Chapter 3
Using Classes and Objects

- Chapter 3 focuses on:
  - Object creation and object references
  - String class and its methods
  - Java API class library
  - Random and Math classes
  - Formatting output
  - Enumerated types
  - Wrapper classes
  - Graphical components and containers
  - Labels and images

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;
- An object variable holds the address of an object
- The object itself must be created separately
- 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

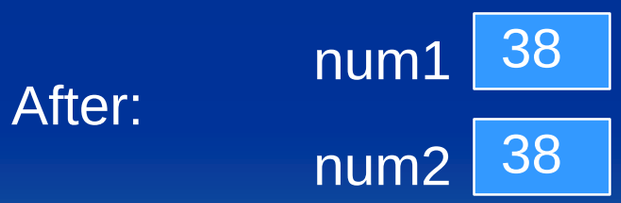
References

- Primitive variable contains the value itself
- 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

Assignment Revisited

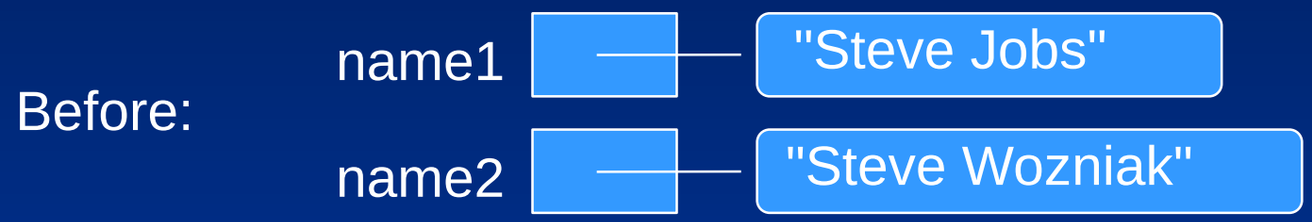
- The act of assignment takes a copy of a value and stores it in a variable
- For primitive types:

num2 = num1;

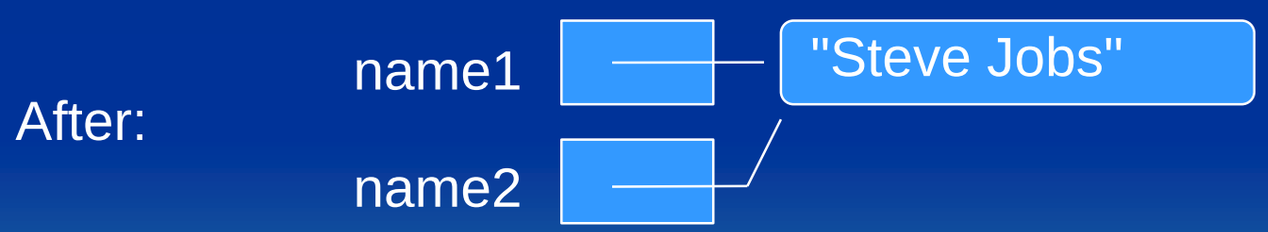


Reference Assignment

- For object references, assignment copies the address:



name2 = name1;



Aliases

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

- 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

The String Class and String Methods

- We don't have to use the new operator to create a String object
  - title = "Java Software Solutions";
- This is special syntax that works **only** for strings
- Each string literal (enclosed in double quotes) represents a String object
- 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

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

A_StringMutation.java

```
package chapter_03;

public class A_StringMutation
{
    public static void main(String[] args)
    {
        String phrase = "Change is inevitable";
        String mutation1, mutation2, mutation3, mutation4;

        System.out.println("Original string: \"" + phrase + "\"");
        System.out.println("Length of string: " + phrase.length());

        mutation1 = phrase.concat(", except from vending machines.");
        mutation2 = mutation1.toUpperCase();
        mutation3 = mutation2.replace('E', 'X');
        mutation4 = mutation3.substring(3, 30);

        // Print each mutated string
        System.out.println("Mutation #1: " + mutation1);
        System.out.println("Mutation #2: " + mutation2);
        System.out.println("Mutation #3: " + mutation3);
        System.out.println("Mutation #4: " + mutation4);

        System.out.println("Mutated length: " + mutation4.length());
    }
}
```

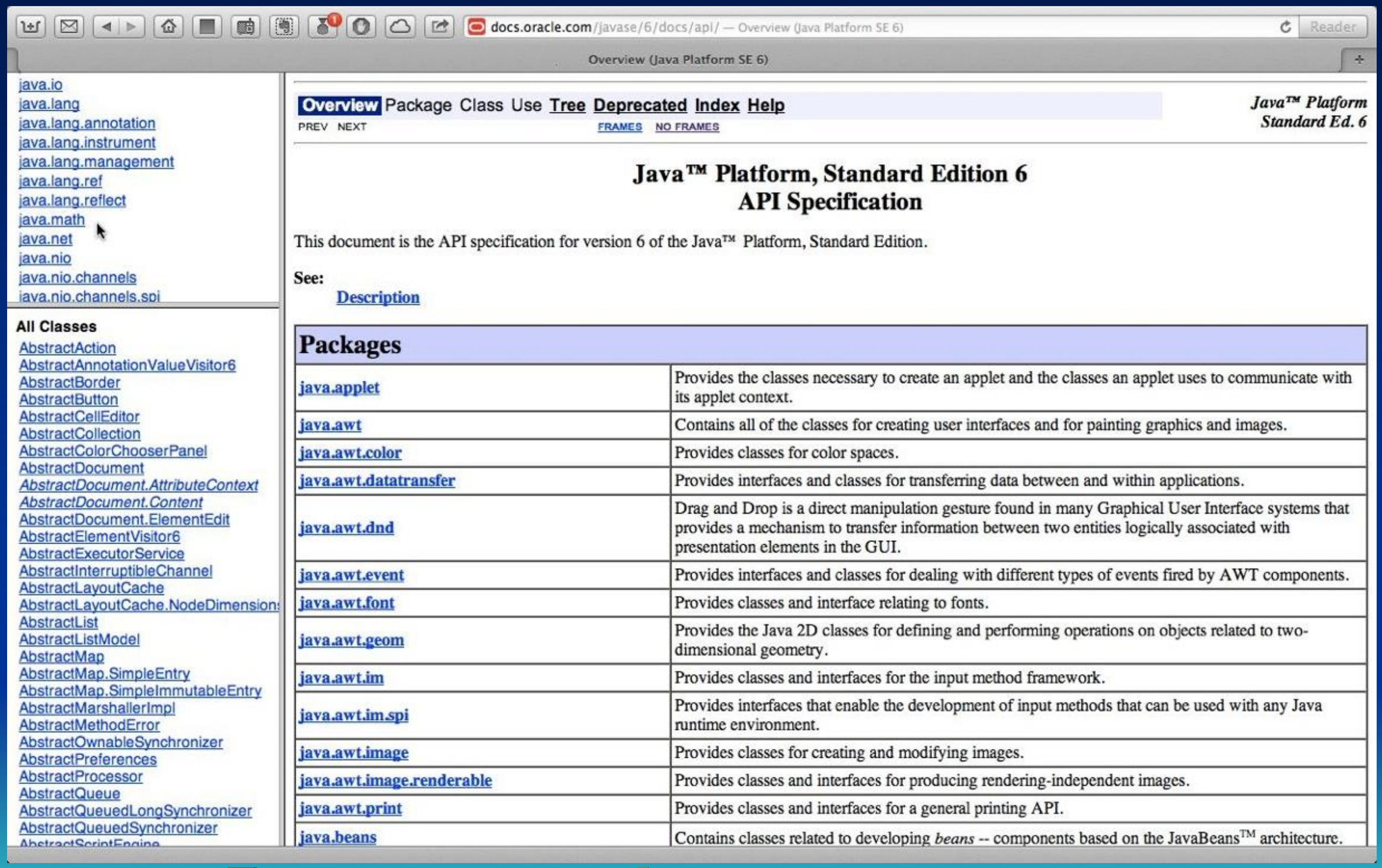
In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4

```
<terminated> A_StringMutation [Java Application] /System/Library/Java/JavaVirtualMac
Original string: "Change is inevitable"
Length of string: 20
Mutation #1: Change is inevitable, except from vending machines.
Mutation #2: CHANGE IS INEVITABLE, EXCEPT FROM VENDING MACHINES.
Mutation #3: CHANGX IS INXVITABLX, XXCXPT FROM VXNDING MACHINXS.
Mutation #4: NGX IS INXVITABLX, XXCXPT F
Mutated length: 27
```

Class Libraries and The Java API

- 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
- Various classes we've already used (System, Scanner, String) are part of the Java standard class library
- 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:
  - The Swing API
  - The Database API

http://docs.oracle.com/javase/6/docs/api/



Packages

- Classes in the Java API are organized into packages.
- There is a general correspondence between packages and API names.

Package	Provides support to
java.applet	Create programs (applets) that are easily transported across the Web.
java.awt	Draw graphics and create graphical user interfaces; AWT stands for Abstract Windowing Toolkit.
java.beans	Define software components that can be easily combined into applications.
java.io	Perform a wide variety of input and output functions.
java.lang	General support; it is automatically imported into all Java programs.
java.math	Perform calculations with arbitrarily high precision.
java.net	Communicate across a network.
java.rmi	Create programs that can be distributed across multiple computers; RMI stands for Remote Method Invocation.
java.security	Enforce security restrictions.
java.sql	Interact with databases; SQL stands for Structured Query Language.
java.text	Format text for output.
java.util	General utilities.
javax.swing	Create graphical user interfaces with components that extend the AWT capabilities.
javax.xml.parsers	Process XML documents; XML stands for eXtensible Markup Language.

FIGURE 3.2 Some packages in the Java standard class library

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 use the * wildcard character
  - import java.util.*;
- All classes of the java.lang package are imported automatically into all programs. It's as if all programs contain the following line:
  - import java.lang.*;
- That's why we didn't have to import the System or String classes explicitly in earlier programs
- The Scanner class, on the other hand, is part of the java.util package, and therefore must be imported

The Random Class

- The Random class is part of the java.util package
- It provides methods that generate pseudorandom numbers
- A Random object performs complicated calculations based on a seed value to produce a stream of seemingly random values

A_StringMutation.java | B_RandomNumbers.java

```
package chapter_03;

import java.util.Random;

public class B_RandomNumbers
{
    public static void main(String[] args)
    {
        Random generator = new Random();
        int num1;
        float num2;

        num1 = generator.nextInt();
        System.out.println("A random integer: " + num1);

        num1 = generator.nextInt(10);
        System.out.println("From 0 to 9: " + num1);

        num1 = generator.nextInt(10) + 1;
        System.out.println("From 1 to 10: " + num1);

        num1 = generator.nextInt(15) + 20;
        System.out.println("From 20 to 34: " + num1);

        num1 = generator.nextInt(20) - 10;
        System.out.println("From -10 to 9: " + num1);

        num2 = generator.nextFloat();
        System.out.println("A random float (between 0-1): " + num2);

        num2 = generator.nextFloat() * 6; // 0.0 to 5.999999
        num1 = (int) num2 + 1;
        System.out.println("From 1 to 6: " + num1);
    }
}
```

int java.util.Random.nextInt(int n)

nextInt

```
public int nextInt(int n)
```

Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. The general contract of `nextInt` is that one `int` value in the specified range is pseudorandomly generated and returned. All `n` possible `int` values are produced with (approximately) equal probability. The method `nextInt(int n)` is implemented by class `Random` as if by:

```
<terminated> B_RandomNumbers [Java Application] /S
A random integer: 2015613059
From 0 to 9: 8
From 1 to 10: 5
From 20 to 34: 22
From -10 to 9: -2
A random float (between 0-1): 0.38228464
From 1 to 6: 4
```

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 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 6

The Quadratic Equation

A_StringMutation.java | B_RandomNumbers.java | C_Quadratic.java

```java
package chapter_03;

import java.util.Scanner;

public class C_Quadratic
{
    public static void main(String[] args)
    {
        int a, b, c; // ax^2 + bx + c
        double discriminant, root1, root2;
        Scanner scan = new Scanner(System.in);

        System.out.print("Enter the coefficient of x squared: ");
        a = scan.nextInt();

        System.out.print("Enter the coefficient of x: ");
        b = scan.nextInt();

        System.out.print("Enter the constant: ");
        c = scan.nextInt();

        // Use the quadratic formula to compute the roots.
        // Assumes a positive discriminant.

        discriminant = Math.pow(b, 2) - (4 * a * c);
        root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
        root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);

        System.out.println("Root #1: " + root1);
        System.out.println("Root #2: " + root2);
    }
}
```

```
<terminated> C_Quadratic [Java Application] /System/
Enter the coefficient of x squared: 3
Enter the coefficient of x: 8
Enter the constant: 4
Root #1: -0.6666666666666666
Root #2: -2.0
```

```
<terminated> C_Quadratic [Java Application] /System/
Enter the coefficient of x squared: 1
Enter the coefficient of x: 1
Enter the constant: 1
Root #1: NaN
Root #2: NaN
```

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
  - NumberFormat class allows you to format values as currency or percentages
  - DecimalFormat class allows you to format values based on a pattern
- Both are part of the java.text package
- 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

Purchase.java Example

```java
package chapter_03;

import java.text.NumberFormat;
import java.util.Scanner;

public class D_Purchase
{
    public static void main(String[] args)
    {
        final double TAX_RATE = 0.06; // 6% sales tax

        int quantity;
        double subtotal, tax, totalCost, unitPrice;

        Scanner scan = new Scanner(System.in);

        NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
        NumberFormat fmt2 = NumberFormat.getPercentInstance();

        System.out.print("Enter the quantity: ");
        quantity = scan.nextInt();

        System.out.print("Enter the unit price: ");
        unitPrice = scan.nextDouble();

        subtotal = quantity * unitPrice;
        tax = subtotal * TAX_RATE;
        totalCost = subtotal + tax;

        // Print output with appropriate formatting
        System.out.println("Subtotal: " + fmt1.format(subtotal));
        System.out.println("Tax: " + fmt1.format(tax) + " at " + fmt2.format(TAX_RATE));
        System.out.println("Total: " + fmt1.format(totalCost));
    }
}
```

```
<terminated> D_Purchase [Java Applicati
Enter the quantity: 4
Enter the unit price: 32
Subtotal: $128.00
Tax: $7.68 at 6%
Total: $135.68
```

Formatting Output (cont.)

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

```
package chapter_03;

import java.text.DecimalFormat;
import java.util.Scanner;

public class E_CircleStats
{
    public static void main(String[] args)
    {
        int radius;
        double area, circumference;

        Scanner scan = new Scanner(System.in);

        System.out.print("Enter the circle's radius: ");
        radius = scan.nextInt();

        area = Math.PI * Math.pow(radius, 2);
        circumference = 2 * Math.PI * radius;

        // Round the output to three decimal places
        DecimalFormat fmt = new DecimalFormat("0.###");

        System.out.println("The circle's area: " + fmt.format(area));
        System.out.println("The circle's circumference: " + fmt.format(circumference));
    }
}
```

```
<terminated> E_CircleStats [Java Application] /S
Enter the circle's radius: 7
The circle's area: 153.938
The circle's circumference: 43.982
```

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
- 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

- Internally, each value of an enumerated type is stored as an integer, called its ordinal value
- The first value in an enumerated type has an ordinal value of zero, the second one, and so on
- However, you cannot assign a numeric value to an enumerated type, even if it corresponds to a valid ordinal value
- 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

Ice Cream

```java
package chapter_03;

public class F_IceCream
{
    enum Flavor
    {
        vanilla, chocolate, strawberry, fudgeRipple, coffee, rockyRoad, mintChocolateChip, cookieDough
    }

    // ------------------------------------------------------------------
    // Creates and uses variables of the Flavor type.
    // ------------------------------------------------------------------
    public static void main(String[] args)
    {
        Flavor cone1, cone2, cone3;

        cone1 = Flavor.rockyRoad;
        cone2 = Flavor.chocolate;

        System.out.println("cone1 value: " + cone1);
        System.out.println("cone1 ordinal: " + cone1.ordinal());
        System.out.println("cone1 name: " + cone1.name());

        System.out.println();
        System.out.println("cone2 value: " + cone2);
        System.out.println("cone2 ordinal: " + cone2.ordinal());
        System.out.println("cone2 name: " + cone2.name());

        cone3 = cone1;

        System.out.println();
        System.out.println("cone3 value: " + cone3);
        System.out.println("cone3 ordinal: " + cone3.ordinal());
        System.out.println("cone3 name: " + cone3.name());
    }
}
```

Problems @ Javadoc

```
<terminated> F_IceCream [Java A
cone1 value: rockyRoad
cone1 ordinal: 5
cone1 name: rockyRoad

cone2 value: chocolate
cone2 ordinal: 1
cone2 name: chocolate

cone3 value: rockyRoad
cone3 ordinal: 5
cone3 name: rockyRoad
```

Wrapper Classes

- The java.lang package contains wrapper classes that correspond to each primitive type
- 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 (Vectors of integers)

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

Wrapper Classes (cont.)

- 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

```
Integer (int value)
  Constructor: creates a new Integer object storing the specified value.

byte byteValue ()
double doubleValue ()
float floatValue ()
int intValue ()
long longValue ()
  Return the value of this Integer as the corresponding primitive type.

static int parseInt (String str)
  Returns the int corresponding to the value stored in the
  specified string.

static String toBinaryString (int num)
static String tohexString (int num)
static String toOctalString (int num)
  Returns a string representation of the specified integer value in the
  corresponding base.
```

Integer Wrapper Example

G_MyIntegerWrapper.java

```
package chapter_03;

public class G_MyIntegerWrapper
{

    public static void main(String[] args)
    {
        Integer age = new Integer(40);
        System.out.println("My Integer age is "+age.toString());

        int myAge = Integer.parseInt("46");
        System.out.println("My int age is " + myAge);

        Integer obj1;
        int num1 = 69;
        obj1 = num1;
        System.out.println("Autoboxing: My obj1 "+obj1.toString());

        Integer obj2 = new Integer(96);
        int num2;
        num2 = obj2;
        System.out.println("Unboxing: My num2 "+num2);

        System.out.println("Maximum int "+Integer.MAX_VALUE);
    }
}
```

Problems @ Javadoc Declaration

<terminated> G_MyIntegerWrapper [Java Appl

```
My Integer age is 40
My int age is 46
Autoboxing: My obj1 69
Unboxing: My num2 96
Maximum int 2147483647
```

Graphical Applications and GUI Components

- So far we used command-line applications, which interact with the user using simple text prompts
- Java applications that have graphical components will serve as a foundation to programs that have true graphical user interfaces (GUIs)
- A GUI component is an object that represents a screen element such as a button or a text field
- GUI-related classes are defined primarily in the java.awt and the javax.swing packages
- The Abstract Windowing Toolkit (AWT) was the original Java GUI package
- The Swing package provides additional and more versatile components
- Both packages are needed to create a Java GUI-based program

GUI Containers

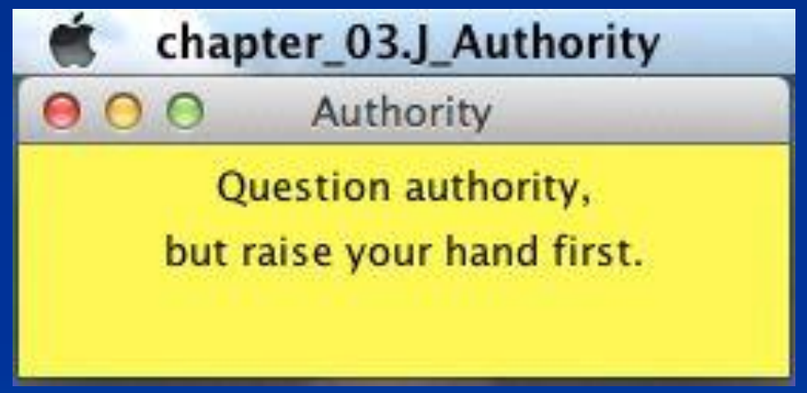
- A GUI container is a component that is used to hold and organize other components
- A frame is a container displayed as a separate window with a title bar
- It can be repositioned and resized on the screen as needed
- A panel is a container that cannot be displayed on its own but is used to organize other components
- A panel must be added to another container (like a frame or another panel) to be displayed
- A GUI container can be classified as either heavyweight or lightweight
- A heavyweight container is one that is managed by the underlying operating system, such as a frame
- A lightweight container is managed by the Java program itself such as a panel

Labels

- A label is a GUI component that displays a line of text and/or an image
- Labels are usually used to display information or identify other components in the interface

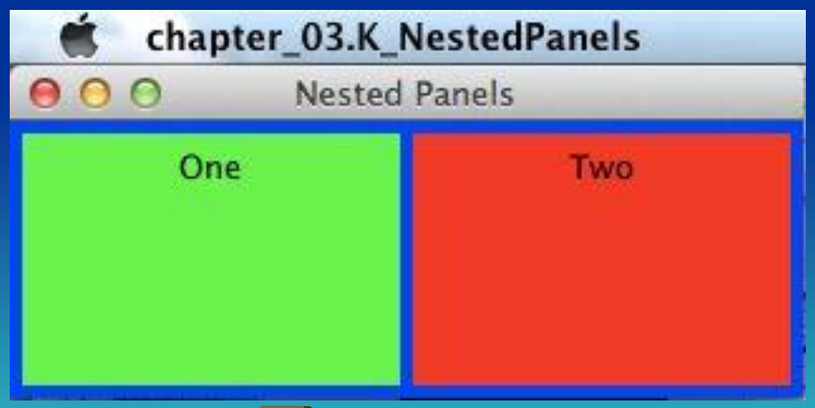
G_MyIntegerWrapper.java | J_Authority.java

```
import javax.swing.JPanel;

public class J_Authority
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Authority");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel primary = new JPanel();
        primary.setBackground(Color.yellow);
        primary.setPreferredSize(new Dimension(250, 75));

        JLabel label1 = new JLabel("Question authority,");
        JLabel label2 = new JLabel("but raise your hand first.");

        primary.add(label1);
        primary.add(label2);

        frame.getContentPane().add(primary);
        frame.pack();
        frame.setVisible(true);
    }
}
```



Nested Panels

- Containers that contain other components make up the containment hierarchy of an interface
- This hierarchy can be as intricate as needed to create the visual effect desired




```
package chapter_03;

import java.awt.Color;
import java.awt.Dimension;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class K_NestedPanels
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Nested Panels");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel subPanel1 = new JPanel();
        subPanel1.setPreferredSize(new Dimension(150, 100));
        subPanel1.setBackground(Color.green);
        JLabel label1 = new JLabel("One");
        subPanel1.add(label1);

        JPanel subPanel2 = new JPanel();
        subPanel2.setPreferredSize(new Dimension(150, 100));
        subPanel2.setBackground(Color.red);
        JLabel label2 = new JLabel("Two");
        subPanel2.add(label2);

        JPanel primary = new JPanel();
        primary.setBackground(Color.blue);
        primary.add(subPanel1);
        primary.add(subPanel2);

        frame.getContentPane().add(primary);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Images

- Images can be displayed in a Java program in various ways
- As we've seen, a JLabel object can be used to display a line of text
- It can also be used to display an image
- That is, a label can be composed of text, an image, or both at the same time
- The ImageIcon class is used to represent the image that is stored in a label
- If text is also included, the position of the text relative to the image can be set explicitly
- The alignment of the text and image within the label can be set as well

```java
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

public class L_LabelDemo
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Label Demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        ImageIcon icon = new ImageIcon("devil.gif");

        JLabel label1, label2, label3;

        label1 = new JLabel("Devil Left", icon, SwingConstants.CENTER);

        label2 = new JLabel("Devil Right", icon, SwingConstants.CENTER);
        label2.setHorizontalTextPosition(SwingConstants.LEFT);
        label2.setVerticalTextPosition(SwingConstants.BOTTOM);

        label3 = new JLabel("Devil Above", icon, SwingConstants.CENTER);
        label3.setHorizontalTextPosition(SwingConstants.CENTER);
        label3.setVerticalTextPosition(SwingConstants.BOTTOM);

        JPanel panel = new JPanel();
        panel.setBackground(Color.cyan);
        panel.setPreferredSize(new Dimension(200, 250));
        panel.add(label1);
        panel.add(label2);
        panel.add(label3);

        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

javax.swing.JLabel.JLabel(String text, Icon icon, int horizontalAlignment)

JLabel

```
public JLabel(String text,
              Icon icon,
              int horizontalAlignment)
```

Creates a `JLabel` instance with the specified text, image, and horizontal alignment. The label is centered vertically in its display area. The text is on the trailing edge of the image.

Parameters:

Another Example

- http://www.java2s.com/Code/Java/Swing-JFC/AsimpledemonstrationoftextalignmentinJLabels.htm

L_LabelDemo.java | *Y_AlignmentExample.java

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

public class Y_AlignmentExample
{
    public static void main(String[] args)
    {
        JLabel label1 = new JLabel("BottomRight", SwingConstants.RIGHT);
        JLabel label2 = new JLabel("CenterLeft", SwingConstants.LEFT);
        JLabel label3 = new JLabel("TopCenter", SwingConstants.CENTER);
        label1.setVerticalAlignment(SwingConstants.BOTTOM);
        label2.setVerticalAlignment(SwingConstants.CENTER);
        label3.setVerticalAlignment(SwingConstants.TOP);

        label1.setBorder(BorderFactory.createLineBorder(Color.black));
        label2.setBorder(BorderFactory.createLineBorder(Color.black));
        label3.setBorder(BorderFactory.createLineBorder(Color.black));

        JFrame frame = new JFrame("AlignmentExample");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel p = new JPanel(new GridLayout(3, 1, 8, 8));
        p.add(label1);
        p.add(label2);
        p.add(label3);
        p.setBorder(BorderFactory.createEmptyBorder(8, 8, 8, 8));
        frame.setContentPane(p);
        frame.setSize(200, 200);
        frame.setVisible(true);
    }
}
```