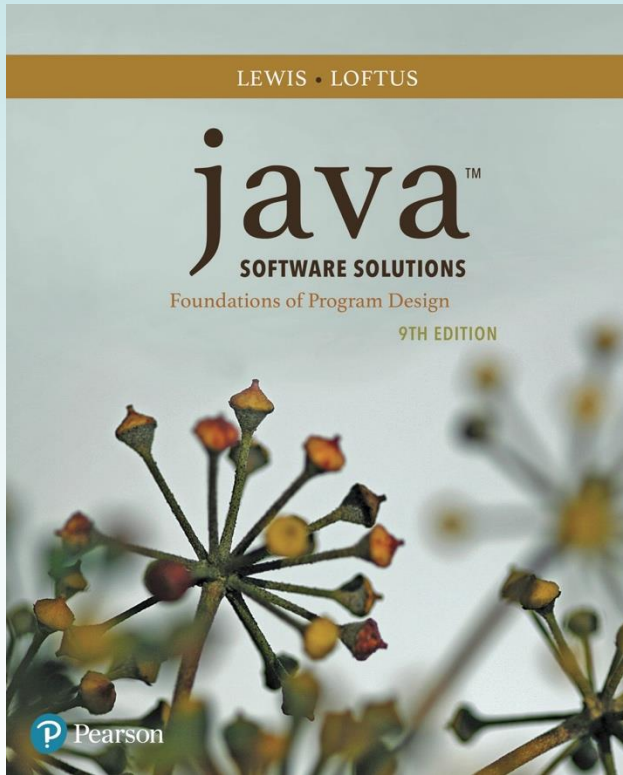# Chapter 3
Using Classes and Objects

Java Software Solutions

Foundations of Program Design

9th Edition

John Lewis
William Loftus

Using Classes and Objects

- We can create more interesting programs using predefined classes and related objects

- Chapter 3 focuses on:

  - object creation and object references
  - the `String` class and its methods
  - the Java API class library
  - Random and `Math` classes
  - formatting output
  - enumerated types
  - wrapper classes
  - JavaFX graphics API
  - shape classes

Outline

→ **Creating Objects**

The String Class

Modularity

Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

Introduction to JavaFX

Shapes and Color

Creating Objects

- A variable holds either a primitive value or a *reference* to an object

- A class name can be used as a type to declare an *object reference variable*

```
String title;
```

- No object is created with this declaration

- An object reference variable holds the address of an object

- The object itself must be created separately

Creating Objects

- Generally, we use the `new` operator to create an object

- Creating an object is called *instantiation*

- An object is an *instance* of a particular class

```
title = new String("Java Software Solutions");
```

This calls the String *constructor*, which is
a special method that sets up the object

Invoking Methods

- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
numChars = title.length()
```

- A method may *return a value*, which can be used in an assignment or expression

- A method invocation can be thought of as asking an object to perform a service

References

- Note that a primitive variable contains the value itself, but an object variable contains the address of the object

- An object reference can be thought of as a pointer to the location of the object

- Rather than dealing with arbitrary addresses, we often depict a reference graphically

num1   `38`

name1   ⬜ ⟶ `"Steve Jobs"`
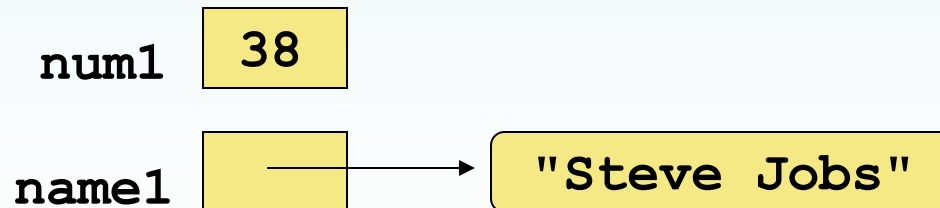
Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable

- For primitive types:

Before:

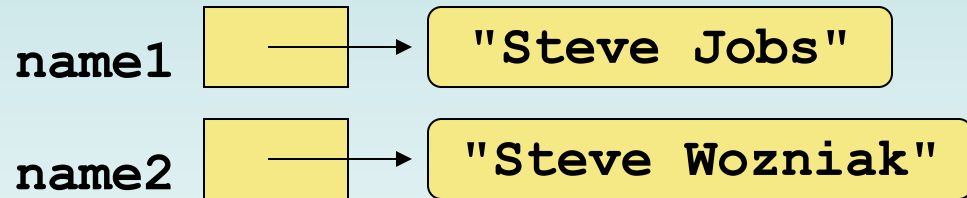num1 `38`

num2 `96`

`num2 = num1;`
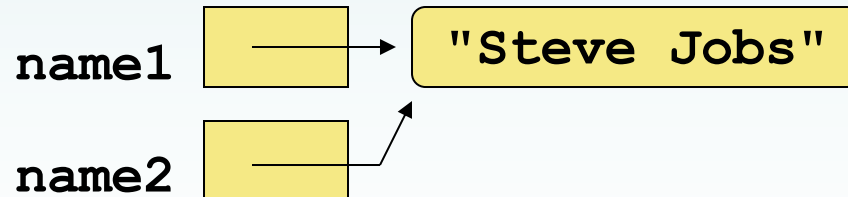
After:

num1 `38`

num2 `38`

Reference Assignment

- For object references, assignment copies the address:

Before:

name1 → "Steve Jobs"

name2 → "Steve Wozniak"

`name2 = name1;`

After:

name1 → "Steve Jobs"

name2 →

Aliases

- Two or more references that refer to the same object are called *aliases* of each other

- That creates an interesting situation: one object can be accessed using multiple reference variables

- Aliases can be useful, but should be managed carefully

- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program

- The object is useless, and therefore is called *garbage*

- Java performs *automatic garbage collection* periodically, returning an object's memory to the system for future use

- In other languages, the programmer is responsible for performing garbage collection

Outline

Creating Objects

→ The String Class

Modularity

Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

Introduction to JavaFX

Shapes and Color

The String Class

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

    `title = "Java Software Solutions";`

- This is special syntax that works <u>only</u> for strings

- Each string literal (enclosed in double quotes) represents a `String` object

- There is a special syntax for text blocks

  - Use """ before and after the text block
  - Common leading whitespace is removed

String Indexes

- It is occasionally helpful to refer to a particular character within a string

- This can be done by specifying the character's numeric *index*

- The indexes begin at zero in each string

- In the string `"Hello"`, the character `'H'` is at index 0 and the `'o'` is at index 4

- See `StringMutation.java`

String Methods

- Once a `String` object has been created, neither its value nor its length can be changed

- Therefore we say that an object of the `String` class is *immutable*

- However, several methods of the `String` class return new `String` objects that are modified versions of the original

Quick Check

What output is produced by the following?

```java
String str = "Space, the final frontier.";
System.out.println(str.length());
System.out.println(str.substring(7));
System.out.println(str.toUpperCase());
System.out.println(str.length());
```

Quick Check

What output is produced by the following?

```
String str = "Space, the final frontier.";
System.out.println(str.length());
System.out.println(str.substring(7));
System.out.println(str.toUpperCase());
System.out.println(str.length());
```

```
26
the final frontier.
SPACE, THE FINAL FRONTIER.
26
```

Outline

Creating Objects

The String Class

→ **Modularity**

Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

Introduction to JavaFX

Shapes and Color

Modularity

- To deal with complexity, language design has added many structures in higher level languages
  - Statements
  - Functions/methods/subroutines/procedures
  - Classes
  - Packages
  - Modules/Components/Subsystems
- Beyond statements, each structure defines a part of the system with a well-defined interface
  - Separates usage contract from implementation details
  - To merely use something we do not need to know how it is implemented
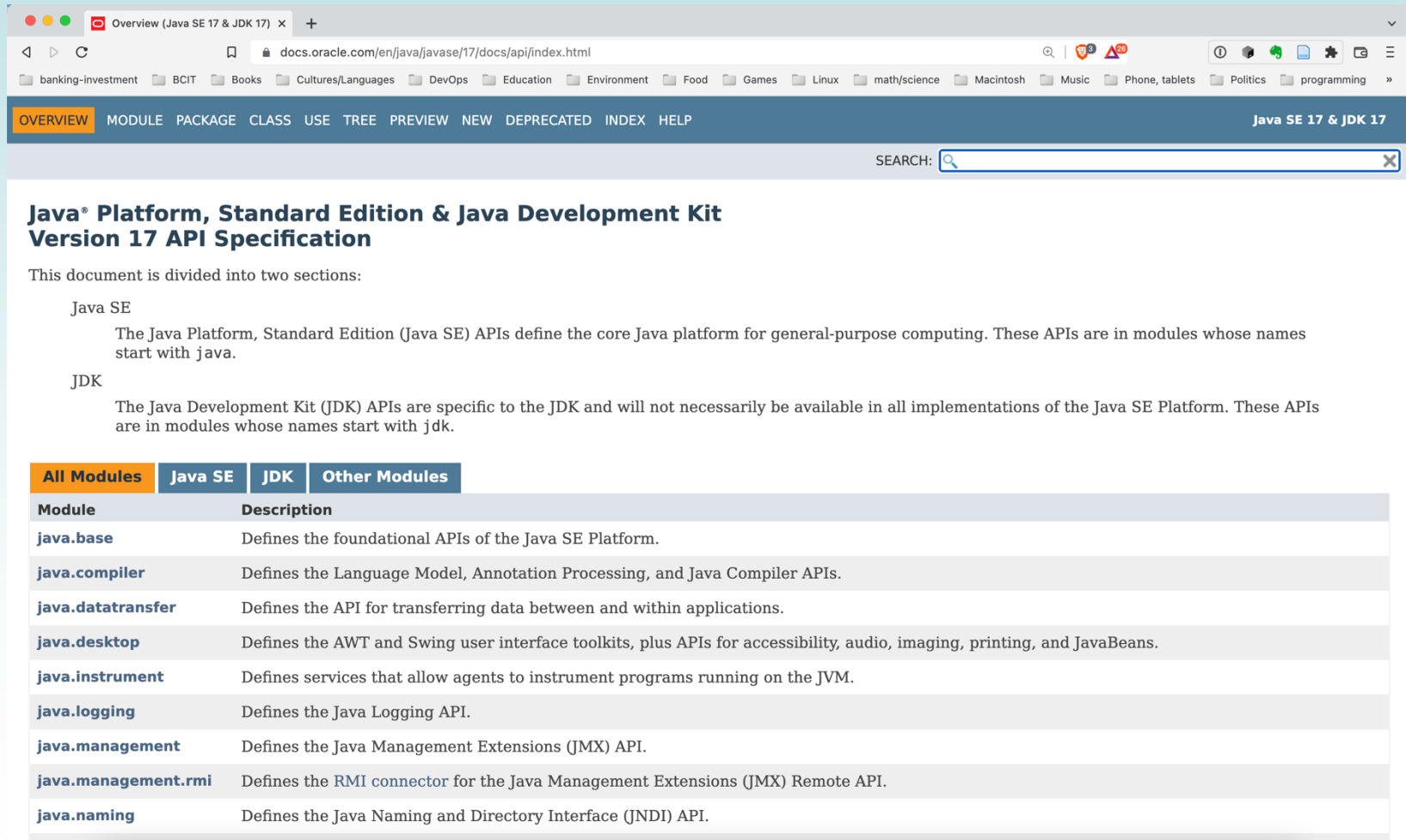
Class Libraries

- A *class library* is a collection of classes that we can use when developing programs

- The *Java standard class library* is part of any Java development environment

- Its classes are not part of the Java language per se, but we rely on them heavily

- Various classes we've already used (`System`, `Scanner`, `String`) are part of the Java standard class library

The Java API

- The Java class library is sometimes referred to as the Java API

- API stands for Application Programming Interface

- Clusters of related classes are sometimes referred to as specific APIs:

  – JavaFX API

  – Database API

  – Network API

  – Streams API

The Java API

- Get comfortable using the online Java API at
  https://docs.oracle.com/en/java/javase/21/docs/api/index.html

Packages

- For purposes of accessing them, classes in the Java API are organized into *packages*

- These often overlap with specific APIs

- Examples:

Package	Purpose
java.lang	General support
java.util	Utilities
java.net	Network communication
javafx.scene.shape	Graphical shapes
javafx.scene.control	GUI controls

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

- Or you can *import* the class, and then use just the class name

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can (but don't) use the * wildcard character

```
import java.util.*;
```

- All classes of the `java.lang` package are imported automatically into all programs

Defining Packages

- We have discussed classes from the standard java packages

- It is easy to define your own packages!

  - The first line in the source code file is a package statement

    - package mypackagename;

    - package q1;

    - package ca.bcit.infosys.servletutils;

  - The source code file must be in a directory (directories) corresponding to the package name

    - Replace period with directory separator

Assignment 1

- For assignments, each programming problem will have a separate package:
  - q1 for problem 1
  - q2 for problem 2
  - similarly q3, q4, q5 for problem 3, 4, 5
- We will also be giving you a template and ant script
  - to build and package your code for assignment submission
  - the lab instructor will walk you through the process

Java Platform Module System

- Partitions Java libraries into commonly used subsystems
- Allows large systems to be designed with reusable components
  - increases scalability
- Module contains
  - Collection of packages
  - Optional resource files and native libraries
  - List of accessible packages in the module
  - List of modules on which this module depends
- Defined by `module-info.java` in base directory

Basic Modules

- Module name: letters, digits, underscores, periods
  - No hierarchy is implied by the name
  - Name should be globally unique
  - Typically name after its highest level package
- Format of simple `module-info.java`:

```
module module.name {
    requires javafx.controls;
    …
    exports my.package;
    …
}
```

- No `module-info.java` → Non modular project

Outline

Random Classes

- Many classes to create random numbers are part of `java.util` and `java.util.random` packages

- These provide methods for pseudorandom numbers, defined in `RandomGenerator`

- See `java.util.random` package API discussion

- Objects from classes implementing `RandomGenerator` perform calculations to produce a stream of seemingly random values

- See `RandomNumbers.java`

Quick Check

Given a `RandomGenerator` object named `gen`, what range of values are produced by the following expressions?

```
gen.nextInt(25)

gen.nextInt(6) + 1

gen.nextInt(100) + 10

gen.nextInt(50) + 100

gen.nextInt(10) - 5

gen.nextInt(22) + 12
```

Quick Check

Given a `RandomGenerator` object named `gen`, what range of values are produced by the following expressions?

	Range
`gen.nextInt(25)`	`0 to 24`
`gen.nextInt(6) + 1`	`1 to 6`
`gen.nextInt(100) + 10`	`10 to 109`
`gen.nextInt(50) + 100`	`100 to 149`
`gen.nextInt(10) - 5`	`-5 to 4`
`gen.nextInt(22) + 12`	`12 to 33`

Quick Check

Write an expression that produces a random integer in the following ranges:

Range

`0 to 12`

`1 to 20`

`15 to 20`

`-10 to 0`

Quick Check

Write an expression that produces a random integer in the following ranges:

Range	
Range	
0 to 12	`gen.nextInt(13)`
1 to 20	`gen.nextInt(20) + 1`
15 to 20	`gen.nextInt(6) + 15`
-10 to 0	`gen.nextInt(11) - 10`

The Math Class

- The `Math` class is part of the `java.lang` package

- The `Math` class contains methods that perform various mathematical functions

- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

The Math Class

- The methods of the `Math` class are *static methods* (also called *class methods*)

- Static methods are invoked through the class name – no object of the `Math` class is needed

    ```
    value = Math.cos(90) + Math.sqrt(delta);
    ```

- We discuss static methods further in Chapter 7
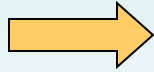
- See `Quadratic.java`

Outline

Creating Objects

The String Class

Modularity

Random and Math Classes

➡ **Formatting Output**

Enumerated Types

Wrapper Classes

Introduction to JavaFX

Shapes and Color

Formatting Output

- It is often necessary to format output values in certain ways so that they can be presented properly

- The Java standard class library contains classes that provide formatting capabilities

- The `NumberFormat` class allows you to format values as currency or percentages

- The `DecimalFormat` class allows you to format values based on a pattern

- Both are part of the `java.text` package

Formatting Output

- The `NumberFormat` class has static methods that return a formatter object

$$getCurrencyInstance()$$

$$getPercentInstance()$$

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format

- See `Purchase.java`

Formatting Output

- The `DecimalFormat` class can be used to format a floating point value in various ways

- For example, you can specify that the number should be truncated to three decimal places

- The constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number

- See `CircleStats.java`

Outline

Enumerated Types

- Java allows you to define an *enumerated type*, which can then be used to declare variables

- An enumerated type declaration lists all possible values for a variable of that type

- The values are identifiers of your own choosing

- The following declaration creates an enumerated type called `Season`

  ```
  enum Season {winter, spring, summer, fall};
  ```

- Any number of values can be listed

Enumerated Types

- Once a type is defined, a variable of that type can be declared:

$$Season\ time;$$

- And it can be assigned a value:

$$time = Season.fall;$$

- The values are referenced through the name of the type

- Enumerated types are *type-safe* – you cannot assign any value other than those listed

Ordinal Values

- Enumerated types are stored in variables as references, like all objects

- All references for a given value are aliases, values are immutable

- Each value has a name and an ordinal

  - The first value in an enumerated type has an ordinal value of zero, the second one, and so on

- You cannot assign a numeric value to an enumerated type variable

Enumerated Types

- The declaration of an enumerated type is a special type of class, and each variable of that type is an object

- The `ordinal` method returns the ordinal value of the object

- The `name` method returns the name of the identifier corresponding to the object's value

- See `IceCream.java`

Outline

Wrapper Classes

- The `java.lang` package contains *wrapper classes* that correspond to each primitive type:

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Wrapper Classes

- The following declaration creates an `Integer` object which represents the integer 40 as an object

  ```
  Integer age = new Integer(40);
  ```

- An object of a wrapper class can be used in any situation where a primitive value will not suffice

  - For example, some objects serve as containers of other objects

  - Primitive values could not be stored in such containers, but wrapper objects could be

- Values of wrapper classes are immutable

Wrapper Classes

- Wrapper classes also contain static methods that help manage the associated type

- For example, the `Integer` class contains a method to convert an integer stored in a `String` to an `int` value:

  ```
  num = Integer.parseInt(str);
  ```

- They often contain useful constants as well

- For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values

Autoboxing

- *Autoboxing* is the automatic conversion of a primitive value to/from a corresponding wrapper object:

```
Integer obj;
int num = 42;
obj = num;
```

- The assignment creates the appropriate `Integer` object (*boxing*)

- The reverse conversion (called *unboxing*) also occurs automatically as needed

Quick Check

Are the following assignments valid? Explain.

```
Double value = 15.75;



Character ch = new Character('T');
char myChar = ch;
```

Quick Check

Are the following assignments valid? Explain.

```
Double value = 15.75;
```

Yes. The double literal is boxed into a `Double` object.

```
Character ch = new Character('T');
char myChar = ch;
```

Yes, the char in the object is unboxed before the assignment.

Outline

Creating Objects

The String Class

Modularity

Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

→ **Introduction to JavaFX**

Shapes and Color

Intro to JavaFX

- The programs we've explored thus far have been text-based

- They are called *command-line applications*, which interact with the user using simple text prompts

- We'll now begin to explore programs that use graphics and graphical user interfaces (GUIs)

- Support for these programs will come from the JavaFX API

- JavaFX has replaced older approaches (AWT and Swing)

JavaFX Module Structure
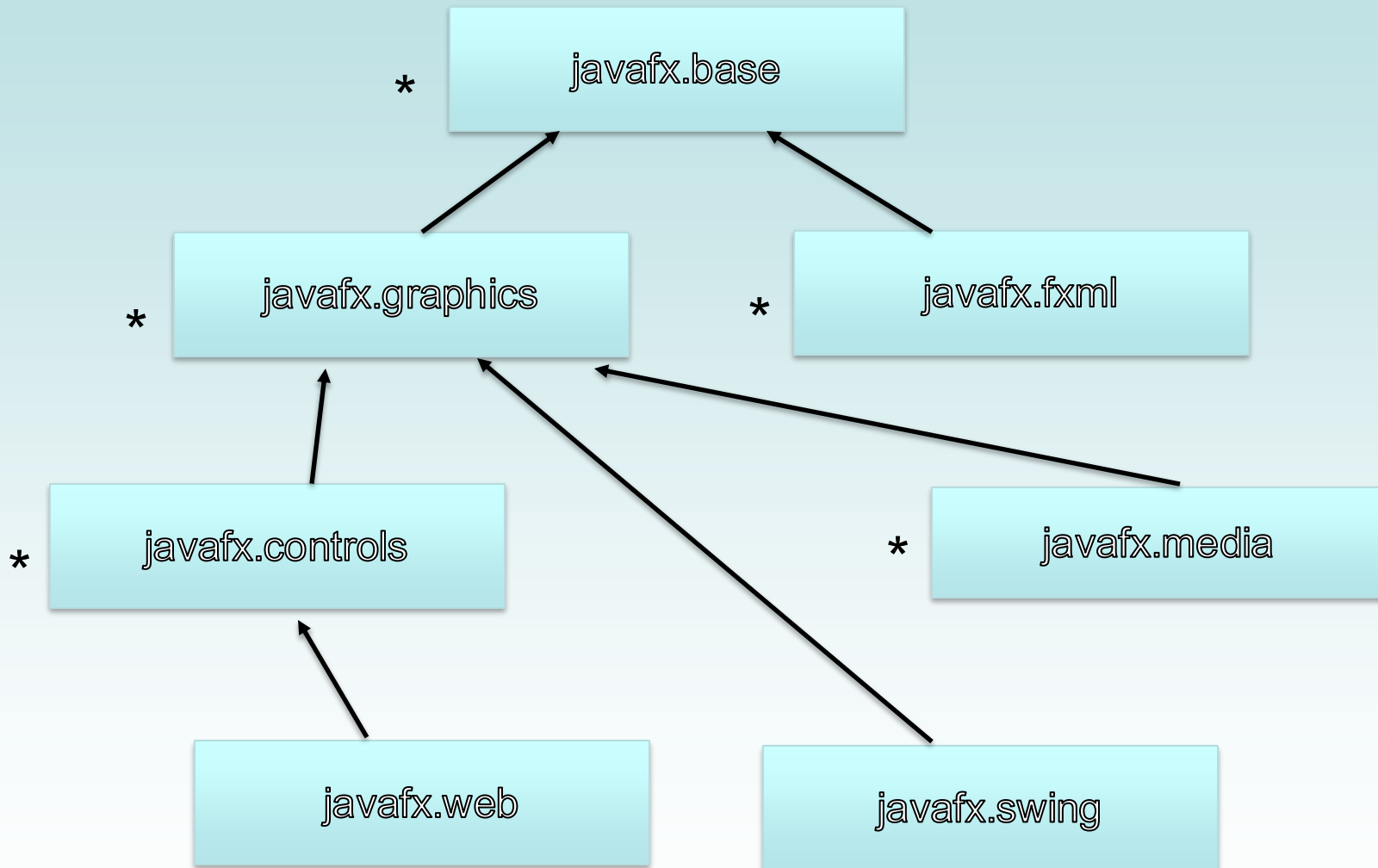
Intro to JavaFX

- JavaFX programs extend the `Application` class, inheriting core graphical functionality

- A JavaFX program has a `start` method

- The `main` method is only needed to launch the JavaFX application

- The `start` method accepts the primary stage (window) used by the program as a parameter

- JavaFX embraces a theatre analogy

- See `HelloJavaFX.java`

Intro to JavaFX

- In this example, two `Text` objects are added to a `Group`

- The group serves as the *root node* of a `Scene`

- The scene is displayed on the primary `Stage` (window)

- The size and background color of the scene can be set when the `Scene` object is created

- The position of each `Text` object is specified explicitly (in this case)

Intro to JavaFX

- The origin of the Java coordinate system is in the upper left corner

- All visible points have positive coordinates
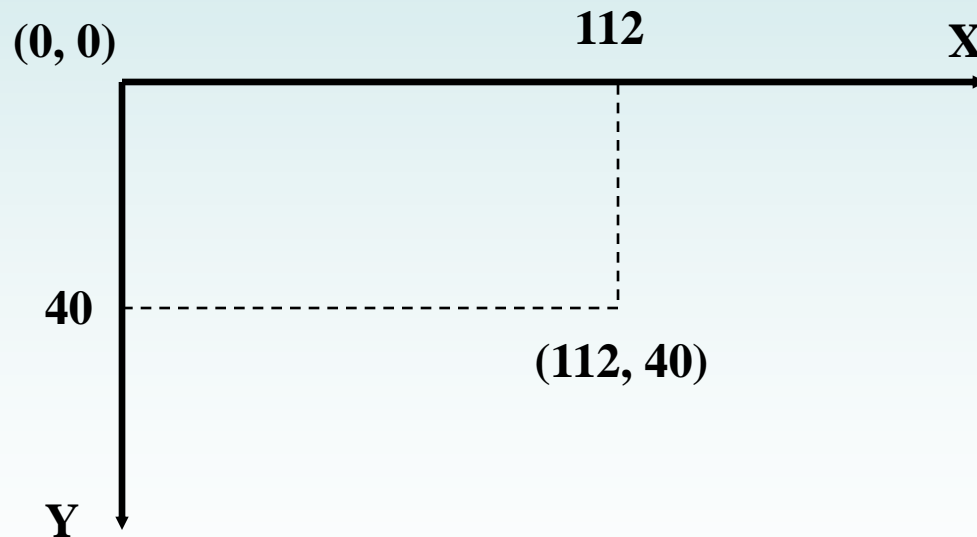
Outline

Creating Objects

The String Class

Modularity

Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

Introduction to JavaFX

→ **Shapes and Color**

Basic Shapes

- JavaFX shapes are represented by classes in the javafx.scene.shape package

- A line segment is defined by the `Line` class, whose constructor accepts the coordinates of the two endpoints:

```
Line(startX, startY, endX, endY)
```

- For example:

```
Line myLine = new Line(10, 20, 300, 80);
```

Basic Shapes

- A rectangle is specified by its upper left corner and its width and height:

$$Rectangle(x, y, width, height)$$

```
Rectangle r = new Rectangle(30, 50, 200, 70);
```

- A circle is specified by its center point and radius:

$$Circle(centerX, centerY, radius)$$

```
Circle c = new Circle(100, 150, 40);
```

Basic Shapes

- An ellipse is specified by its center point and its radius along the x and y axis:

```
Ellipse(centerX, centerY, radiusX, radiusY)

Ellipse e = new Ellipse(100, 50, 80, 30);
```

- Shapes are drawn in the order in which they are added to the group

- The stroke and fill of each shape can be set
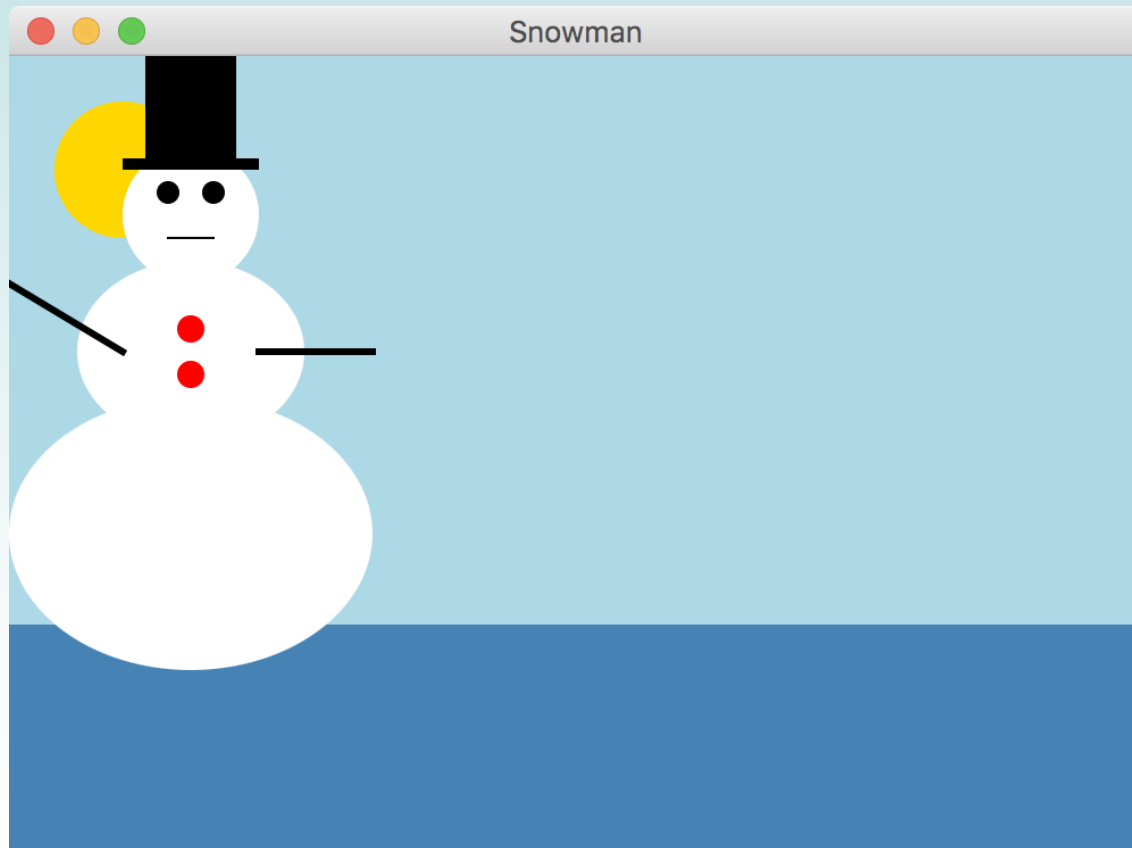
- See `Einstein.java`

Basic Shapes

- Groups can be nested within groups

- *Translating* a shape or group shifts its position along the x or y axis

- A shape or group can be rotated using the `setRotate` method

- See `Snowman.java`
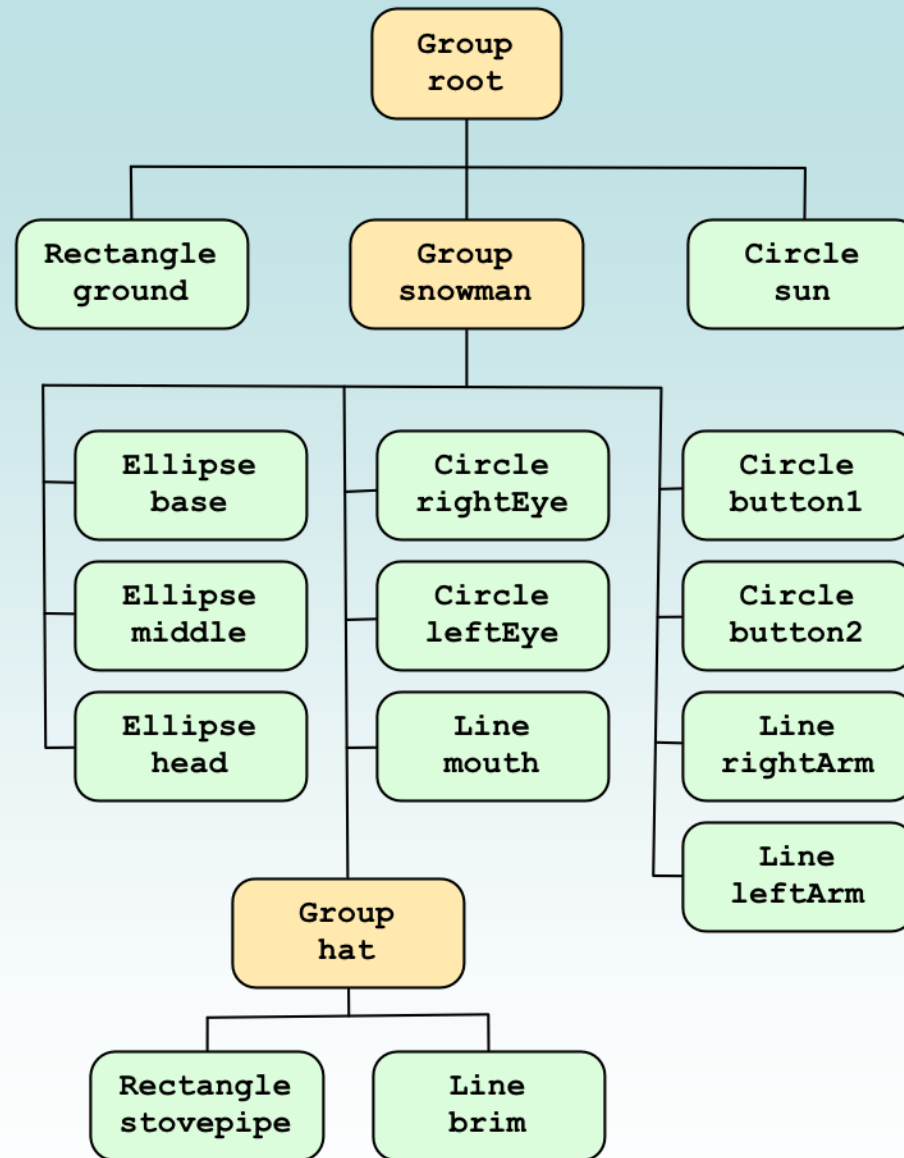
Basic Shapes

- Without translating (shifting) the snowman's position:

Representing Color

- A color in Java is represented by a `Color` object

- A color object holds three numbers called an RGB value, which stands for Red-Green-Blue

- Each number represents the contribution of that color

- This is how the human eye works

- Each number in an RGB value is in the range 0 to 255

Representing Color

- A color with an RGB value of 255, 255, 0 has a full contribution of red and green, but no blue, which is a shade of yellow

- The static `rgb` method in the `Color` class returns a `Color` object with a specific RGB value:

  ```
  Color purple = Color.rgb(183, 44, 150);
  ```

- The `color` method uses percentages:

  ```
  Color maroon = Color.color(0.6, 0.1, 0.0);
  ```

Representing Color

- For convenience, several `Color` objects have been predefined, such as:

`Color.BLACK`	0, 0, 0
`Color.WHITE`	255, 255, 255
`Color.CYAN`	0, 255, 255
`Color.PINK`	255, 192, 203
`Color.GRAY`	128, 128, 128

- See the online documentation of the `Color` class for a full list of predefined colors

Summary

- Chapter 3 focused on:

  - object creation and object references
  - the `String` class and its methods
  - the Java standard class library
  - Random and `Math` classes
  - formatting output
  - enumerated types
  - wrapper classes
  - JavaFX graphics API
  - shape classes