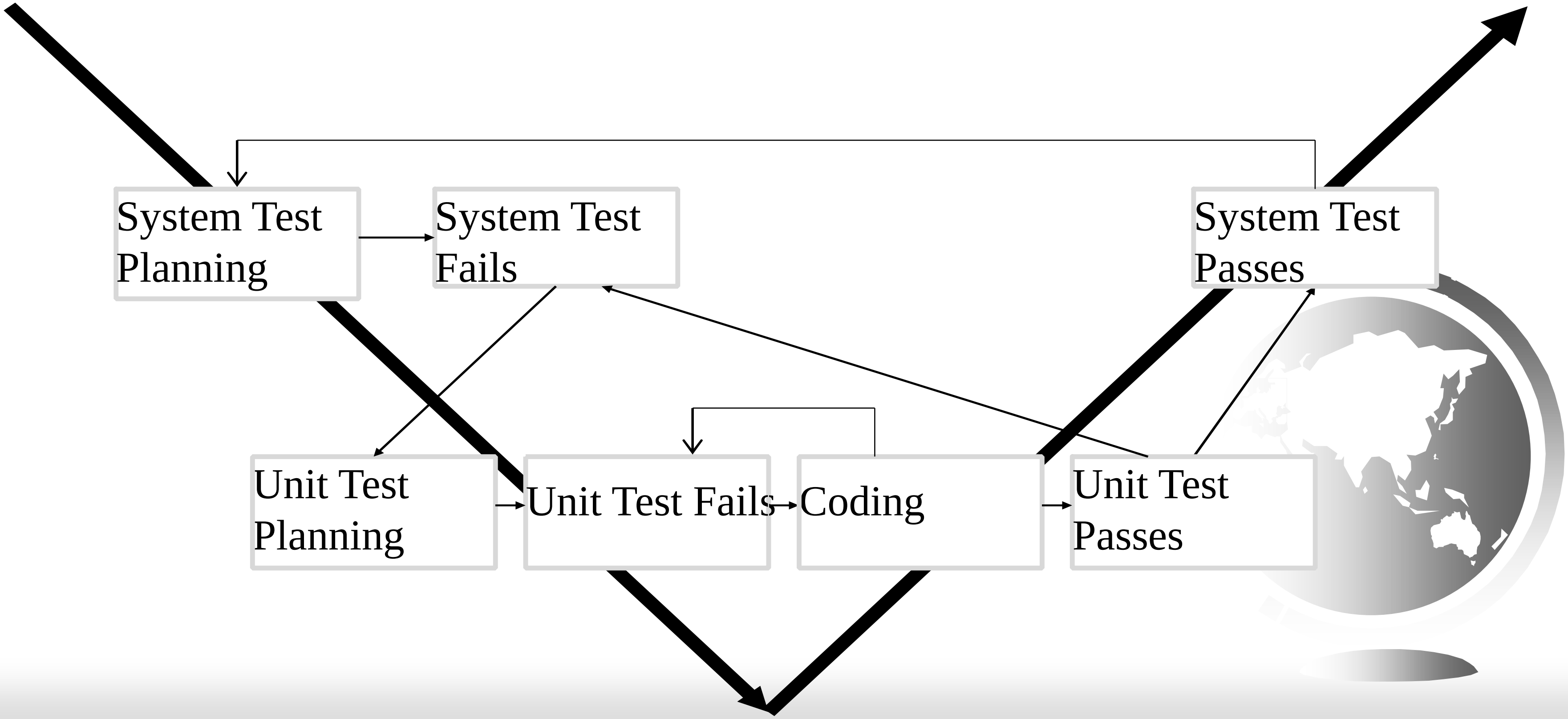
“More than the act of testing, the act of designing tests is one of the best [defect] preventers known …

The thought process that must take place to create useful tests can discover and eliminate problems at every stage of development”

Boris Beizer

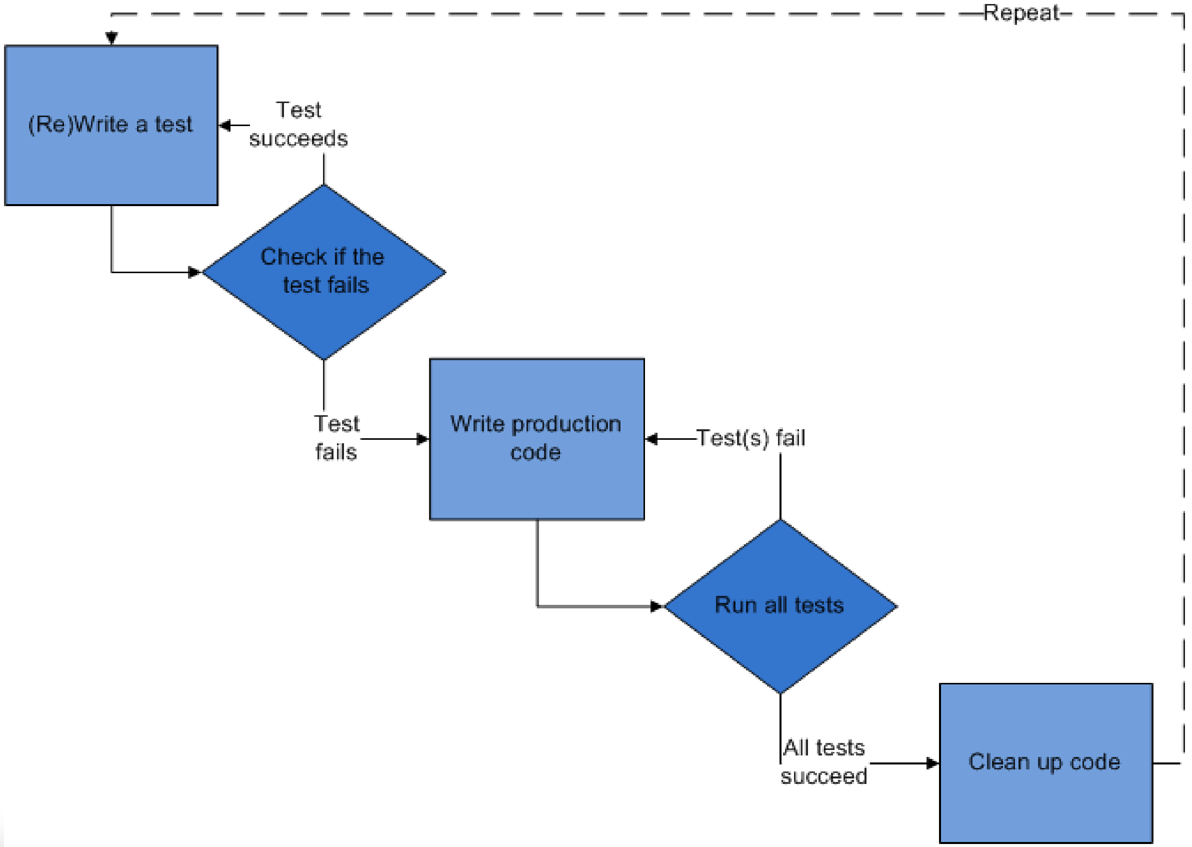
Test Driven Development

- Part of the Extreme Programming (XP) methodology
- Create test cases before writing code
- Only write new code when there is a test that doesn't work



Test Driven Development Cycle

- Add a test
- Run all tests and see if the new one fails
- Write some code
- Run tests
- Refactor code
- Repeat



Testing behaviour, not implementation

- Test that something does what it's supposed to do
- Test the interface, it's contract
  - Shouldn't care how it does it
- Test the public methods of a unit
- This will cover all of the private methods without the need to change something's visibility
- If something private is still screaming out to be tested then it's probably important enough to be extracted
- You should be able to change how the unit does something (different datastructure, different algorithm, private methods ahoy etc..) without changing your test

Unit Testing

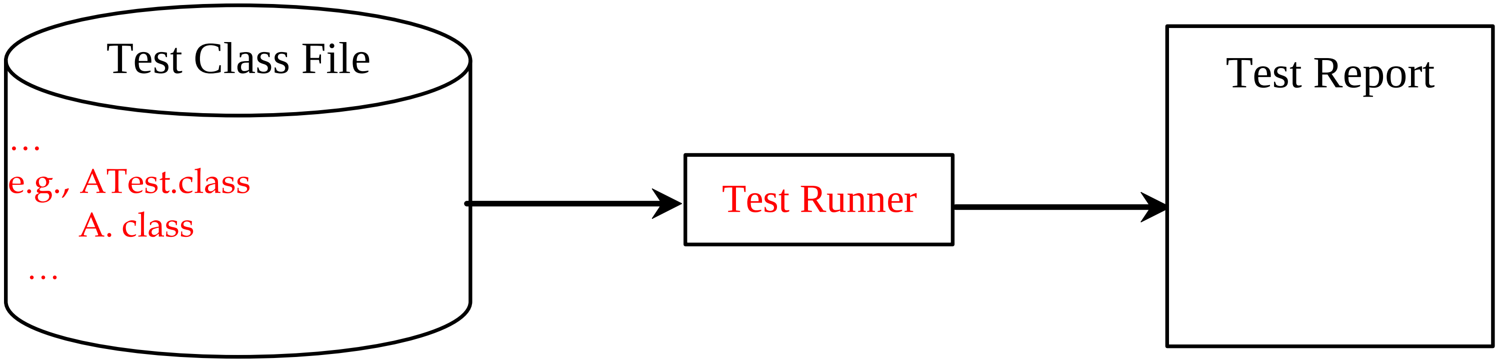
- Unit is a portion of source code such as an interface or a class
- Unit testing is a method by which these units can be tested to determine if they perform as expected
- Produces a report that indicates the success or failure of each test
- Avoids “Scroll Blindness” from too many print statements

JUnit Testing Framework

- Standard method for writing code to test units
- Consists of a library of routines with which to write tests
- Consists of executable code that runs the tests and produces the report
- JUnit is integrated into both Eclipse and Netbeans
- Will be using JUnit4 rather than JUnit3

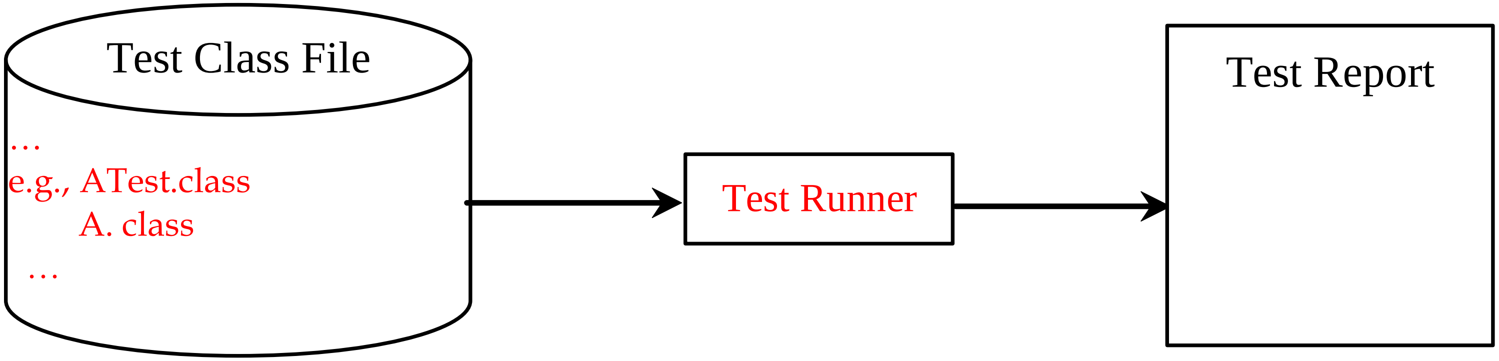
JUnit Basics

- *JUnit* is the de facto framework for testing Java programs.
- JUnit is a third-party open source library packed in a jar file.
- The jar file contains a tool called *test runner*, which is used to run test programs.



JUnit Basics

- Suppose you have a class named A.
- To test this class, you write a test class named ATest.
- This test class, called a *test class*, contains the methods you write for testing class A.
- The test runner executes ATest to generate a test report.



Test Class File

…
e.g., ATest.class
A. class
…

Test Runner

Test Report

JUnit4 uses annotations

- Special form of syntatic metadata that can be added to Java source code
- Hint or message added to Java source code
- Embedded in the class file and are available to libraries and tools
- Provides information to the run-time that can be used to change or better describe the meaning of the code

Test Unit

- A class that contains annotated methods
- Methods execute code in the targeted unit
- Output of these methods make up the unit test reports
- Methods can have any name
- Annotations determine the function of the test
- JUnit3 requires methods have specific names or prefixes/suffixes rather than annotations

A JUnit Test Class

To use JUnit, create a test class. By convention, if the class to be tested is named A, the test class should be named ATest. A simple template of a test class may look like this:

```
package mytest;
import org.junit.*;
import static org.junit.Assert.*;
public class ATest {
  @Test
  public void m1() {
    // Write a test method
  }
  @Test
  public void m2() {
    // Write another test method
  }
  @Before
  public void setUp() throws Exception {
    // Common objects used by test methods may be set up here
  }
}
```

Set Up and Tear Down - all tests

```
@BeforeClass
public void method()
```

☞ performs the method before the start of all tests

☞ used to perform time intensive activities

```
@AfterClass
public void method()
```

☞ performs the method after all tests have finished

☞ used to perform clean-up activities

Set Up and Tear Down – each test

```
@Before
public void method()
```

☞ performs the method() before each test

☞ method can prepare the test environment

```
@After
public void method()
```

☞ performs the method() after each test

Test Methods

```
@Test
public void method()
```

☞ identifies that this method is a test method.

```
@Ignore
public void method()
```

☞ ignore the test method

Exception and Timeout

```
@Test(expected=SQLException.class)
public void method()
```

- tests if the method throws the named exception

```
@Test(timeout=100)
public void method()
```

- fails if the method takes longer then 100 milliseconds

Assert Statements

- JUnit library provided family of test methods
- Failure of a test in these methods generates an AssertionError
- JUnit Framework catches the exception and indicates that the test failed in the report
- Test that does not throw an exception appears in the report as a success

Assert Test Methods

`fail([String msg])`

- Generates an AssertionError
- Optional messsage included in the report
- Message is optional for all assert methods

`assertTrue(boolean condition)`

- Generates an AssertionError is the condition is false

`assertFalse(boolean condition)`

- Generates an AssertionError is the condition is true

Assert Test Methods

`assertsEquals(expected,actual)`

- Test if the values are the same
- Overloaded version for all primitives & Object
- Object must have a comparable interface

`assertsEquals(expected,actual,tolerance)`

- Special version for doubles and float
- Tolerance is the number of decimals that must be the same

`assertsArrayEquals(expected,actual)`

- Test if the values in the two arrays are the same
- Overloaded version for all primitives & Object

Assert Test Methods

`assertNull(object)`

☞ Checks if the object is null

`assertNotNull(object)`

☞ Checks if the object is not null

`assertSame(expected, actual)`

☞ Checks if both variables refer to the same object

`assertNotSame(expected, actual)`

☞ Check that both variables do not refer to the same object

Summary

- ☞ Tests need failure atomicity (ability to know exactly what failed).
  - – Each test should have a clear, long, descriptive name.
    - Netbeans can automatically generate test methods based on the class you are testing
    - Use your own method names
  - – Assertions should always have clear messages to know what failed.
  - – Write many small tests, not one big test.
    - Each test should have roughly just 1 assertion at its end.

Summary

- Always use a `timeout` parameter to every test.
- Test for expected errors / exceptions.
- Choose a descriptive assert method, not always `assertTrue`.
- Avoid complex logic in test methods if possible.
- Use set-up and tear-down methods to reduce redundancy between tests.

Exercise 20

- Open Exercise 19
- Write test units to test the following actions:
  - findAll()
  - findID()
  - findDiet()
  - create()
  - update()
  - delete()

JavaFX Basics

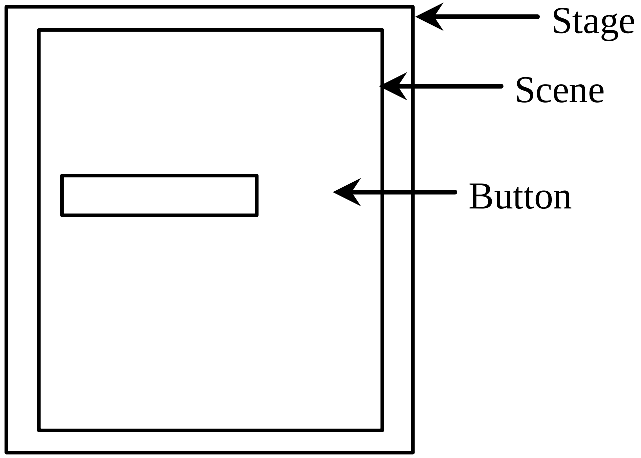
JavaFX vs Swing and AWT

- Swing and AWT are replaced by the JavaFX platform for developing rich Internet applications.

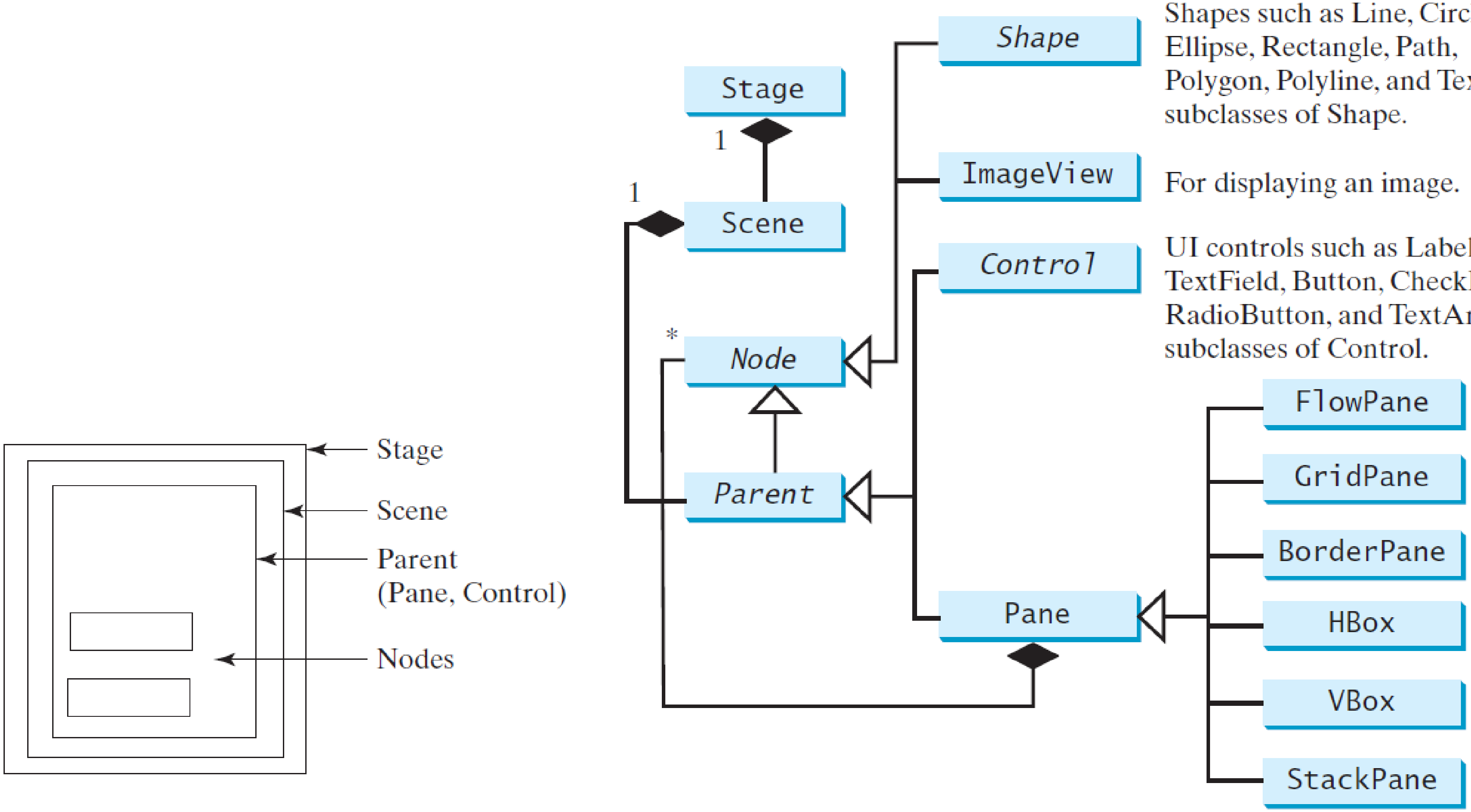
- When Java was introduced, the GUI classes were bundled in a library known as the *Abstract Windows Toolkit (AWT)*.
  - AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
  - In addition, AWT is prone to platform-specific bugs.
- The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*.
  - Swing components are painted directly on canvases using Java code.
  - Swing components depend less on the target platform and use less of the native GUI resource.
- With the release of Java 8, Swing is replaced by a completely new GUI platform known as *JavaFX*.

Basic Structure of JavaFX

- ☞ Application
- ☞ Override the start(Stage) method
- ☞ Stage, Scene, and Nodes



Panes, UI Controls, and Shapes



Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.

For displaying an image.

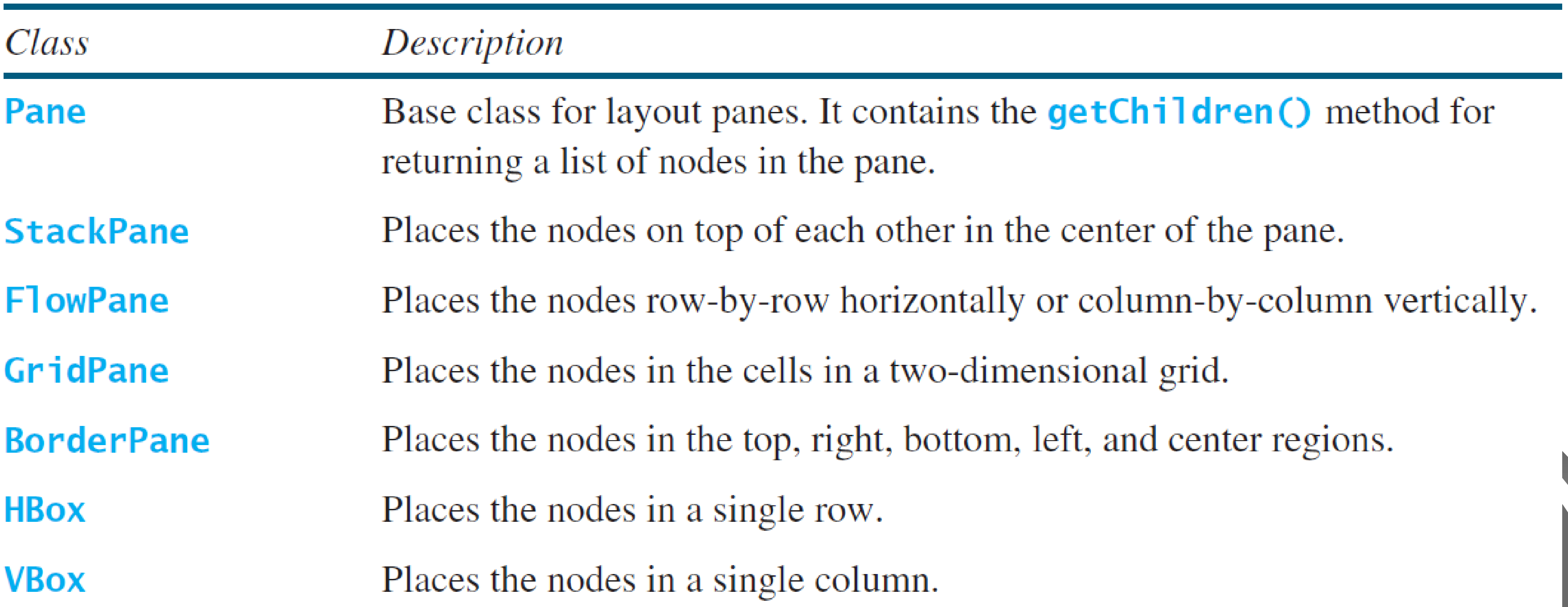
UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.

(a)

(b)

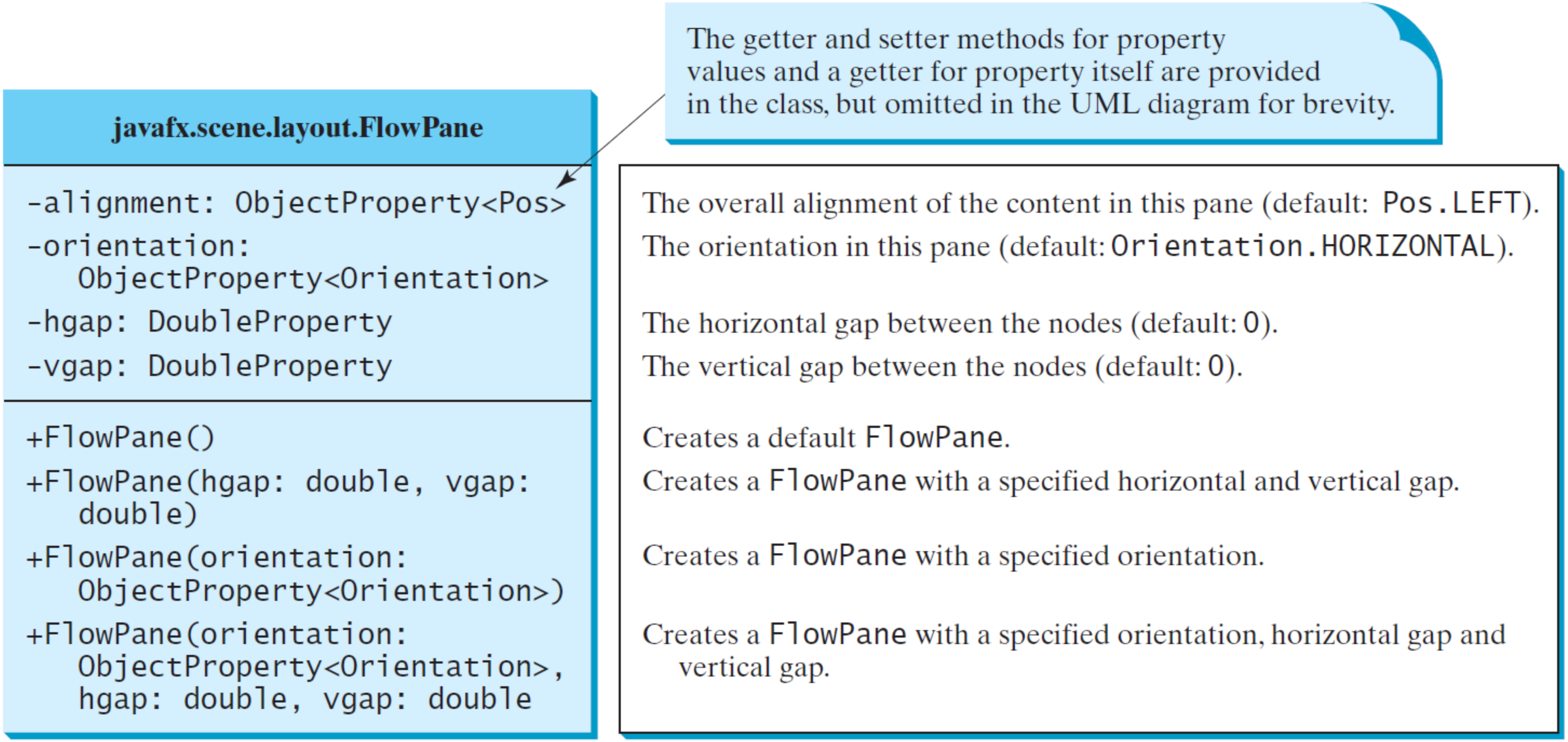
Layout Panes

JavaFX provides many types of panes for organizing nodes in a container.



Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

FlowPane

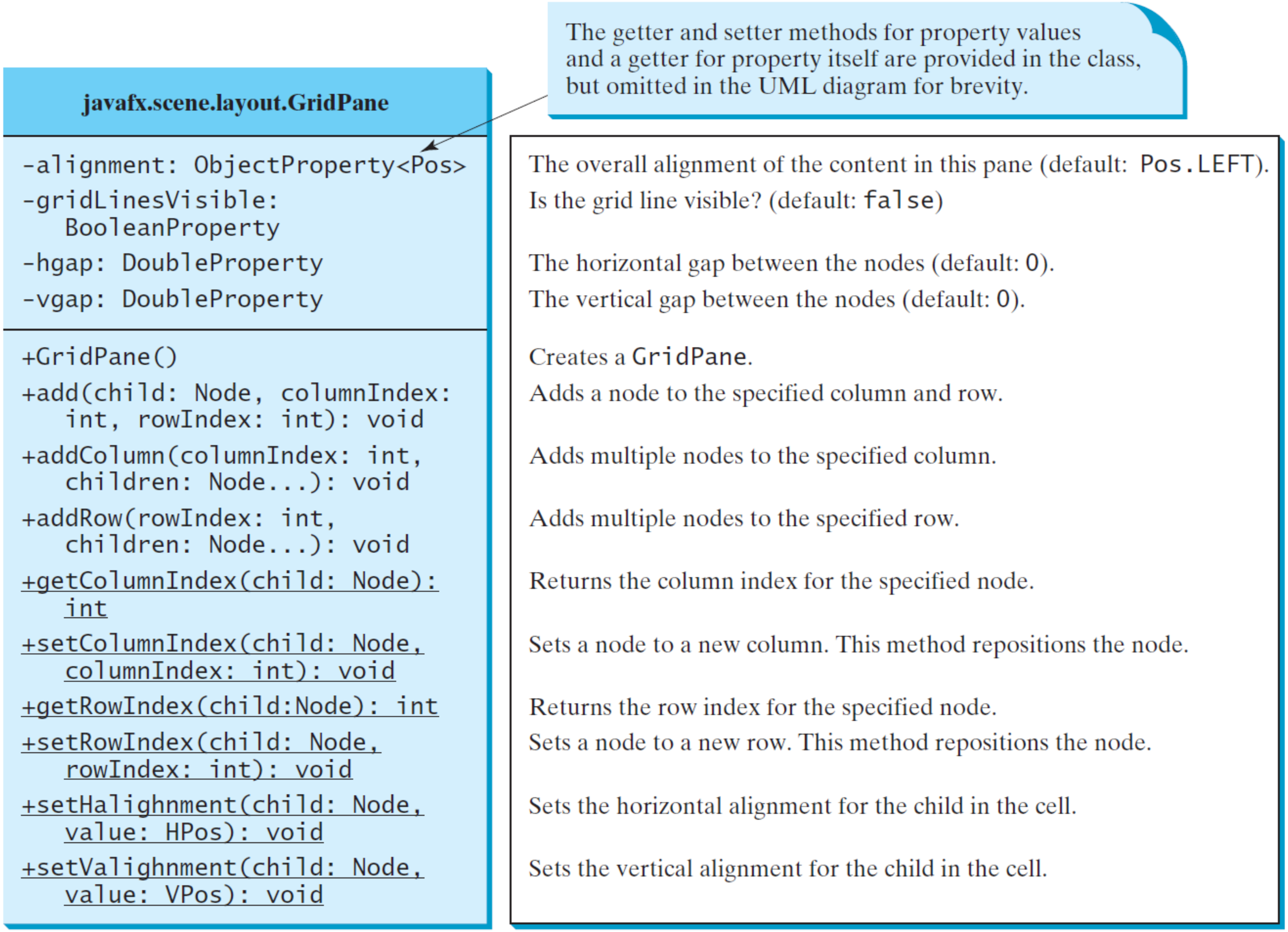


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.FlowPane

`-alignment: ObjectProperty<Pos>`	The overall alignment of the content in this pane (default: `Pos.LEFT`).
`-orientation: ObjectProperty<Orientation>`	The orientation in this pane (default: `Orientation.HORIZONTAL`).
`-hgap: DoubleProperty`	The horizontal gap between the nodes (default: `0`).
`-vgap: DoubleProperty`	The vertical gap between the nodes (default: `0`).
`+FlowPane()`	Creates a default `FlowPane`.
`+FlowPane(hgap: double, vgap: double)`	Creates a `FlowPane` with a specified horizontal and vertical gap.
`+FlowPane(orientation: ObjectProperty<Orientation>)`	Creates a `FlowPane` with a specified orientation.
`+FlowPane(orientation: ObjectProperty<Orientation>, hgap: double, vgap: double`	Creates a `FlowPane` with a specified orientation, horizontal gap and vertical gap.

GridPane

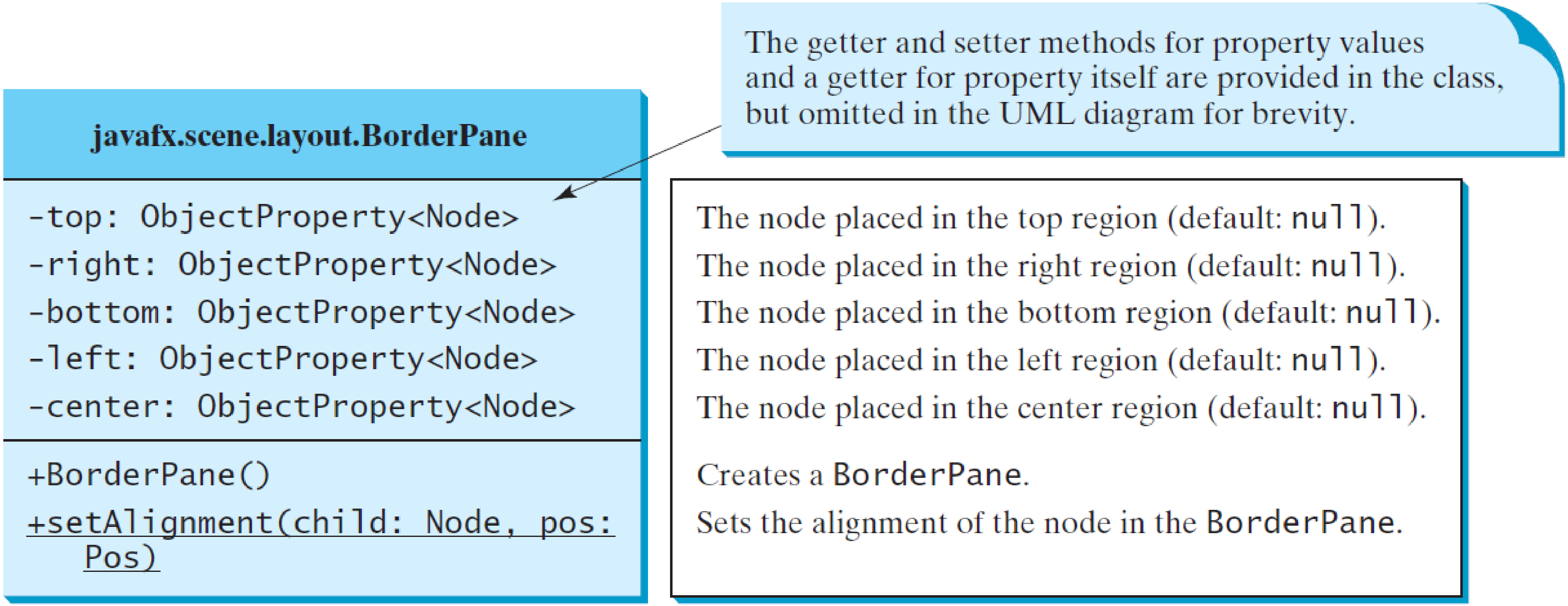


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.GridPane

`-alignment: ObjectProperty<Pos>`	The overall alignment of the content in this pane (default: `Pos.LEFT`).
`-gridLinesVisible: BooleanProperty`	Is the grid line visible? (default: `false`)
`-hgap: DoubleProperty`	The horizontal gap between the nodes (default: `0`).
`-vgap: DoubleProperty`	The vertical gap between the nodes (default: `0`).
`+GridPane()`	Creates a `GridPane`.
`+add(child: Node, columnIndex: int, rowIndex: int): void`	Adds a node to the specified column and row.
`+addColumn(columnIndex: int, children: Node...): void`	Adds multiple nodes to the specified column.
`+addRow(rowIndex: int, children: Node...): void`	Adds multiple nodes to the specified row.
`+getColumnIndex(child: Node): int`	Returns the column index for the specified node.
`+setColumnIndex(child: Node, columnIndex: int): void`	Sets a node to a new column. This method repositions the node.
`+getRowIndex(child:Node): int`	Returns the row index for the specified node.
`+setRowIndex(child: Node, rowIndex: int): void`	Sets a node to a new row. This method repositions the node.
`+setHalighnment(child: Node, value: HPos): void`	Sets the horizontal alignment for the child in the cell.
`+setValighnment(child: Node, value: VPos): void`	Sets the vertical alignment for the child in the cell.

BorderPane



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.BorderPane

```
-top: ObjectProperty<Node>
-right: ObjectProperty<Node>
-bottom: ObjectProperty<Node>
-left: ObjectProperty<Node>
-center: ObjectProperty<Node>

+BorderPane()
+setAlignment(child: Node, pos:
   Pos)
```

The node placed in the top region (default: `null`).

The node placed in the right region (default: `null`).

The node placed in the bottom region (default: `null`).

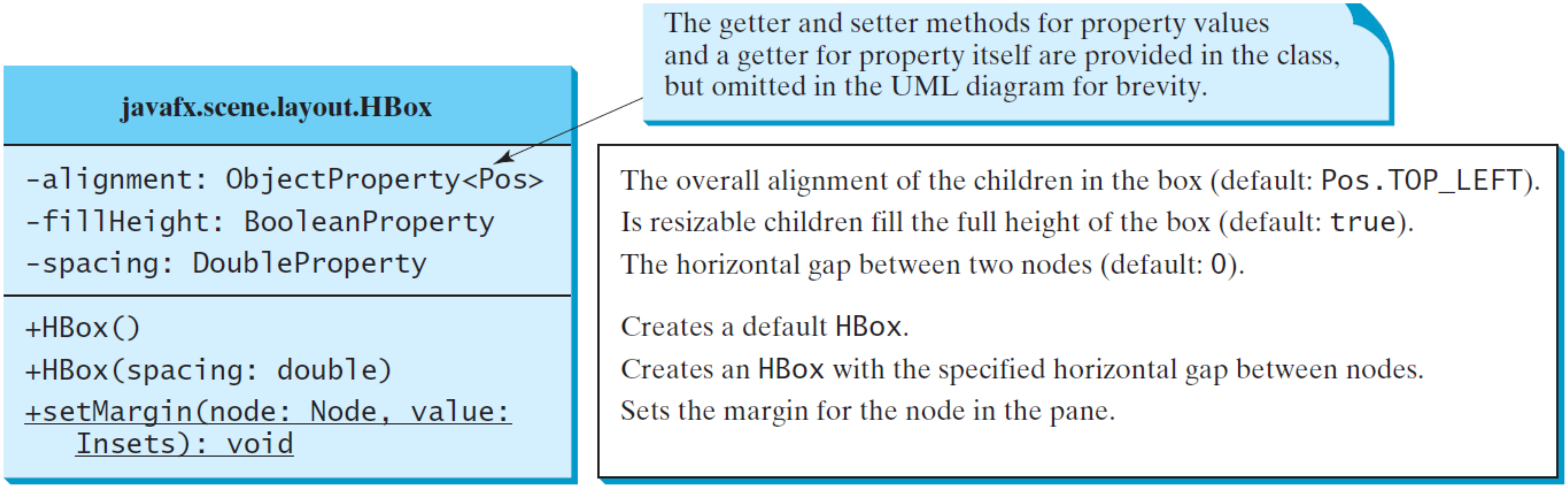
The node placed in the left region (default: `null`).

The node placed in the center region (default: `null`).

Creates a `BorderPane`.

Sets the alignment of the node in the `BorderPane`.

HBox



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

```
javafx.scene.layout.HBox

-alignment: ObjectProperty<Pos>
-fillHeight: BooleanProperty
-spacing: DoubleProperty

+HBox()
+HBox(spacing: double)
+setMargin(node: Node, value:
   Insets): void
```

The overall alignment of the children in the box (default: Pos.TOP_LEFT).

Is resizable children fill the full height of the box (default: true).

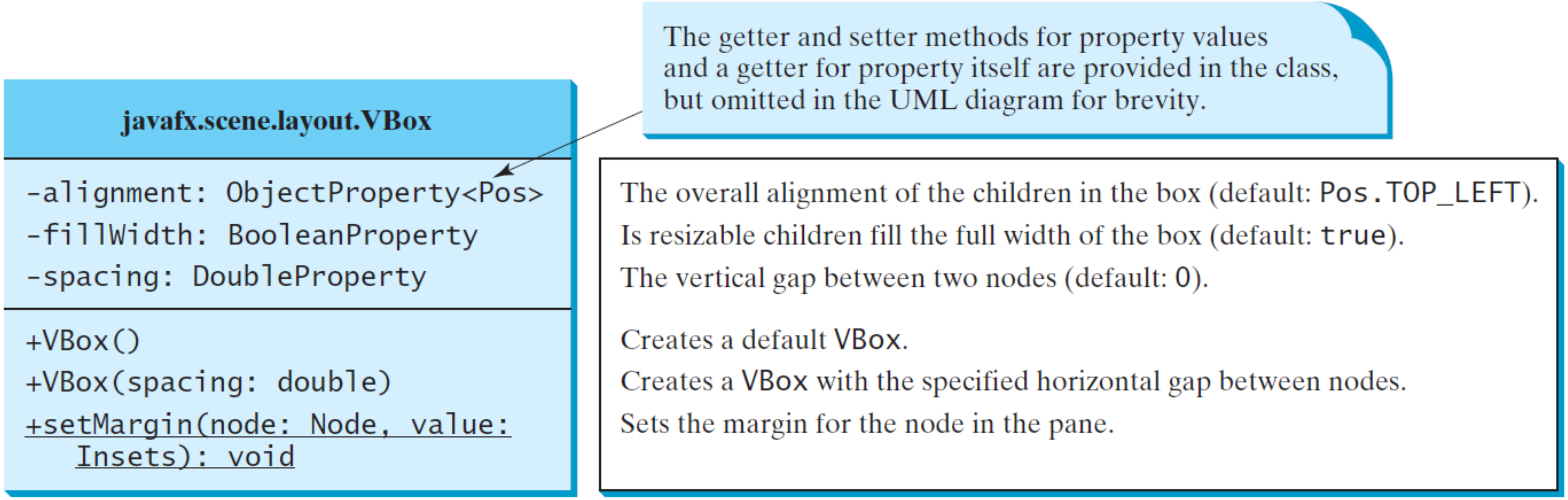
The horizontal gap between two nodes (default: 0).

Creates a default HBox.

Creates an HBox with the specified horizontal gap between nodes.

Sets the margin for the node in the pane.

VBox



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.VBox

```
-alignment: ObjectProperty<Pos>
-fillWidth: BooleanProperty
-spacing: DoubleProperty

+VBox()
+VBox(spacing: double)
+setMargin(node: Node, value:
   Insets): void
```

The overall alignment of the children in the box (default: Pos.TOP_LEFT).

Is resizable children fill the full width of the box (default: true).

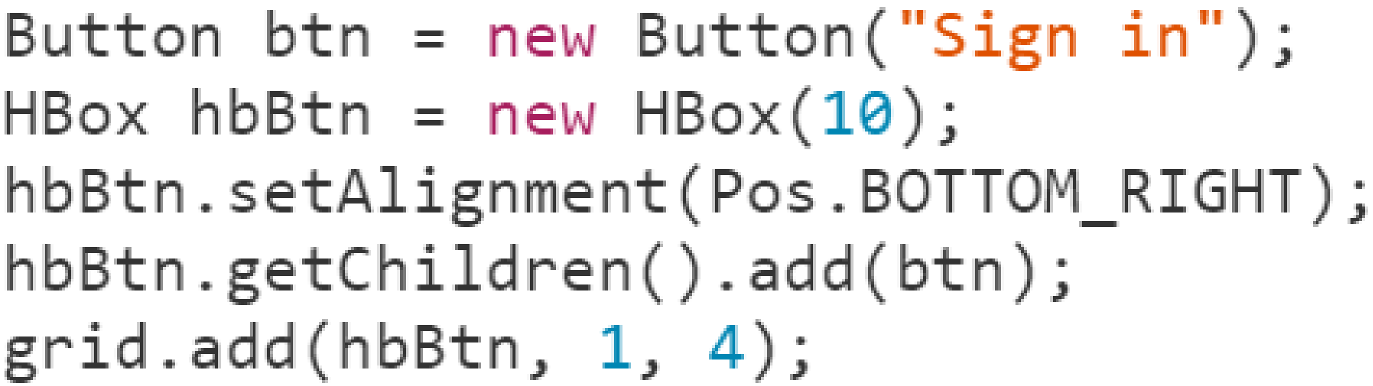
The vertical gap between two nodes (default: 0).

Creates a default VBox.

Creates a VBox with the specified horizontal gap between nodes.

Sets the margin for the node in the pane.

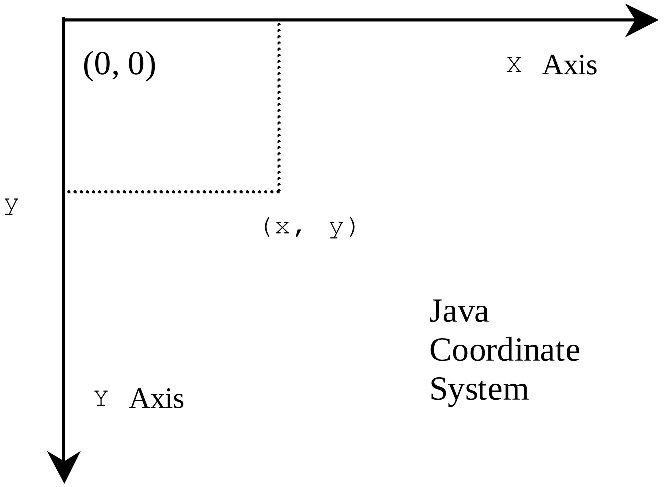
Hbox example



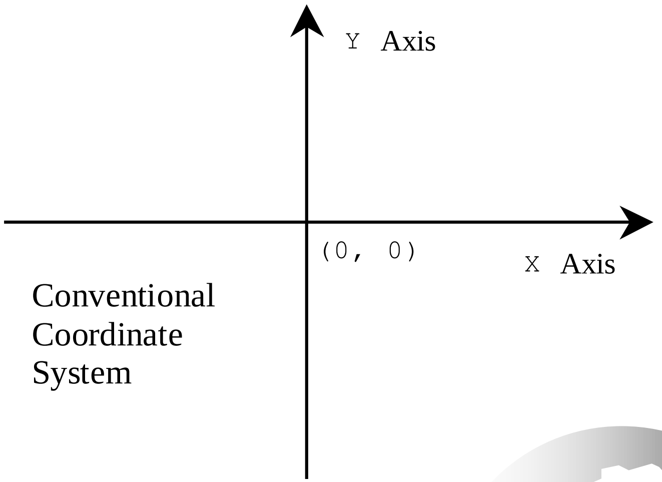
```
Button btn = new Button("Sign in");
HBox hbBtn = new HBox(10);
hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
hbBtn.getChildren().add(btn);
grid.add(hbBtn, 1, 4);
```

Display a Shape

This example displays a circle in the center of the pane.

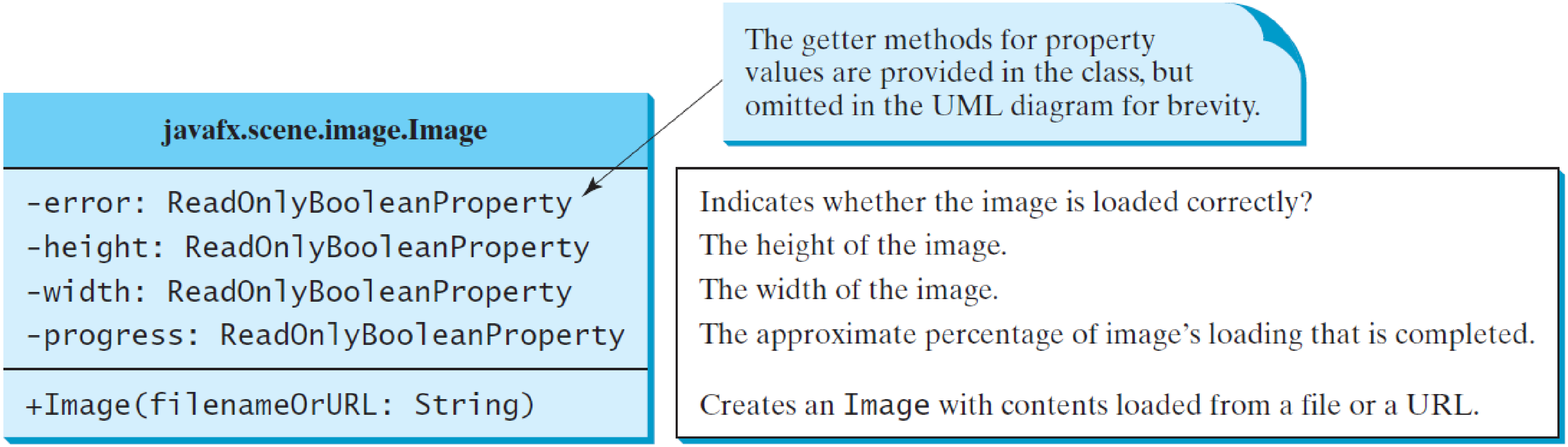


Java Coordinate System



Conventional Coordinate System

The Image Class



The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.image.Image

```
-error: ReadOnlyBooleanProperty
-height: ReadOnlyBooleanProperty
-width: ReadOnlyBooleanProperty
-progress: ReadOnlyBooleanProperty

+Image(filenameOrURL: String)
```

Indicates whether the image is loaded correctly?

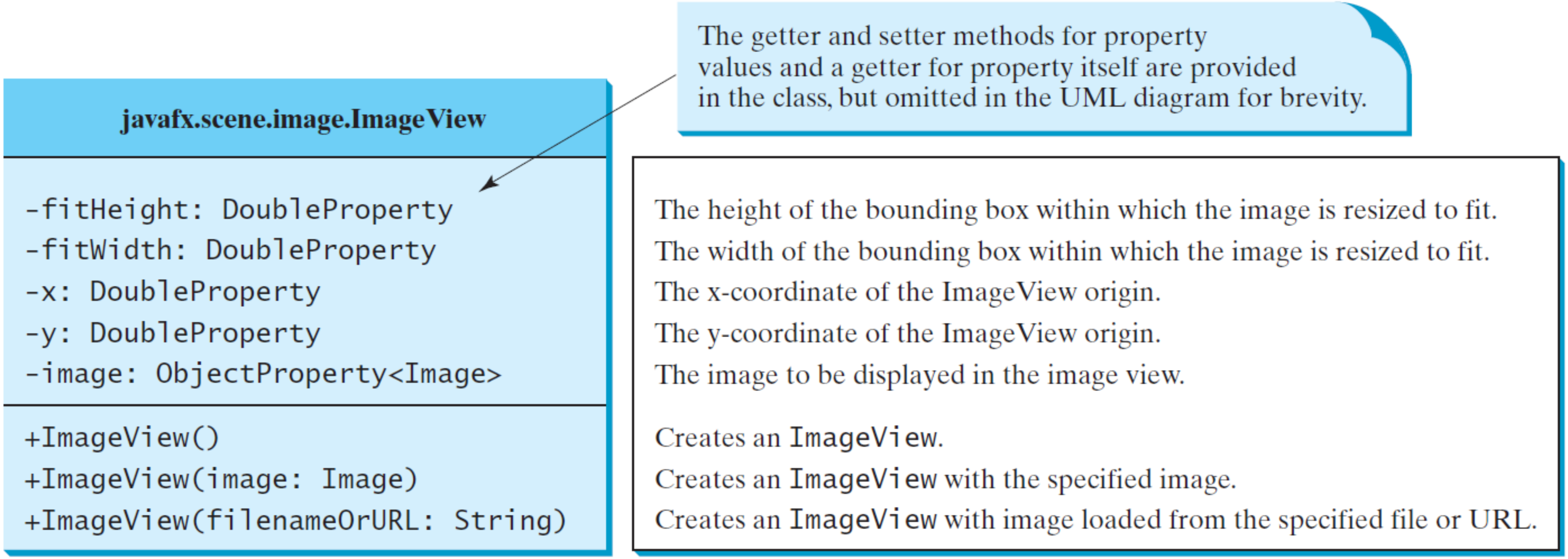
The height of the image.

The width of the image.

The approximate percentage of image's loading that is completed.

Creates an `Image` with contents loaded from a file or a URL.

The ImageView Class



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.image.ImageView

```
-fitHeight: DoubleProperty
-fitWidth: DoubleProperty
-x: DoubleProperty
-y: DoubleProperty
-image: ObjectProperty<Image>

+ImageView()
+ImageView(image: Image)
+ImageView(filenameOrURL: String)
```

The height of the bounding box within which the image is resized to fit.

The width of the bounding box within which the image is resized to fit.

The x-coordinate of the ImageView origin.

The y-coordinate of the ImageView origin.

The image to be displayed in the image view.

Creates an ImageView.

Creates an ImageView with the specified image.

Creates an ImageView with image loaded from the specified file or URL.

Exercise 21

- Display images
- Display random 0 or 1