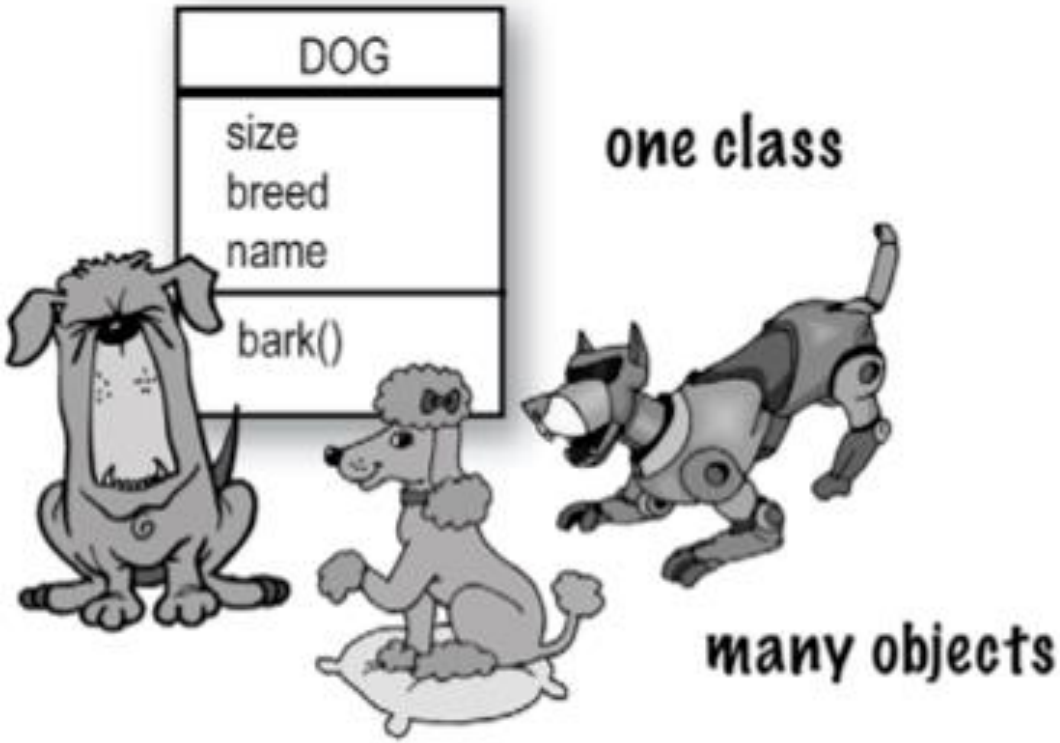
Java Class 5



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; initially that value is `null`

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");
```

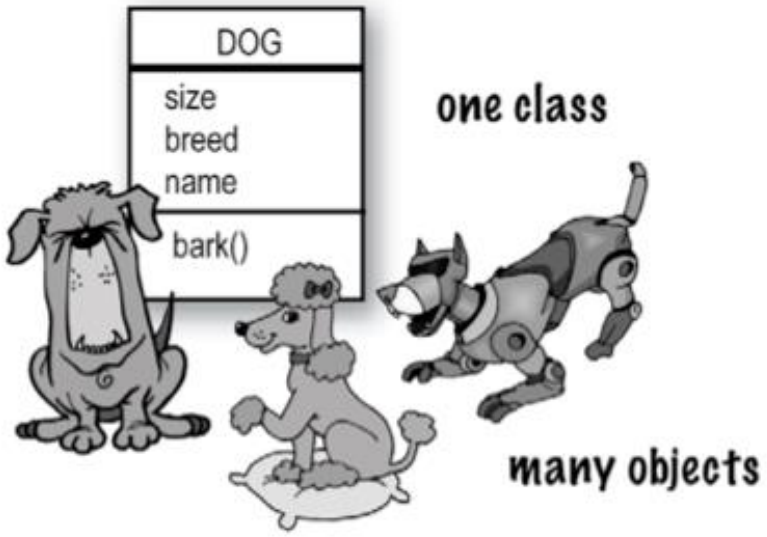
This calls the String *constructor*, which is a special method that sets up the object

Classes and Objects

A class is a blueprint from which multiple objects can be created

Each class declares:

- Instance variables: descriptive characteristics that determine the state of the object
- Methods: behaviors of the object, i.e., what it can do




```
public class Dog  {

    // instance variables
    int size;
    String bread;
    String name;

    // a method
    void bark() {
      System.out.println("Ruff! Ruff!");
    }
}
```

Creating Objects

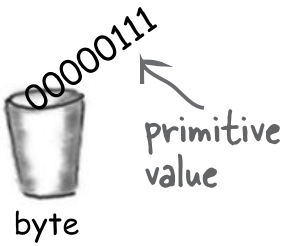
How to create objects once a class is established? How to "hold" those created objects?

For primitive types (such as `byte`, `int`, `double`), we use a variable of that particular type as a cup to store its actual value

Primitive Variable

```
byte x = 7;
```

The bits representing 7 go into the variable. (00000111).



Objects are different

- There is actually no such thing as an object variable
- There is only an object reference variable
- An object reference variable holds something like a pointer or an address that represents a way to get to the object

Creating Objects: Reference Variable

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

type → `Dog myDog;` ← name

No `Dog` object is created with this declaration

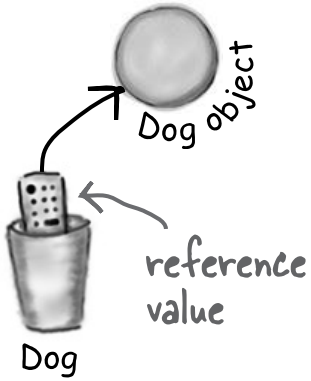
The object itself must be created separately using the `new` operator, but the object itself does not go into the variable

Reference Variable

```
Dog myDog = new Dog();
```

The bits representing a way to get to the Dog object go into the variable.

The Dog object itself does not go into the variable!

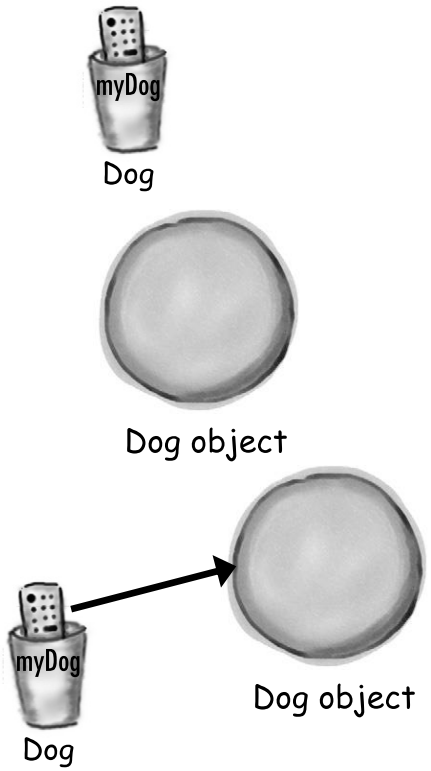


Creating Objects: Three Steps

- Three step of object declaration, creation and assignment

```
1          3  2
Dog myDog = new Dog();
```

- Step 1: allocate space for a `Dog` reference variable and name it `myDog`
- Step 2: allocate space for a new `Dog` object on the heap (a dynamic pool of memory)
- Step 3: link the `Dog` object and the `Dog` reference via assignment operator



The Dot Operator

The dot operator (.) gives you access to an object's instance variables (states) and methods (behaviors)

```
Dog myDog = new Dog();
myDog.bark();
```

- It says: use the `Dog` object referenced by the variable `myDog` to invoke the bark() method
- Think of a `Dog` reference variable as a `Dog` remote control
- When you use the dot operator on an object reference variable, it is like pressing a button on the remote control for that object

`System.out.println` revisited

- There is a class called `System`, which contains an object reference variable `out` as one of its instance variables
- The object referenced by `out` has a method called `println`

Testing the Dog Class

```
public class DogTestDrive  {

  public static void main (String[] args) {

    // Create a Dog object
    Dog myDog = new Dog();

    // Set the size of this Dog object
    myDog.size = 40;

    // Call its bark() method
    myDog.bark();

  }

}
```

Output

```
Ruff! Ruff!
```

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()
```

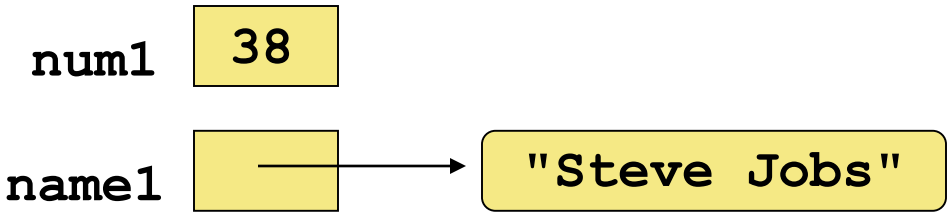
A method may *return a value*, which can be used in an assignment or expression

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

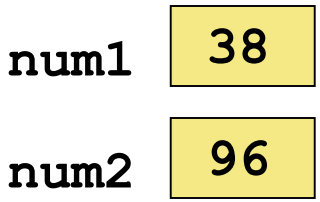


Assignment Revisited

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

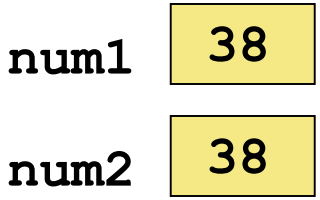
For primitive types:

Before:




```
num2 = num1;
```

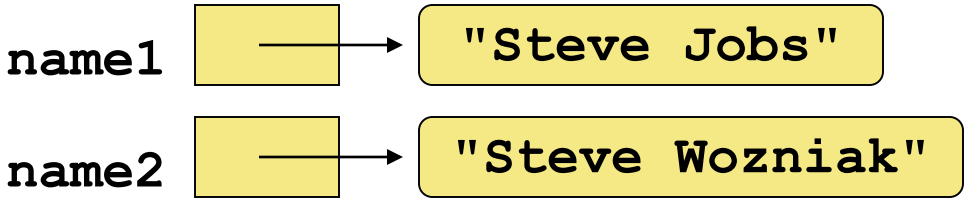
After:



Reference Assignment

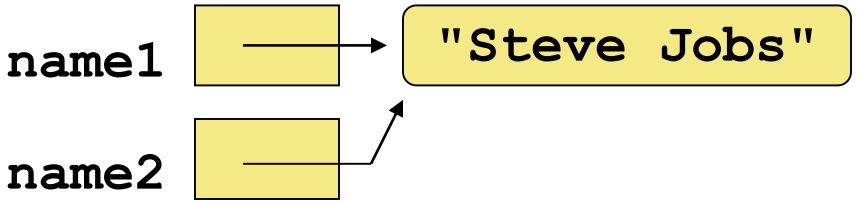
For object references, assignment copies the address:

Before:




```
name2 = name1;
```

After:



The String Class

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

```
String title = "Java Software Solutions";*

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

```
String title;
title = "Java Software Solutions";
```

This is special syntax that works only for strings *

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

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

See page 105 for methods in the `String` class

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

```
//********************************************************************
//  StringMutation.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the String class and its methods.
//********************************************************************

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());

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

continued

continued

```
        // 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
```

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

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:

- The JavaFX API
- The Database API

Packages

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

Examples:

Package	Purpose
java.lang	General support
java.text	Format text for output
java.awt	Graphics and graphical user interfaces
javax.io	Provides a wider variety of IO functions
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

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.*;
```

The import Declaration

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

Formatting Output

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
```

```
run:
Enter the coefficient of x squared: 3
Enter the coefficient of x: 8
Enter the constant: 4
Root #1: -0.667
Root #2: -2.000
BUILD SUCCESSFUL (total time: 42
seconds)
```

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

It is part of the `java.text` package

Formatting Output

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

`getCurrencyInstance()`
`getPercentInstance()`

```
NumberFormat fmt1=NumberFormat.getCurrencyInstance();

NumberFormat fmt2=NumberFormat.getPercentInstance();
```

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

```
System.out.println ("Subtotal: " +
fmt1.format(subtotal));
```

```
//********************************************************************
//  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);
```

continued

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));
    }
}
```

continued

```
        NumberFormat f                                    tance();
        NumberFormat f                                    ance();

        System.out.pri
        quantity = sca

        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));
    }
}
```

Sample Run

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

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

```
DecimalFormat fmt = new DecimalFormat ("0.###");
```

Formatting Parameters

```
DecimalFormat fmt = new DecimalFormat ("0.###");
```

0 means output a digit in that position

means output any nonzero digits with leading and trailing digits shown as absent

. (period) is the decimal separator

, (comma) is the grouping separator

```
//********************************************************************
//  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 * 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));
    }
}
```

Sample Run

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

continued

```
        System.ou
        radius =

        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));
    }
}
```

Group Exercises

Ex: 3.1

Ex: 3.2

Ex: 3.3

Ex: 3.4

Ex: 3.5

Assignment for Class 7

Review StringMutation, Purchase, CircleStats

Read Chapter 3.4, 3.5, 3.6, 3.7, 3.8

Begin Programming Project

(This is an individual assignment!)