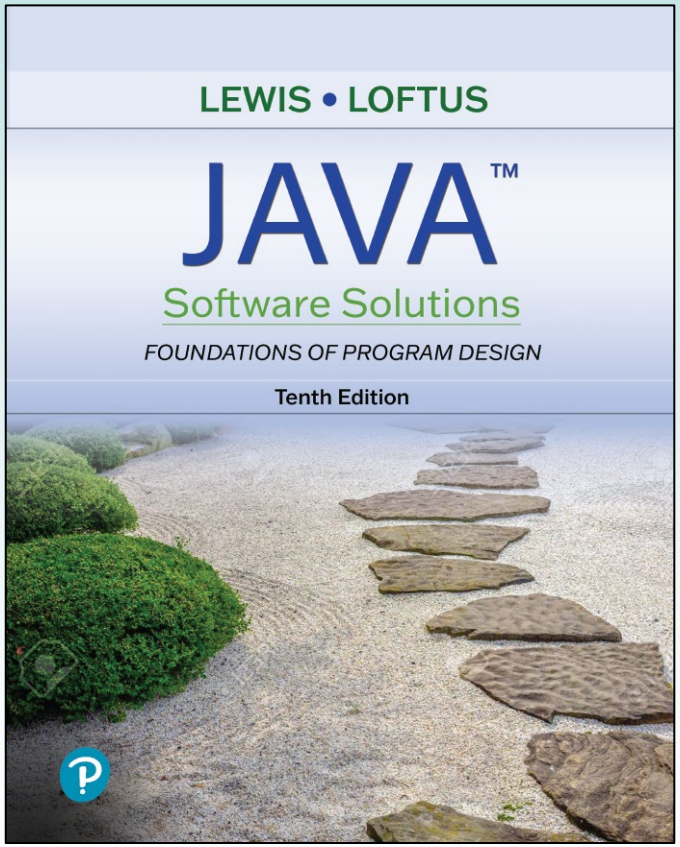
Chapter 3
Using Classes and Objects



Java Software Solutions
Foundations of Program Design
10th Edition

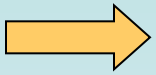
revised 09/18/2025, 02/09/2026, 08/08/2026

John Lewis
William Loftus

Using Classes and Objects

- Chapter 3 focuses on:
  - object creation and object references
  - the `String` class and its methods
  - the Java API class library
  - the `Random` and `Math` classes
  - formatting output
  - enumerated types
  - wrapper classes
  - Introduction to JavaFX ☺
  - Shapes and Color ☺

Outline



Creating Objects

The String Class

The Random and Math Classes

Formatting Output

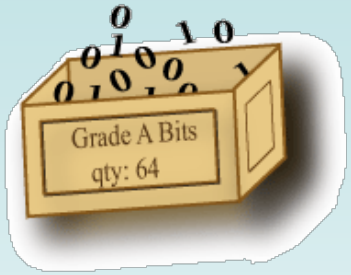
- **Enumerated Types**

Wrapper Classes

- **Introduction to JavaFX**
- **Shapes and Color**

§3.1 Creating Objects

- A variable is a small region of memory
  - A “little box of memory”
- A variable holds either:
  - a primitive value or
  - a reference to an object (the address of an object in memory)



- A variable can only hold the type it was declared to hold
  - `int sum;` // primitive type int
  - `Scanner scan;` // object reference

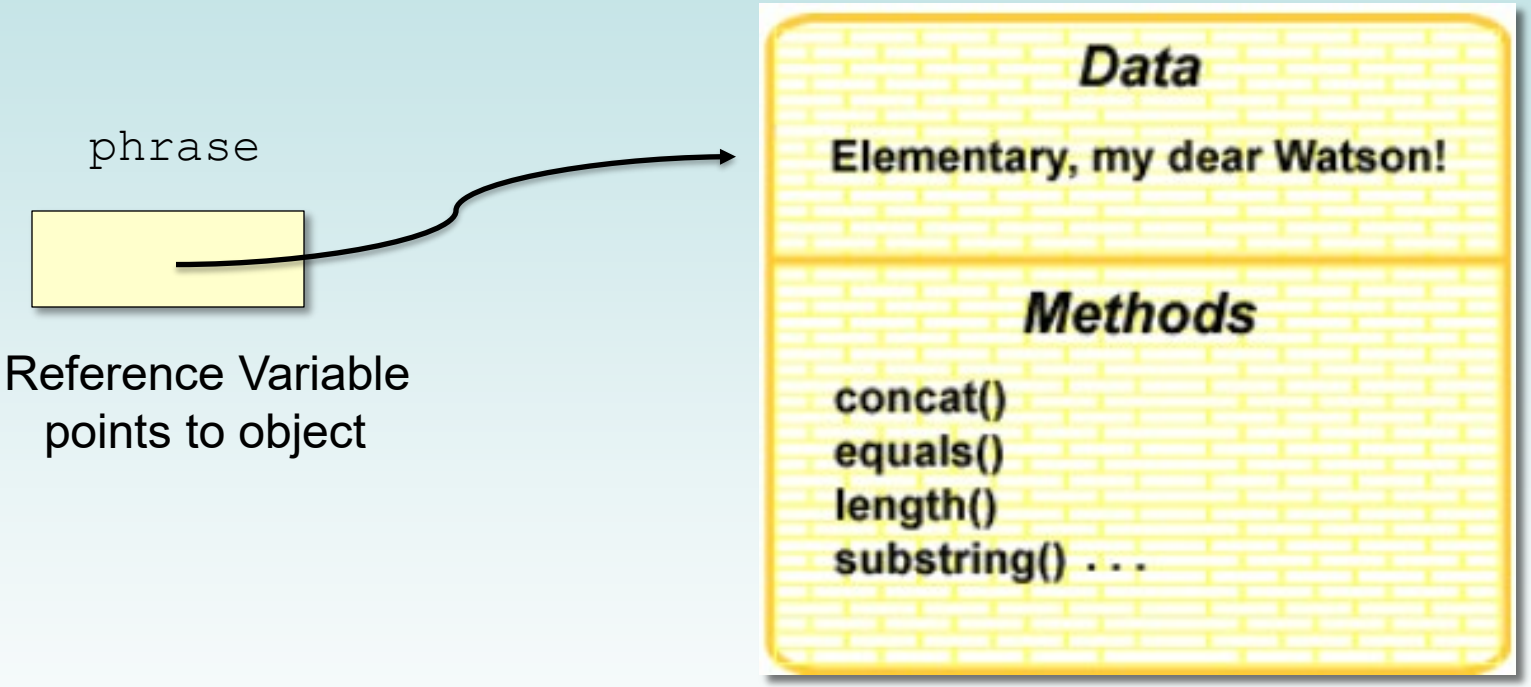
Primitive Variable

```
int sum = 43;
```

1. The variable (a section of memory, 4 bytes)
2. The name of the variable (in source code)
3. What is in the variable (a primitive int)

Reference Variable

A reference variable is a section of memory (4 or 8 bytes long) that holds a reference to an object.

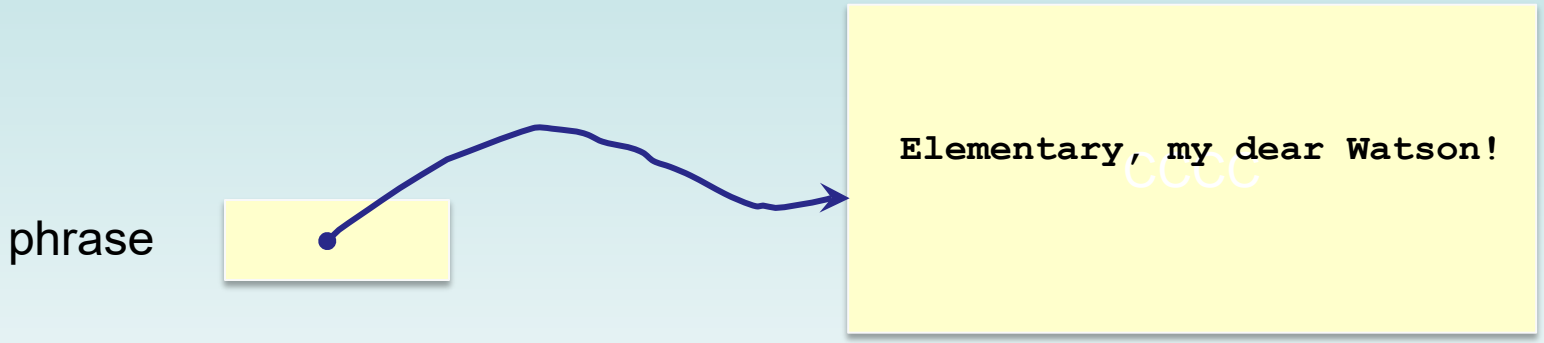


An object holds both data and methods that use that data.
An object is much bigger than a variable.

```
String phrase = "Elementary, my dear Watson!" ;
```

Five Things

```
String phrase = "Elementary, my dear Watson!";
```



A String object

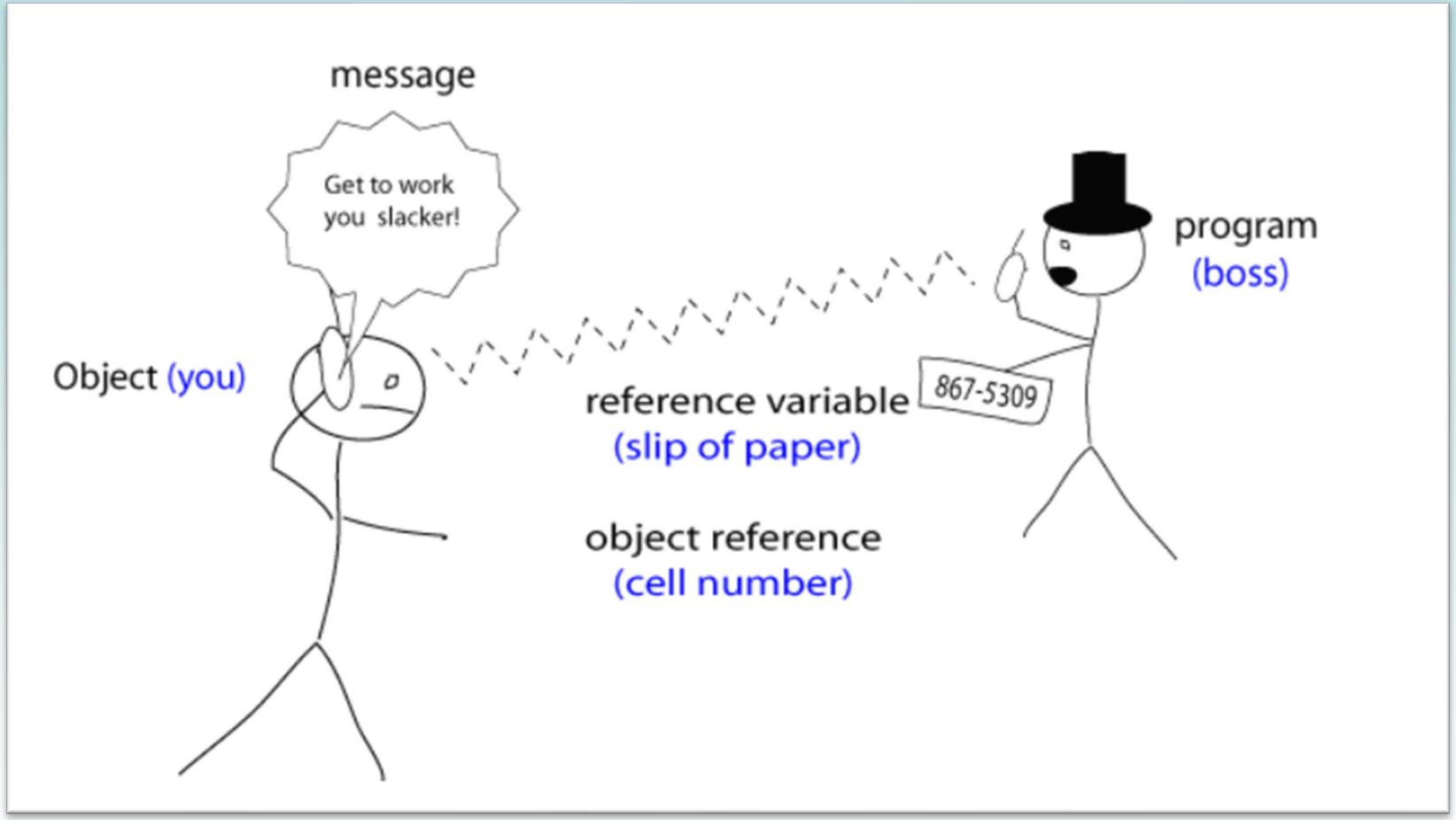
1. The variable at run-time (here, an object reference variable)
2. The name of the variable (in source code)
3. What is in the variable (a reference to an object at run time)
4. The object (here, a string object)
5. What is in the object (data and methods)

Creating Objects

- To declare an *object reference variable,* use the class name followed by the variable's name

```
String phrase;
```

- No object is created with this declaration
- An object reference variable at run time holds the address of an object in memory (it points to an object when there is one)
- The object itself must be created separately
- An object reference variable can point to different objects *of the same class* at different times



Creating Objects

- Use the **`new`** operator to create an object
- Creating an object is called *instantiation* or *construction*
- An object is an *instance* of a particular class

```
String phrase;
. . . // other statements may be here

phrase = new String("Change is inevitable");
```

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

However, for String objects, a shorthand notation can be used to do the same thing:

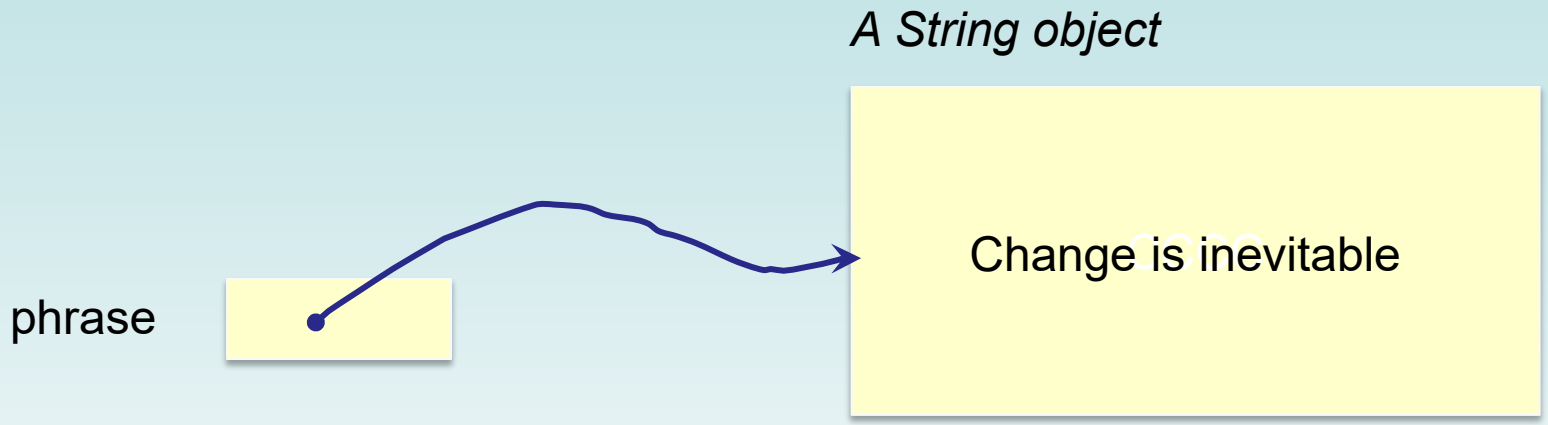
```
phrase = "Change is inevitable";
```

```
public class Demo
{
   // object is accessed using reference variable phrase
   public static void main(String[] args)
   {
      String phrase = "Change is inevitable";
      System.out.println("Original string:" + phrase );

      // Change what phrase points to
      phrase = "Things stay the same";
      System.out.println("New string:" + phrase );
    }
}
```

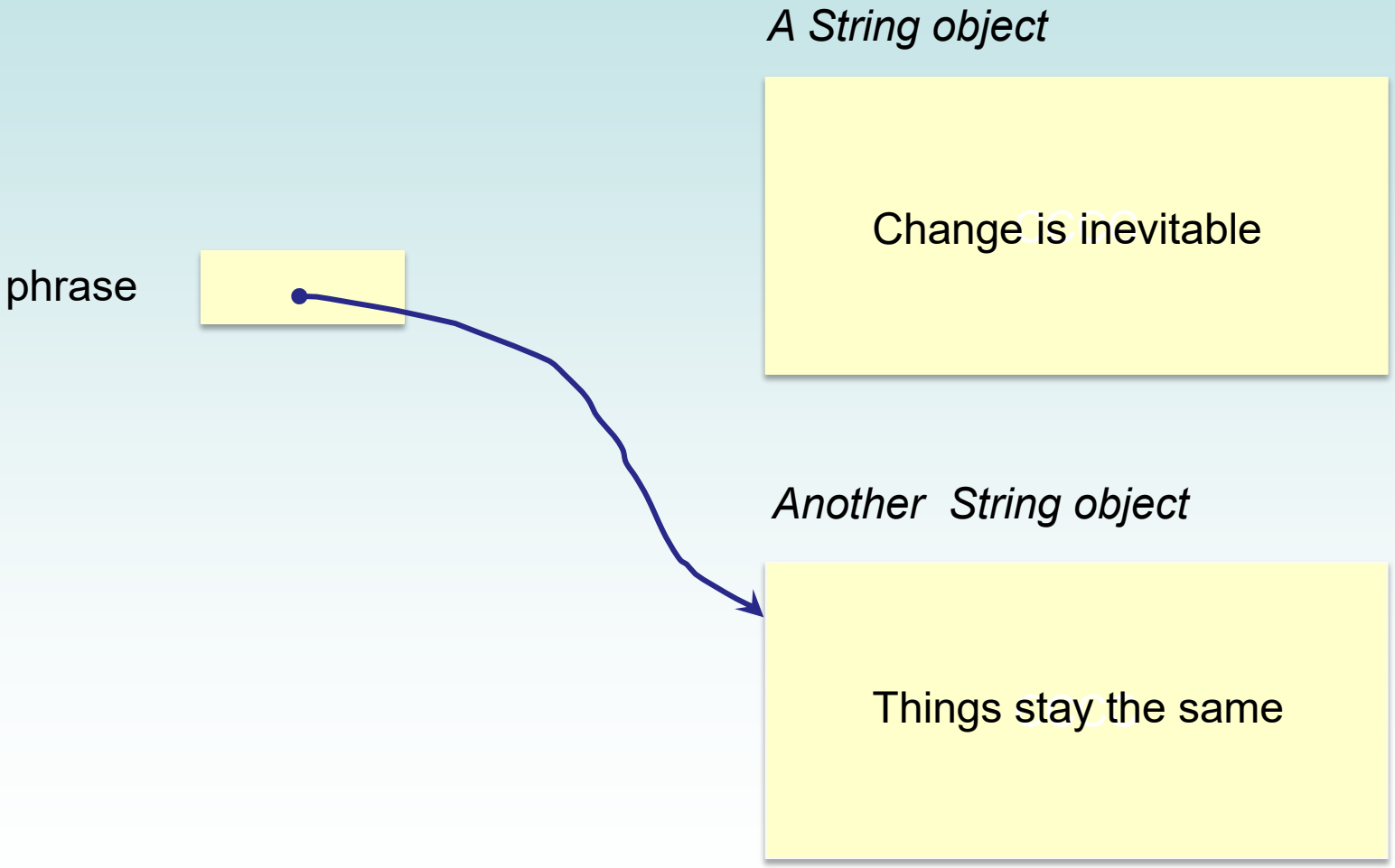
```
Original string: Change is inevitable
New string: Things stay the same
```

The third statement creates a new object and then a reference to that new object is put into the variable *phrase*.

```
String phrase = "Change is inevitable";
```



```
phrase = "Things stay the same";
```



Invoking Methods

- Once an object has been instantiated, use the *dot operator* to invoke (to call) (to run) its methods

```
numChars = phrase.length()
```

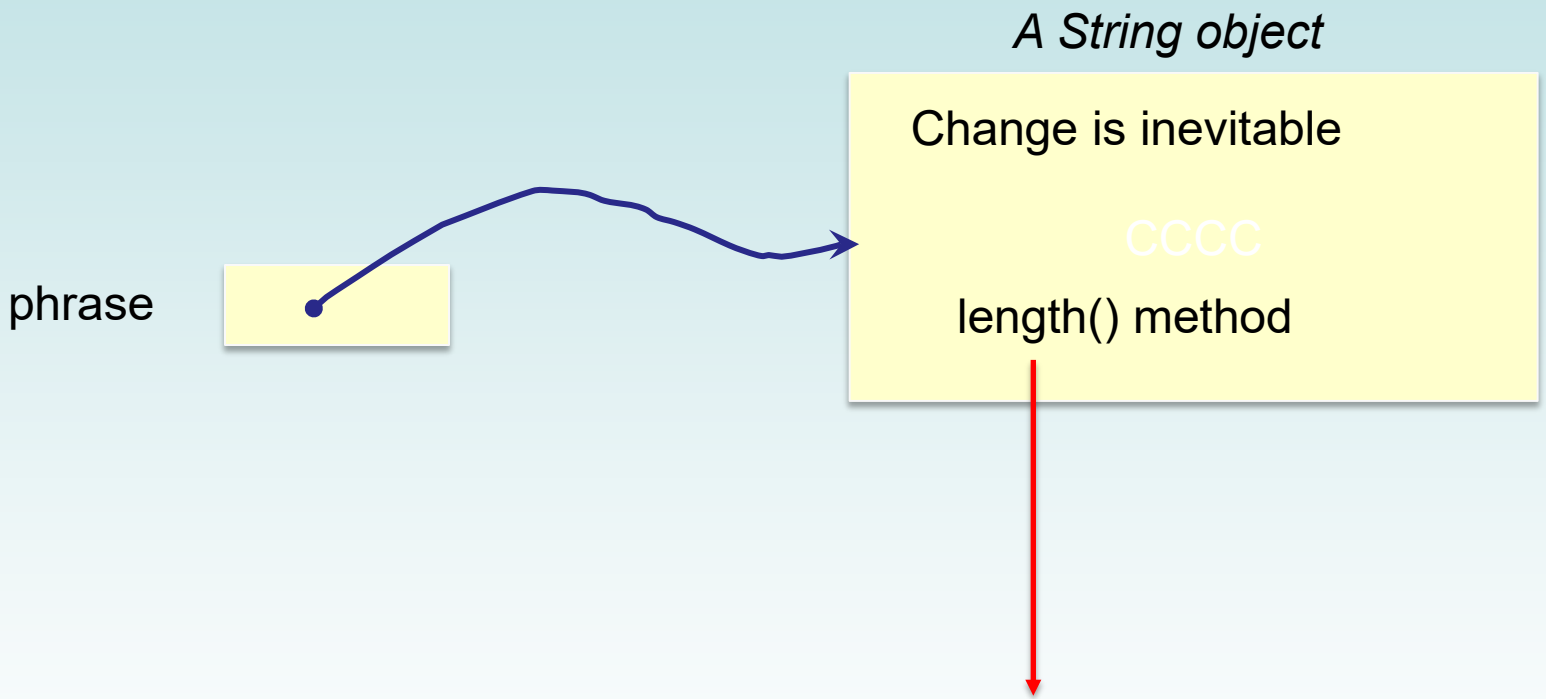
- A method may *return a value*, which can then be used in an assignment or expression
  - the value is either a primitive type or an object reference
- A method invocation (a call) asks an object to perform a service

```
public class Demo
{

   public static void main(String[] args)
   {
      String phrase = "Change is inevitable";
      System.out.println("Original string:" + phrase );

      // Invoke a method
      System.out.println("Length:" + phrase.length() );
    }
}
```

Call a method of the object pointed to by *phrase*.

```
Original string: Change is inevitable
Length: 20
```

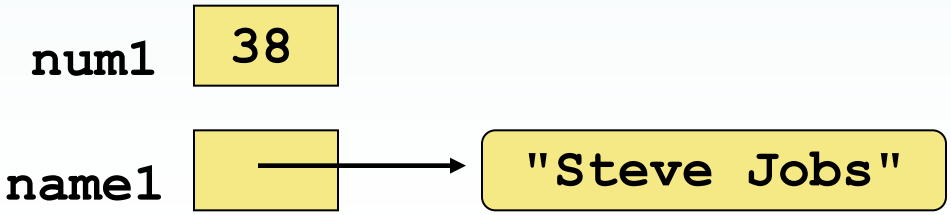
```
System.out.println("Length:" + phrase.length() );
```




```
System.out.println("Length:" +  20  );
```

References

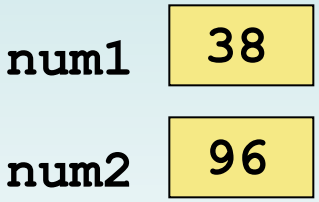
- A primitive variable contains a value
  - The value itself is in the variable (the “box of memory”)
- An object reference variable contains the address of an object
  - The object is somewhere else in memory
  - Use the address to get to the object
- An object reference is a pointer to the object
  - Show a reference graphically as an arrow:



Assignment Revisited

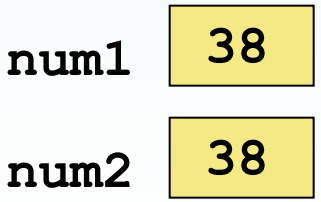
- Assignment makes a copy of a value and stores it in a variable
- For primitive types:

Before:




```
num2 = num1; //copy the value in num1 to num2
```

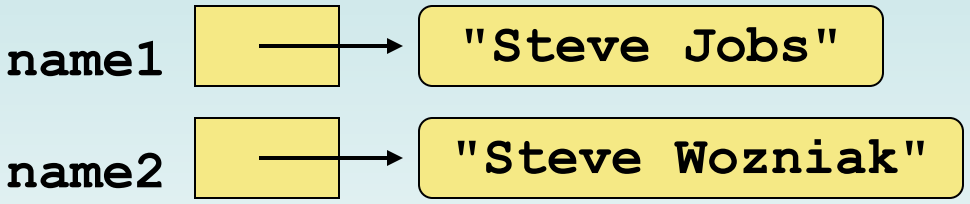
After:



Reference Assignment

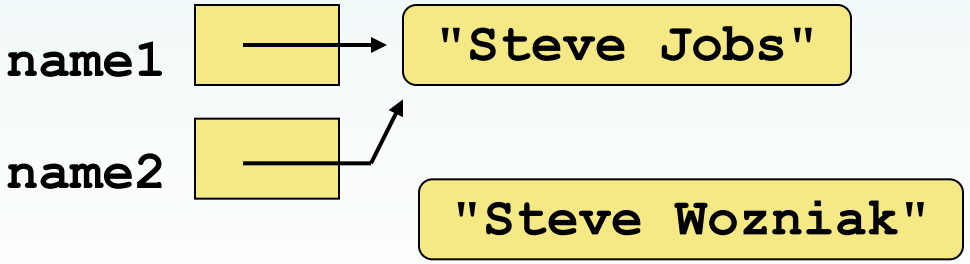
For object references, assignment copies the reference to an object, but not the object:

Before:




```
//copy the reference in name1 to name2
name2 = name1;
```

After:



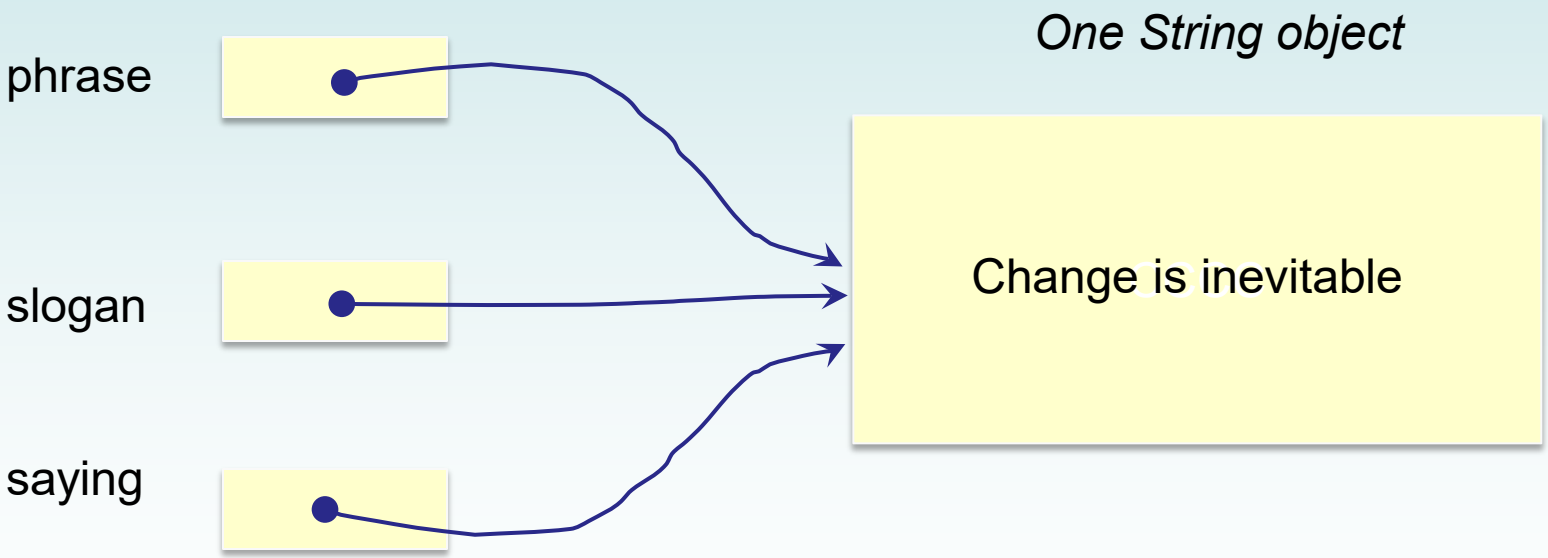
Aliases

- Two or more references that refer to the same object are called *aliases* of each other
- A particular object can be accessed using multiple reference variables
  - Many people have your phone number. All of them can call you using their copy of your number. But there is only one you.
- Aliases are useful, but tricky
- Changing an object using a reference changes it for all of its aliases, because there is really only one object

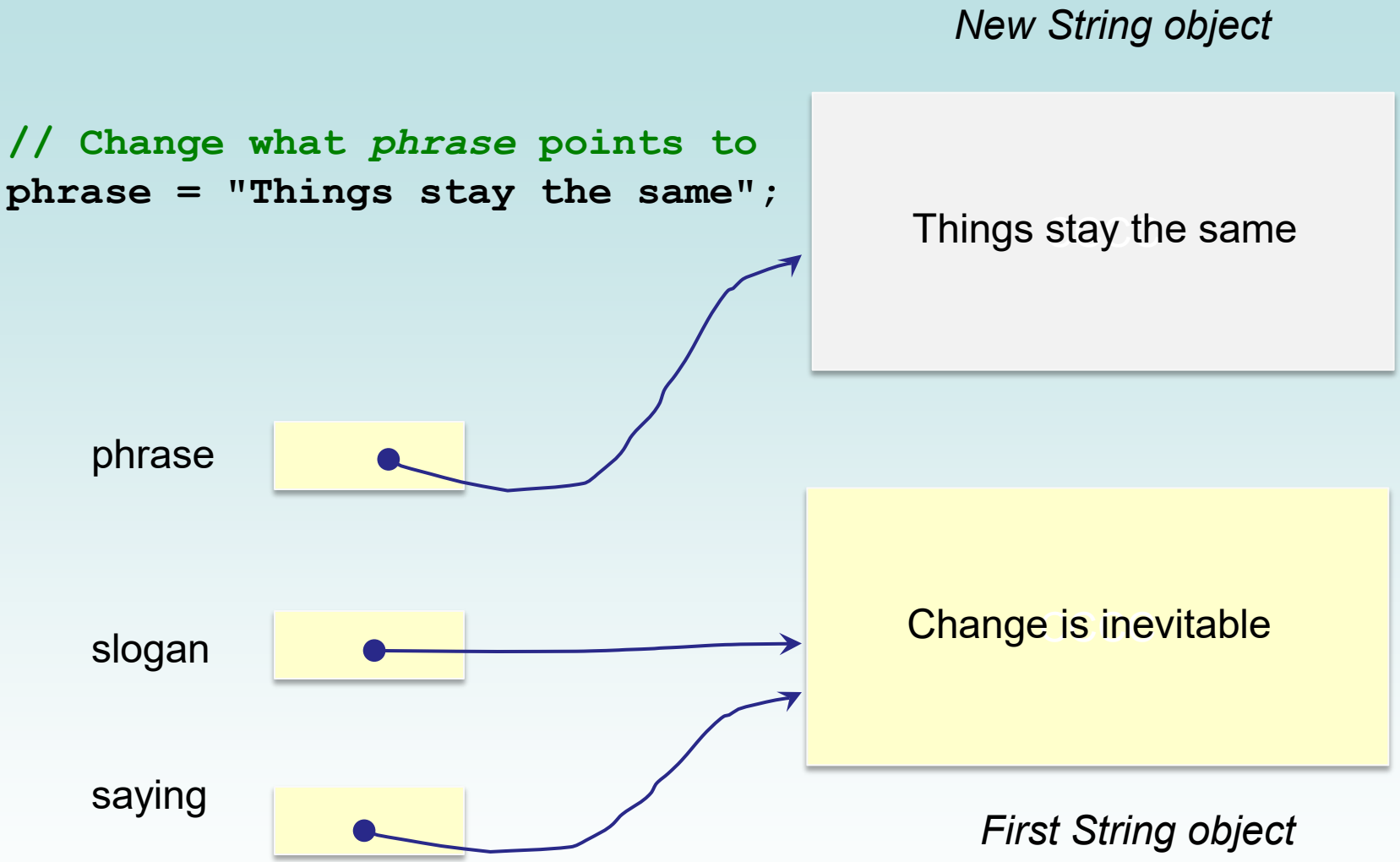
```
public class AliasDemo  // Perhaps try this with BlueJ
{
   //  a single object can be accessed using multiple reference variables
   public static void main(String[] args)
   {
      String phrase = "Change is inevitable";
      String slogan, saying ;
      slogan = phrase ;
      saying = phrase ;
      System.out.println("Original string:" + phrase );
      System.out.println("Using slogan: " + slogan );
      System.out.println("Using saying: " + saying );

      // Change what phrase points to
      phrase = "Things stay the same";
      System.out.println("New string:" + phrase );
      System.out.println("Using slogan: " + slogan );
      System.out.println("Using saying: " + saying );
   }
}
```

```
Original string: Change is inevitable
Using slogan: Change is inevitable
Using saying: Change is inevitable
New string: Things stay the same
Using slogan: Change is inevitable
Using saying: Change is inevitable
```

```
String phrase = "Change is inevitable";
String slogan, saying ;
slogan = phrase ;
saying = phrase ;
```



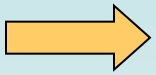
Three reference variables pointing at the same object.



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 useless objects' memory to the system for future use
- In other languages, the programmer is responsible for performing garbage collection

Outline



- Creating Objects
- The String Class
- The Random and Math Classes
- Formatting Output
- ☺ Enumerated Types
- Wrapper Classes
- ☺ Introduction to JavaFX
- ☺ Shapes and Color

§3.2 The String Class

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

```
String title;

title = "Java Software Solutions";
```

- This puts a reference to the string in `title`
- Omitting **`new`** works only for strings
- A string literal (enclosed in double quotes) corresponds to a `String` object

String Methods

- Once a `String` object has been created it cannot be changed
- We say that an object of the `String` class is *immutable*
- However, several methods of the `String` class create new `String` objects that are modified versions of the original
  - A `String` object can create other `String` objects .
- After creating the new `String`, the method returns a reference to it

String Indexes

- You can refer to a particular character within a string
- Do this with the character's numeric *index*
- Indexes begin at zero

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

Methods

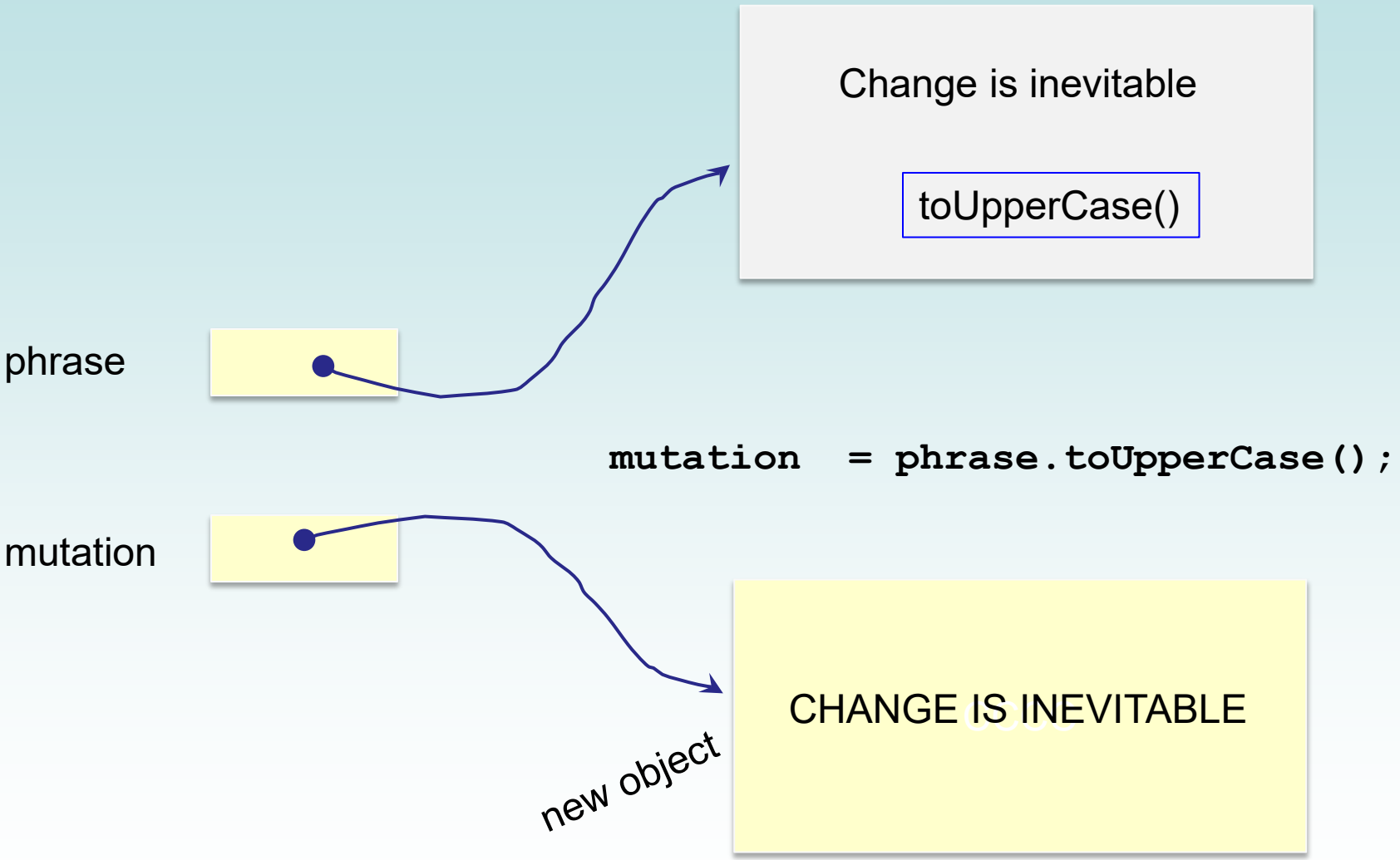
- **`toUpperCase()`**
  - creates a new string, based on the original string, but with upper case characters
- **`replace(char oldChar, char newChar)`**
  - creates a new string by replacing all occurrences of `oldChar` in the original string with `newChar`.
- **`substring(int beginIndex)`**
  - creates a new string using chars from `beginIndex` to the end
- **`substring(int beginIndex, int endIndex)`**
  - creates a new string using chars from `beginIndex` to `endIndex-1`

```
public class StringMut
{
   //-----------------------------------------------------------------
   //  Prints a string and various mutations of it.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      String phrase = "Change is inevitable";
      String mutation;

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

      // Create  a new string
      mutation  = phrase.toUpperCase();

      // Print mutatation string
      System.out.println("Mutation: " + mutation);

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

Output

```
Original string: "Change is inevitable"
Length of string: 20
Mutation: CHANGE IS INEVITABLE
Mutated length: 20
```

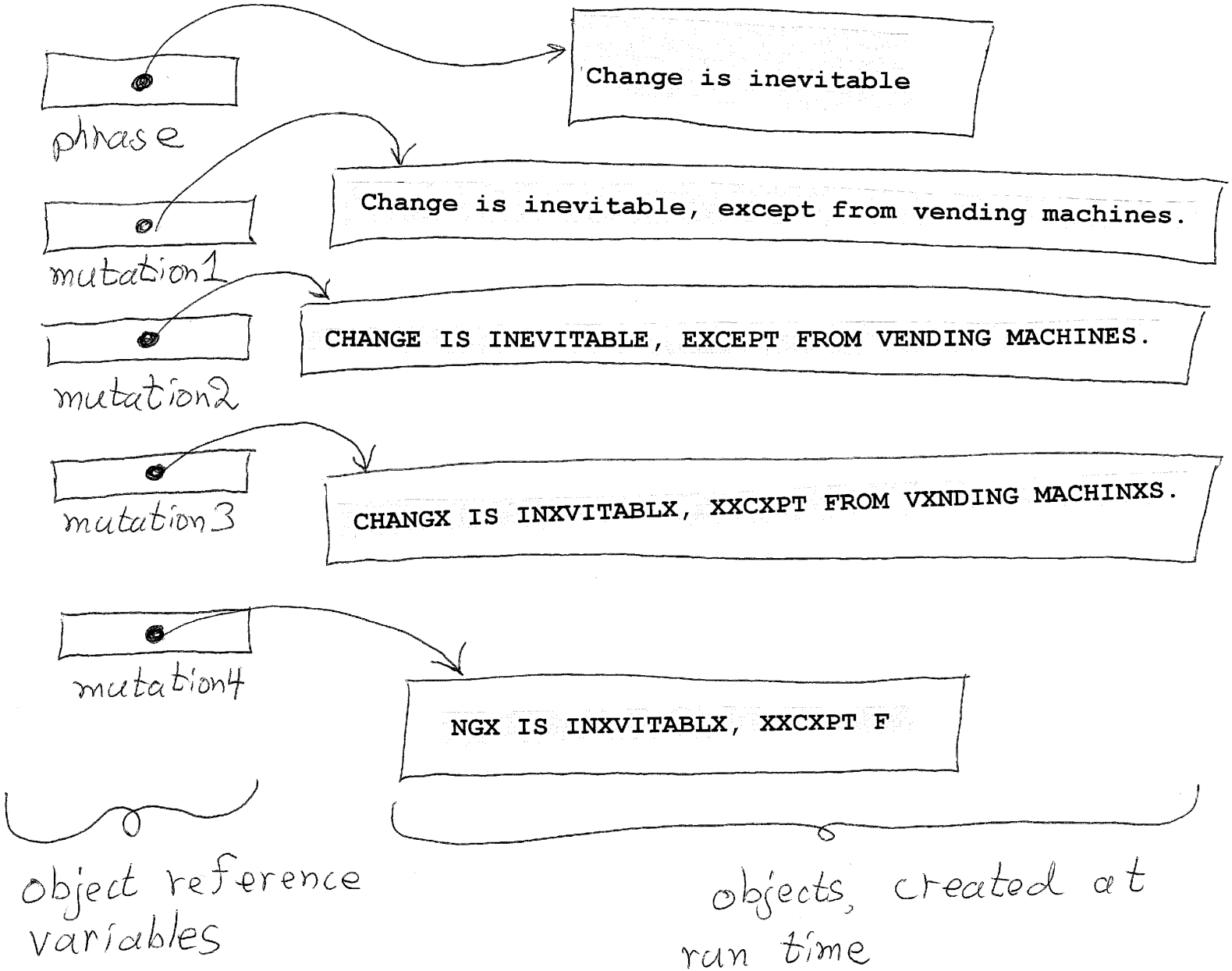
```
String phrase = "Change is inevitable";
```



```
public class StringMutation
{
   //-----------------------------------------------------------------
   //  Prints a string and various mutations of it.
   //-----------------------------------------------------------------
   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());

      // Create 4 new String objects and point 4 reference variables
      // at them
      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());
   }
}
```

Output

```
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
```



Quick Check

What output is produced by the following?

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

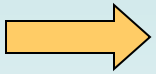
Quick Check What output is produced by the following?

```
String str = "Space, the final frontier.";

System.out.println(str);
System.out.println(str.length());
System.out.println(str.substring(7));
System.out.println(str.toUpperCase());
System.out.println(str.length());
System.out.println(str);
```

creates a new string which is used by println and then vanishes. The original string does not change.

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

Outline



- **Creating Objects**
- **The String Class**
- **The Random and Math Classes**
- **Formatting Output**
- ☺ **Enumerated Types**
- **Wrapper Classes**
- ☺ **Introduction to JavaFX**
- ☺ **Shapes and Color**

Class Libraries

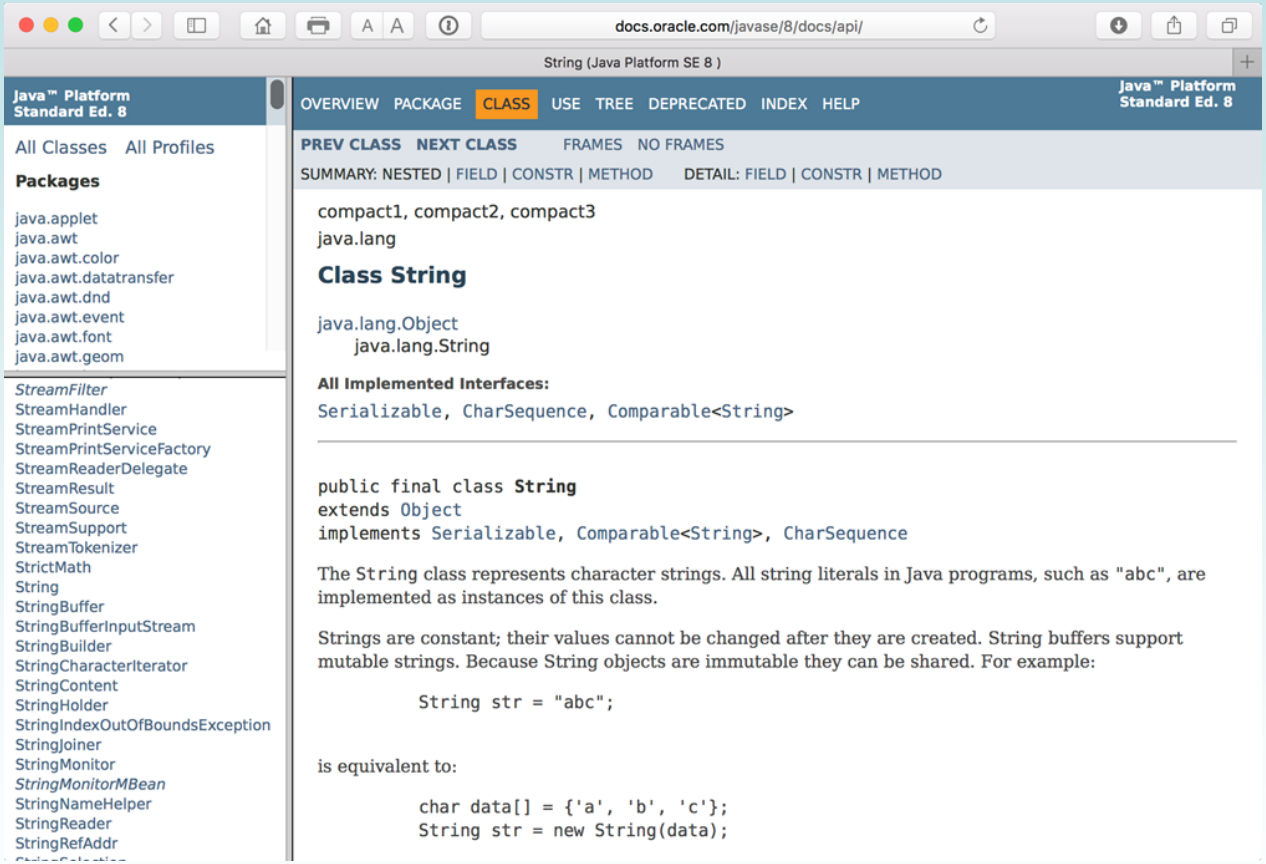
- A *class library* is a collection of pre-written classes used to develop programs
  - A library contains many packages
  - A package contains many classes
- The *Java standard class library* is part of the Java development environment
- Its classes are not part of the fundamental Java language, but are often used
- 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 **A**pplication **P**rogramming **I**nterface
- Clusters of related classes are sometimes referred to as specific APIs:
  - The JavaFX API
  - The Database API

The Java API

- Get comfortable using the online Java API documentation



§3.3 Packages

- Classes in the Java API are organized into *packages*
- 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 scan;
```

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

```
import java.util.Scanner;

Scanner scan;
```

- To import all classes in a particular package, use the * wildcard character

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

The import Declaration

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

§3.4 The Random Class

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

double nextDouble() return the next pseudorandom, uniformly distributed double value between 0.0 and a little bit less than 1.0

float nextFloat() return the next pseudorandom, uniformly distributed float value between 0.0 and little bit less than 1.0

double nextGaussian() return the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0

int nextInt() return the next pseudorandom, uniformly distributed int value: negative, zero, or positive

int nextInt(int n) return the next pseudorandom, uniformly distributed int value between 0 (inclusive) and n (exclusive)

void setSeed(long seed) set the seed of this random number generator using a single long seed.

```
//********************************************************************
//  RandomNumbers.java       Author: Lewis/Loftus
//
//  Demonstrates the creation of pseudo-random numbers using the
//  Random class.
//********************************************************************

import java.util.Random;

public class RandomNumbers
{
   //-----------------------------------------------------------------
   //  Generates random numbers in various ranges.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Random generator = new Random();
      int num1;
      double num2;

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

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

Sample Run

```
A random integer:
672981683
From 0 to 9: 0
```

continued

```
        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);
    }
}
```

Sample Run

```
A random integer: 672981683
From 0 to 9: 0
From 1 to 10: 3
From 20 to 34: 30
From -10 to 9: -4
A random double (between 0-1): 0.18538326
From 1 to 6: 3
```

Quick Check

Given a `Random` 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	`gen.nextInt(13)`
1 to 20	`gen.nextInt(20) + 1`
15 to 20	`gen.nextInt(6) + 15`
-10 to 0	`gen.nextInt(11) - 10`

§3.5 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(0.45) + Math.sqrt(delta);
```

- Arguments to trig methods are in radians
- Most methods expect (and return) double
- See `Quadratic.java`

```
//********************************************************************
//  Quadratic.java       Author: Modified from the textbook
//
//  Demonstrates the use of the Math class to perform a calculation
//  based on user input.
//********************************************************************

import java.util.Scanner;

public class Quadratic
{
   //-----------------------------------------------------------------
   //  Determines the roots of a quadratic equation.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      double 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.nextDouble();
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

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

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

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

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

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

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

This program assumes that the discriminant is positive.
A better program would deal with all cases.

continued

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

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

Sample Run

```
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
```

Some Methods of the Math class

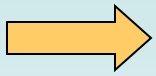
Method	Returns	
Math.sqrt(x)	$\sqrt{x}$	x is double precision, return value is double precision
Math.pow(x,y)	x^y	
Math.sin(x)	sin(x)	
Math.cos(x)	cos(x)	
Math.tan(x)	tan(x)	
Math.asin(x)	arcsin(x)	
Math.acos(x)	arccos(x)	
Math.atan(x)	arctan(x)	
Math.exp(x)	e^x	
Math.log(x)	ln(x)	
Math.log10(x)	log10(x)	
Math.round(x)	If x is float, the closest int to x. If x is double, the closest long to x.	
Math.abs(x)	|x| (argument any numeric type)	
Math.max(x,y)	The maximum of the two values (arguments both same numeric type)	
Math.min(x,y)	The minimum of the two values (arguments both same numeric type)	

Constants of the Math class

Constant	Value
Math.PI	pi – π – double precision value
Math.E	e – the base of natural logarithms, 2.718281828…

Never hard code a literal for either of these.

Outline



- Creating Objects
- The String Class
- The Random and Math Classes
- Formatting Output
  - NumberFormat
  - DecimalFormat
- ☺ Enumerated Types
- Wrapper Classes
- ☺ Introduction to JavaFX
- ☺ Shapes and Color

§3.6 Formatting Output *(skim)*

- It is often nice to format output values
- The Java standard class library contains classes that format numbers
- The `NumberFormat` class formats values as currency or as percentages
- The `DecimalFormat` class formats values based on a pattern
- Both are part of the `java.text` package

Formatting Output *(skip)*

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

```
NumberFormat.getCurrencyInstance()

NumberFormat.getPercentInstance()
```

- `new` is not used to create the formatter object
- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format

(skip)

```
//********************************************************************
//  Purchase.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the NumberFormat class to format output.
//********************************************************************

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

public class Purchase
{
   //-----------------------------------------------------------------
   //  Calculates the final price of a purchased item using values
   //  entered by the user.
   //-----------------------------------------------------------------
   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);
```

(skip)

Sample Run

```
Enter the quantity: 5
Enter the unit price:3.87
Subtotal: $19.35
Tax: $1.16 at 6%
Total: $20.51
```

continued

```
        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));
    }
}
```

Formatting Output *(skim)*

- 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`

```
//********************************************************************
//  CircleStats.java       Author: Lewis/Loftus
//
//  Demonstrates the formatting of decimal values using the
//  DecimalFormat class.
//********************************************************************

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

public class CircleStats
{
   //-----------------------------------------------------------------
   //  Calculates the area and circumference of a circle given its
   //  radius.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      int radius;
      double area, circumference;

      Scanner scan = new Scanner(System.in);
```

continued

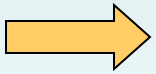
continued

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

        area = Math.PI * radius * radius ;
        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));
    }
}
```

Sample Run

```
Enter the circle's radius: 5
The circle's area: 78.54
The circle's circumference: 31.416
```

Outline



- **Creating Objects**
- **The String Class**
- **The Random and Math Classes**
- **Formatting Output**
- ☺ **Enumerated Types**
- **Wrapper Classes**
- ☺ **Introduction to JavaFX**
- ☺ **Shapes and Color**

☺ §3.7 Enumerated Types *(skip)*

- A variable can be an *enumerated type*
- An enumerated type declaration lists all possible values for a variable of that type
- The values are identifiers of your own choosing
- To create an enumerated type called `Season`

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

- The list can have any number of values

☺ Enumerated Types *(skip)*

- A declare a variable:

```
Season time;
```

- Assign 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 *(skip)*

- 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

☺ Enumerated Types *(skip)*

- An enumerated type is a special type of class
  - 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`

```
//********************************************************************
//  IceCream.java       Author: Lewis/Loftus
//
//  Demonstrates the use of enumerated types.
//********************************************************************

public class 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());
```

continued

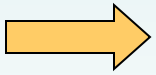
continued

```
        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: "
    }
}
```

Output

```
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
```

Outline

- **Creating Objects**
- **The String Class**
- **The Random and Math Classes**
- **Formatting Output**
  - ☺ **Enumerated Types**
-  **Wrapper Classes**
  - ☺ **Introduction to JavaFX**
  - ☺ **Shapes and Color**

§3.8 Wrapper Classes

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

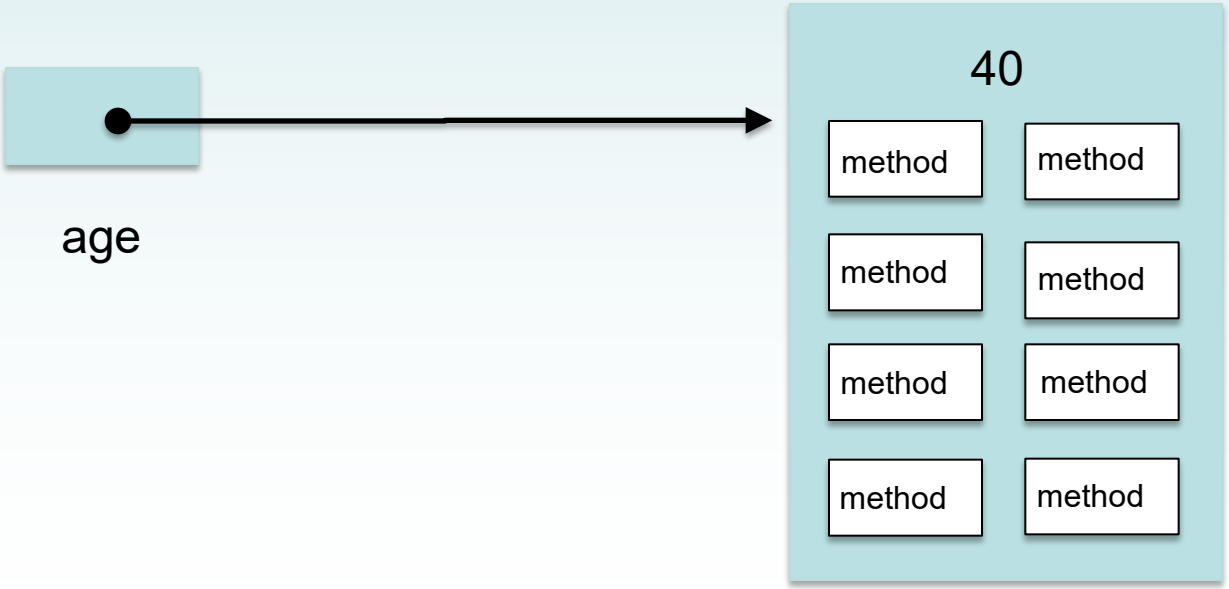
Primitive Type	Wrapper Class
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

Wrapper Classes

- A wrapper holds a primitive value along with methods and constants

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

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



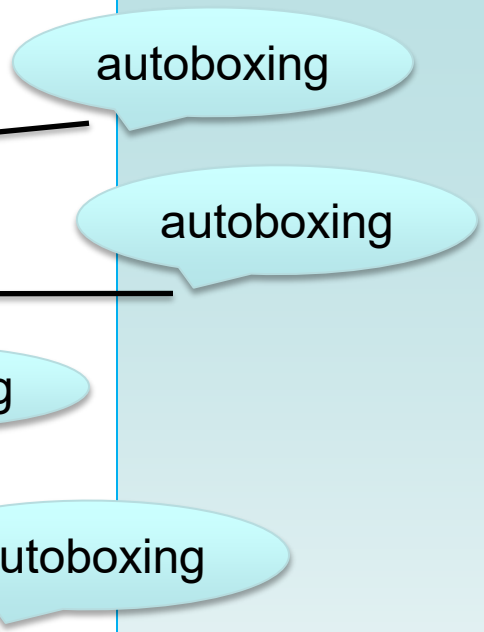
Wrapper Classes

- Use a wrapper class when a primitive value will not work
- For example, often an object contains references to other objects
  - But sometimes all it needs is an integer.
  - A primitive int value can not be stored when a reference is required, but a reference to a wrapper object can.

```
public class WrapperDemo
{
  public static void main ( String[] args )
  {
     Integer val1 = Integer.valueOf(1);
     System.out.println( val1 );
     Integer val2 = Integer.valueOf(2);
     System.out.println( val2 );

    // Calculating Sum
    System.out.println( val1+val2 );

    // Example with another type
    Character another = Character.valueOf('a');
    System.out.println(another);
  }
}
```

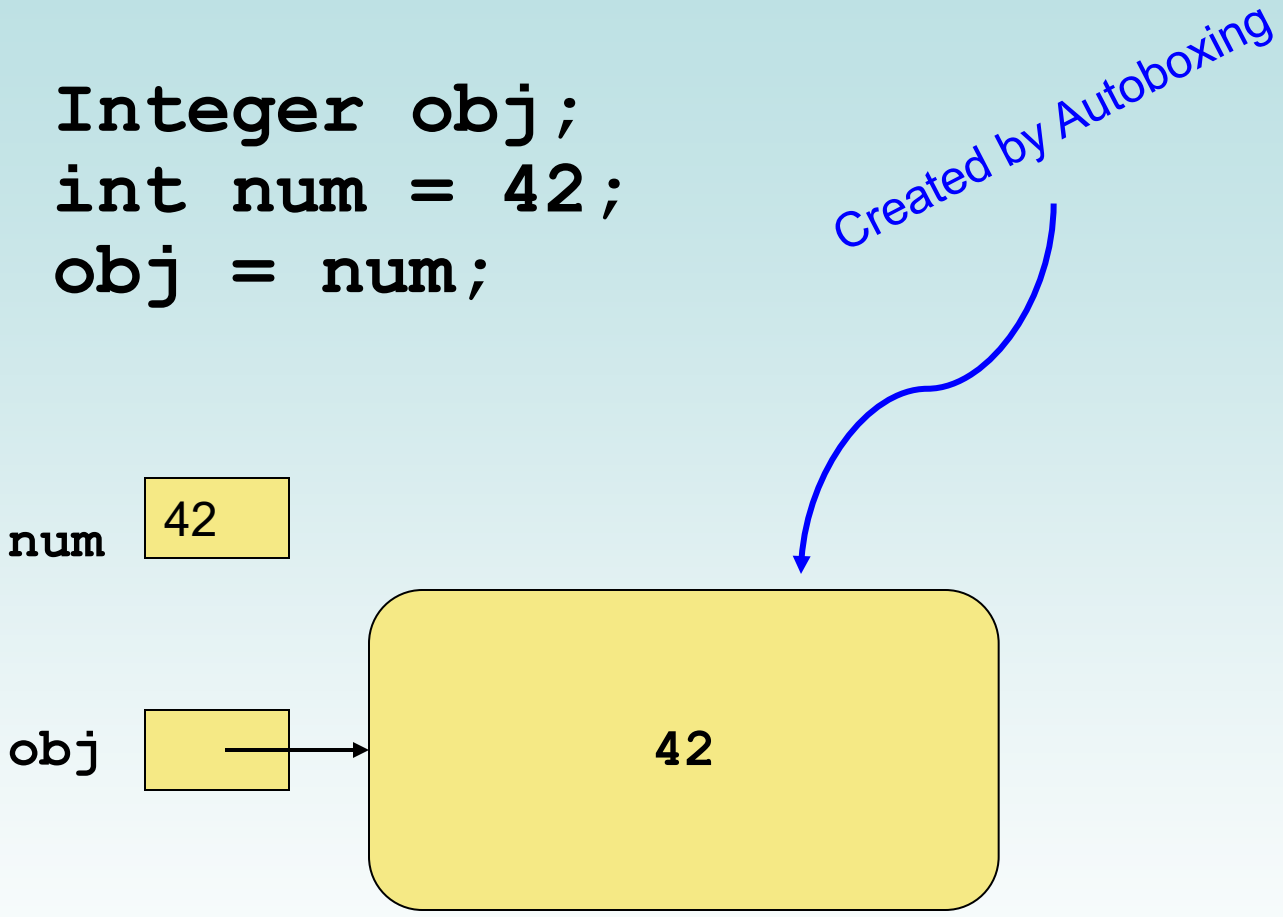
```
1
2
3
a
```

```
public class WrapperDemo
{
  public static void main ( String[] args )
  {
     Integer val1 = 1;
     System.out.println( val1 );
     Integer val2 = 2;
     System.out.println( val2 );

    // Calculating Sum
    System.out.println( val1+val2 );

    // Example with another type
    Character another = 'a';
    System.out.println(another);
  }
}
```

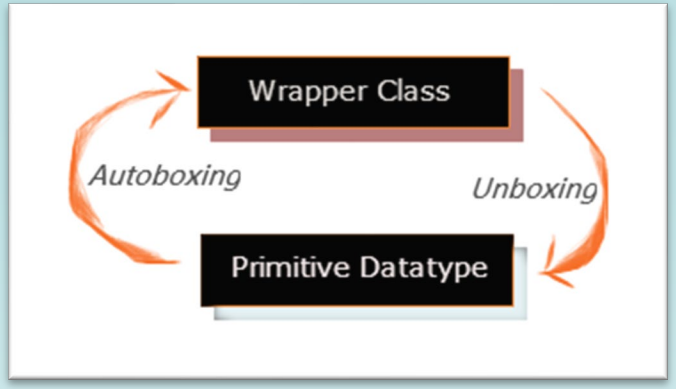



```
1
2
3
a
```

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



Autoboxing



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

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

- The assignment creates the appropriate `Integer` object and puts a reference into `obj`
- The reverse conversion (called *unboxing*) also occurs automatically as needed

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

Quick Check

Are the following assignments valid? Explain.

```
Double value = 15.75;
```

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

Quick Check

Are the following assignments valid? Explain.

```
Double value = 15.75;
```

Yes. The double literal is autoboxed into a **`Double`** object.

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

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

Outline

- Creating Objects
- The String Class
- The Random and Math Classes
- Formatting Output
  - ☺ Enumerated Types
- Wrapper Classes
  - ☺ Introduction to JavaFX
  - ☺ Shapes and Color

☺ Intro to JavaFX (skip)

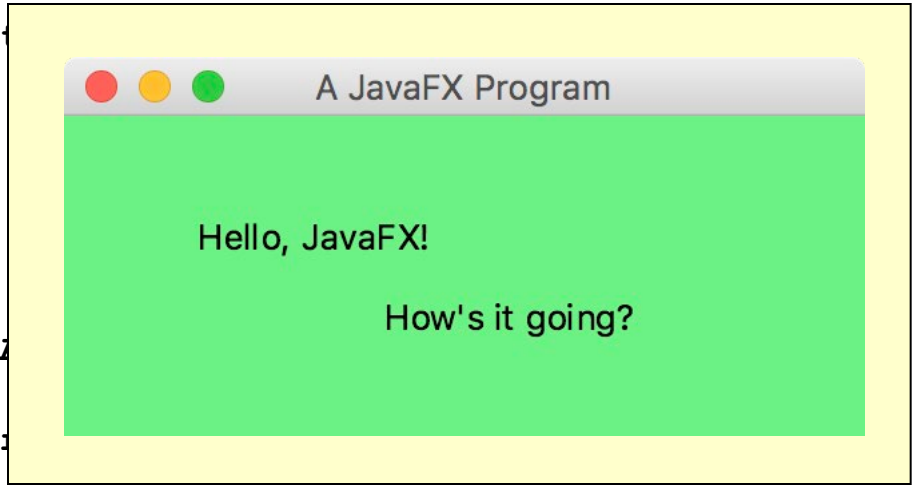
- Our programs so far have been text-based
- They are called *command-line applications*, which interact with the user using simple text prompts
- We'll now begin to explore programs that use graphics and graphical user interfaces (GUIs)
- These programs use the JavaFX API
- JavaFX has replaced older approaches (AWT and Swing)

☺Intro to JavaFX (skip)

- 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 has the primary stage (window) as a parameter
- JavaFX embraces a theatre analogy
- See `HelloJavaFX.java`

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HelloJavaFX extends Application
{
    public void start(Stage primaryStage)
    {
        Text hello = new Text(50, 50, "Hello, JavaFX!");
        Text question = new Text(120, 80, "How's it going?");

        Group root = new Group(hello, question);
        Scene scene = new Scene(root, 300, 120, Color.LIGHTGREEN);
        primaryStage.setTitle("A JavaFX Program");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args)
    {
        launch(args);
    }
}
```

```
import javafx.application.Applicat
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HelloJavaFX extends A
{
    public void start(Stage primar
    {
        Text hello = new Text(50, 50, "Hello, JavaFX!");
        Text question = new Text(120, 80, "How's it going?");

        Group root = new Group(hello, question);
        Scene scene = new Scene(root, 300, 120, Color.LIGHTGREEN);
        primaryStage.setTitle("A JavaFX Program");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args)
    {
        launch(args);
    }
}
```

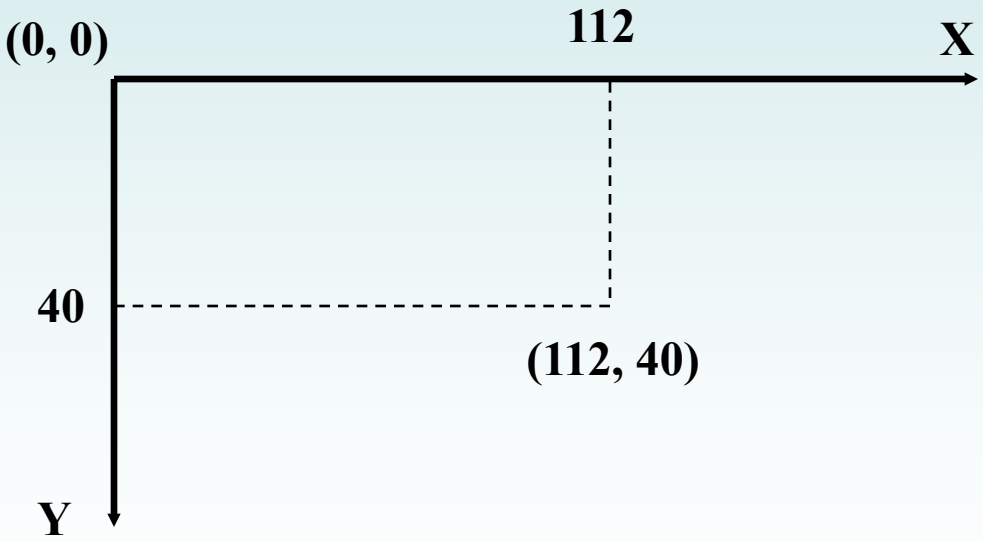


☺ Intro to JavaFX

- 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

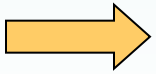
The Random and Math Classes

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 (of type `double`):

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

- For example:

```
Line myLine = new Line(10.1, 20.3,30.1,80.08);
```

☺ 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)
```

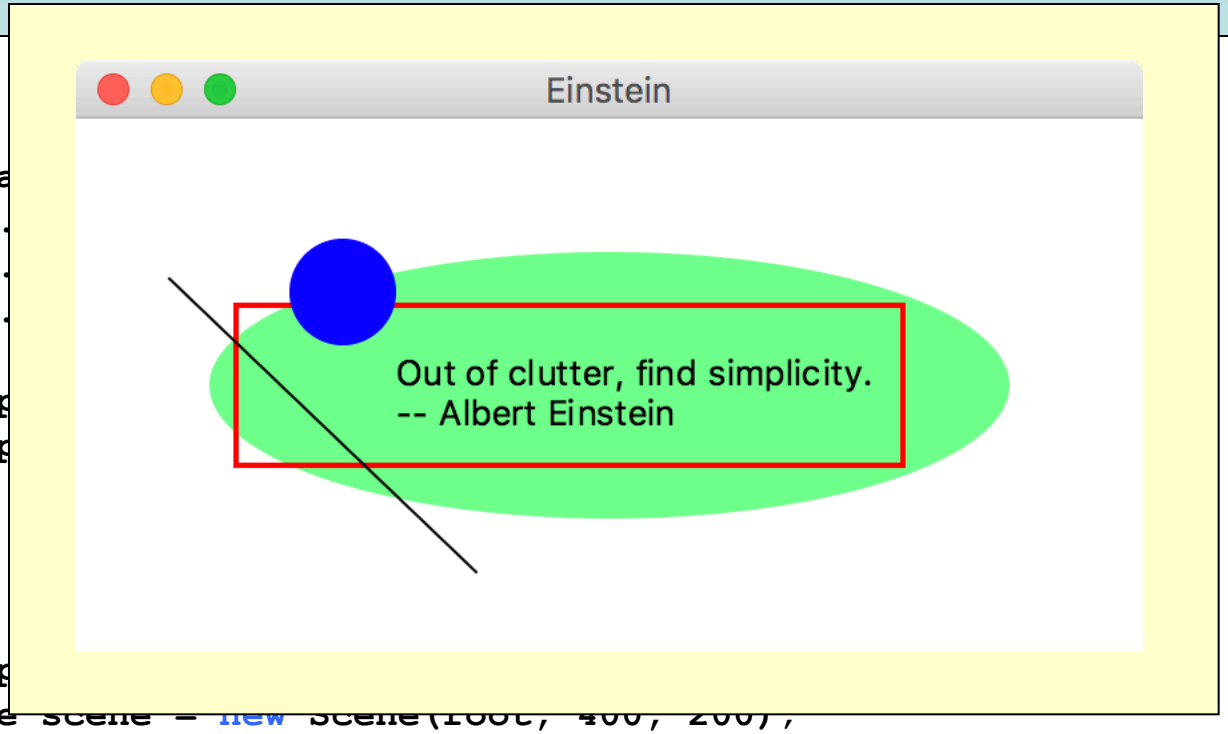
```
Ellipse e = new Ellipse(100, 50, 80, 30);
```

- 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`

```
//********************************************************************
//  Einstein.java       Author: Lewis/Loftus
//
//  Demonstrates the use of various shape classes.
//********************************************************************

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class Einstein extends Application
{
    //-----------------------------------------------------------------
    //  Creates and displays several shapes.
    //-----------------------------------------------------------------
    public void start(Stage primaryStage)
    {
        Line line = new Line(35, 60, 150, 170);

        Circle circle = new Circle(100, 65, 20);
        circle.setFill(Color.BLUE);
```

```
        Rectangle rect = new Rectangle(60, 70, 250, 60);
        rect.setStroke(Color.RED);
        rect.setStrokeWidth(2);
        rect.setFill(null);

        Ellipse ellipse = new Ellipse(200, 100, 150, 50);
        ellipse.setFill(Color.PALEGREEN);

        Text quote = new Text(120, 100, "Out of clutter, find " +
                "simplicity.\n-- Albert Einstein");

        Group root = new Group(ellipse, rect, circle, line, quote);
        Scene scene = new Scene(root, 400, 200);

        primaryStage.setTitle("Einstein");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    //  We will typically exclude the main method. Use it to launch
    //  the application if needed.
}
```

continued




```
        Recta
        rect.
        rect.
        rect.

        Ellip
        ellip

        Text


        Group
        Scene scene = new Scene(root, 400, 200);

        primaryStage.setTitle("Einstein");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    //  We will typically exclude the main method. Use it to launch
    //  the application if needed.
}
```

☺ 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`

```
//********************************************************************
//  Snowman.java       Author: Lewis/Loftus
//
//  Demonstrates the translation of a set of shapes.
//********************************************************************

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;

public class Snowman extends Application
{
    //-----------------------------------------------------------------
    //  Presents a snowman scene.
    //-----------------------------------------------------------------
    public void start(Stage primaryStage)
    {
        Ellipse base = new Ellipse(80, 210, 80, 60);
        base.setFill(Color.WHITE);

        Ellipse middle = new Ellipse(80, 130, 50, 40);
        middle.setFill(Color.WHITE);
```

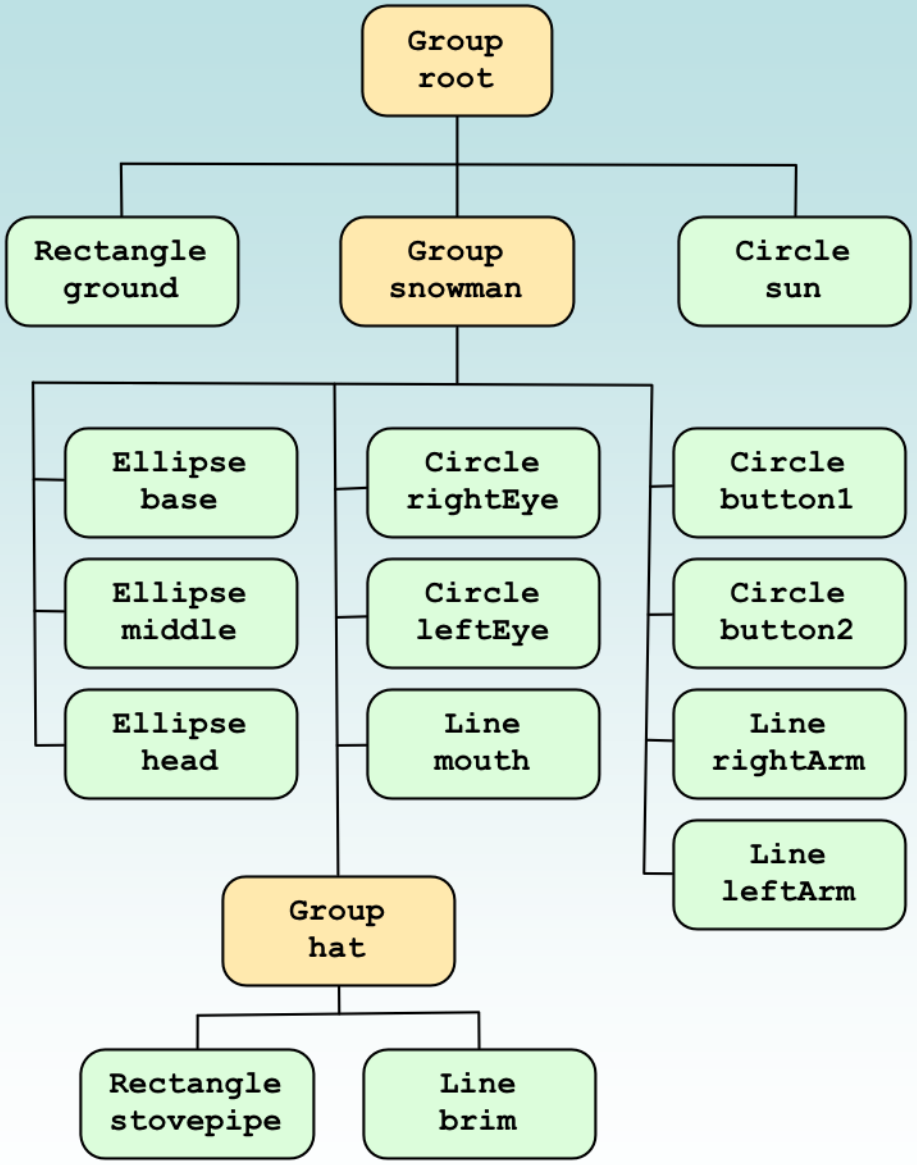
```
Circle head = new Circle(80, 70, 30);
head.setFill(Color.WHITE);

Circle rightEye = new Circle(70, 60, 5);
Circle leftEye = new Circle(90, 60, 5);
Line mouth = new Line(70, 80, 90, 80);

Circle topButton = new Circle(80, 120, 6);
topButton.setFill(Color.RED);
Circle bottomButton = new Circle(80, 140, 6);
bottomButton.setFill(Color.RED);

Line leftArm = new Line(110, 130, 160, 130);
leftArm.setStrokeWidth(3);
Line rightArm = new Line(50, 130, 0, 100);
rightArm.setStrokeWidth(3);

Rectangle stovepipe = new Rectangle(60, 0, 40, 50);
Rectangle brim = new Rectangle(50, 45, 60, 5);
Group hat = new Group(stovepipe, brim);
hat.setTranslateX(10);
hat.setRotate(15);
```

continued

```
        Group snowman = new Group(base, middle, head, leftEye, rightEye,
            mouth, topButton, bottomButton, leftArm, rightArm, hat);
        snowman.setTranslateX(170);
        snowman.setTranslateY(50);

        Circle sun = new Circle(50, 50, 30);
        sun.setFill(Color.GOLD);

        Rectangle ground = new Rectangle(0, 250, 500, 100);
        ground.setFill(Color.STEELBLUE);

        Group root = new Group(ground, sun, snowman);
        Scene scene = new Scene(root, 500, 350, Color.LIGHTBLUE);

        primaryStage.setTitle("Snowman");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

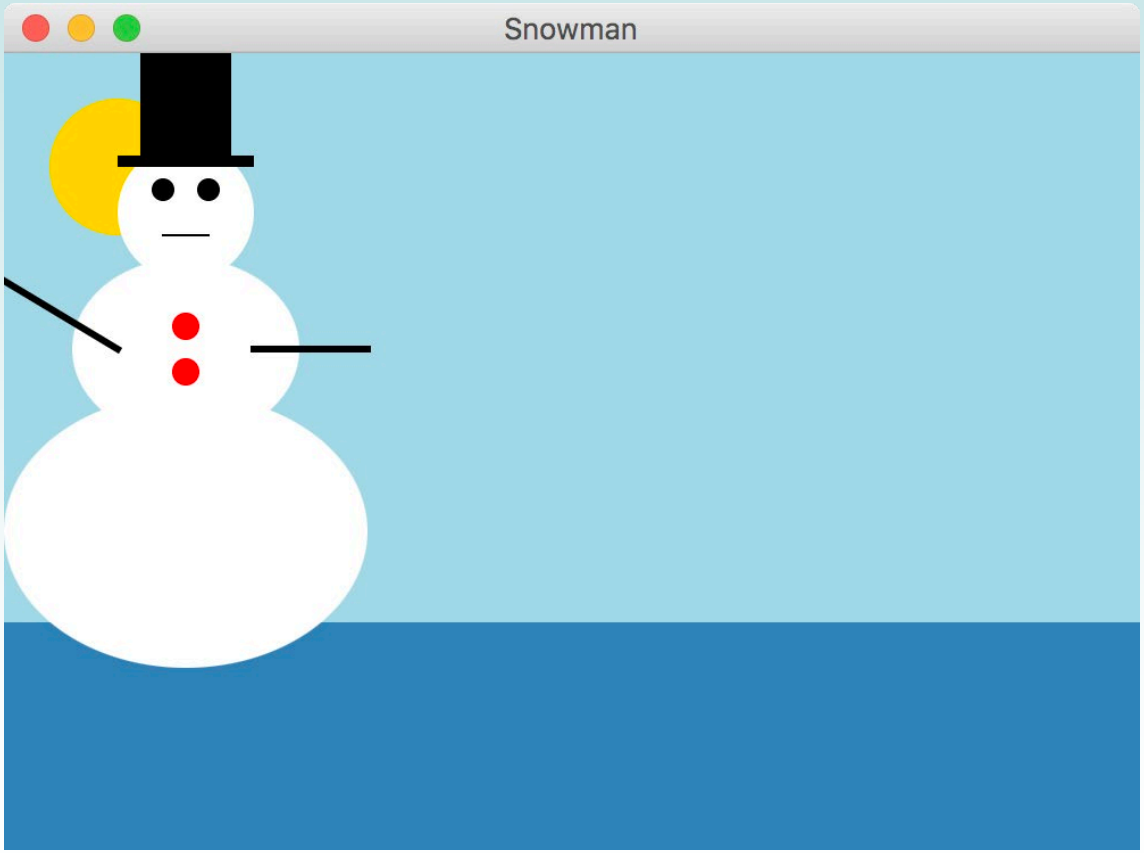
continu


```
    }
}
```



☺ 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 (very roughly)
- 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, many `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
  - the `Random` and `Math` classes
  - formatting output
  - enumerated types
  - wrapper classes
  - JavaFX graphics API
  - shape classes