- **Java CSS**

JavaFX cascading style sheets (CSS) can be used to specify styles for UI nodes. JavaFX cascading style sheets are based on CSS with some extensions. CSS defines the style for webpages. It separates the contents of webpages from its style. JavaFX CSS can be used to define the style for the UI and separates the contents of the UI from the style. You can define the look and feel of the UI in a JavaFX CSS file and use the style sheet to set the color, font, margin, and border of the UI components. A JavaFX CSS file makes it easy to modify the style without modifying the Java source code. A JavaFX style property is defined with a prefix **–fx-** to distinguish it from a property in CSS. All the available JavaFX properties are defined in http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html. Below is an example of style sheet.

```
.plaincircle {
-fx-fill: white; -fx-stroke: black;
}

.circleborder {
-fx-stroke-width: 5; -fx-stroke-dash-array: 12 2 4 2;
}

.border {
-fx-border-color: black; -fx-border-width: 5;
}

#redcircle {
-fx-fill: red; -fx-stroke: red;
}

#greencircle {
-fx-fill: green; -fx-stroke: green;
}
```

A style sheet uses the style class or style id to define styles. Multiple style classes can be applied to a single node, and a style id to a unique node. The syntax **.styleclass** defines a style class. In the above example, the style classes are named **plaincircle**, **circleborder**, and **border**. The syntax **#styleid** defines a style id. Here, the style ids are named **redcircle** and **greencircle**. Each node in JavaFX has a styleClass variable of the List type, which can be obtained from invoking **getStyleClass()**. You can add **multiple style classes to a node and only one id to a node**. Each node in JavaFX has an id variable of the String type, which can be set using the **setID(String id)** method. You can set only one id to a node. The Scene and Parent classes have the stylesheets property, which can be obtained from invoking the **getStylesheets()** method. This property is of the ObservableList type. You can add multiple style sheets into this property. You can load a style sheet into a Scene or a Parent. Note that Parent is the superclass for containers and UI control. An example is shown on the next page.

The program loads the style sheet from the file **mystyle.css** by adding it to the stylesheets property of the Scene object. The file should be **placed in the same directory with the source code** for it to run correctly. After the style sheet is loaded, the program sets the style class **plaincircle** for circle1 and circle2 and sets the style id **redcircle** for circle3. The program sets style classes **circleborder** and **plaincircle** and an id **greencircle** for circle4. The style class border is set for both pane1 and pane2. The style sheet is set in the scene, thus, all the nodes inside the scene can use this style sheet. Note the style class plaincircle and id greencircle both are applied to circle4. plaincircle sets fill to white and greencircle

sets fill to green. The **property settings in id take precedence** over the ones in classes. Thus, circle4 will be displayed in green in this program.

```
public class StyleSheetDemo extends Application {
    public void start(Stage primaryStage) {
        HBox hBox = new HBox(5);
        Scene scene = new Scene(hBox, 300, 250);
        scene.getStylesheets().add("mystyle.css"); // Load the stylesheet
        Pane pane1 = new Pane();
        Circle circle1 = new Circle(50, 50, 30);
        Circle circle2 = new Circle(150, 50, 30);
        Circle circle3 = new Circle(100, 100, 30);
        pane1.getChildren().addAll(circle1, circle2, circle3);
        pane1.getStyleClass().add("border");
        circle1.getStyleClass().add("plaincircle");
        circle2.getStyleClass().add("plaincircle");
        circle3.setId("redcircle");
        Pane pane2 = new Pane();
        Circle circle4 = new Circle(100, 100, 30);
        circle4.getStyleClass().addAll("circleborder", "plainCircle");
        circle4.setId("greencircle");
        pane2.getChildren().add(circle4);
        pane2.getStyleClass().add("border");
        hBox.getChildren().addAll(pane1, pane2);
        primaryStage.setTitle("StyleSheetDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

- **JavaFX Menu**

You can create menus in JavaFX. Menus make selection easier and are widely used in window applications. JavaFX provides five classes that implement menus: MenuBar, Menu, MenuItem, CheckMenuItem, and RadioButtonMenuItem.

MenuBar is a top-level menu component used to hold the menus. A menu consists of menu items that the user can select (or toggle on or off). A menu item can be an instance of MenuItem, CheckMenuItem, or RadioButtonMenuItem. Menu items can be associated with nodes and keyboard accelerators. The sequence of implementing menus in JavaFX is as follows:

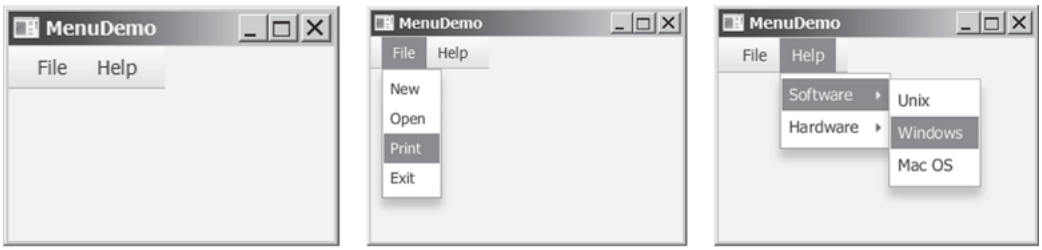
1. Create a menu bar and add it to a pane. For example, the following code creates a pane and a menu bar, and adds the menu bar to the pane:

```
MenuBar menuBar = new MenuBar();
Pane pane = new Pane();
pane.getChildren().add(menuBar);
```

2. Create menus and add them under the menu bar. For example, the following creates two menus and adds them to a menu bar,

```
Menu menuFile = new Menu("File");
Menu menuHelp = new Menu("Help");
menuBar.getMenus().addAll(menuFile, menuHelp);
Menu softwareHelpSubMenu = new Menu("Software");
Menu hardwareHelpSubMenu = new Menu("Hardware");
```

```
menuHelp.getItems().add(softwareHelpSubMenu);
menuHelp.getItems().add(hardwareHelpSubMenu);
softwareHelpSubMenu.getItems().add(new MenuItem("Unix"));
softwareHelpSubMenu.getItems().add(new MenuItem("Windows"));
softwareHelpSubMenu.getItems().add(new MenuItem("Mac OS"));
```



(a) (b) (c)

You can also add a CheckMenuItem to a Menu. CheckMenuItem is a subclass of MenuItem that adds a Boolean state to the MenuItem and displays a check when its state is true. For example,

```
menuHelp.getItems().add(new CheckMenuItem("Check it"));
```

The menu items generate ActionEvent. To handle ActionEvent, implement the setOn-Action method. The MenuItem has a graphic property for specifying a node to be displayed in the menu item. Usually, the graphic is an image view. The classes Menu, MenuItem, CheckMenuItem, and RadioMenuItem have another constructor that you can use to specify a graphic. For example,

```
Menu menuFile = new Menu("File", new ImageView("image/usIcon.gif"));
```