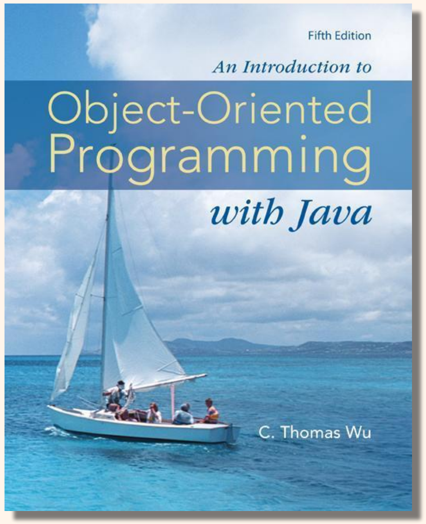


Chapter 2

Getting Started with Java

Objectives

After you have read and studied this chapter, you should be able to

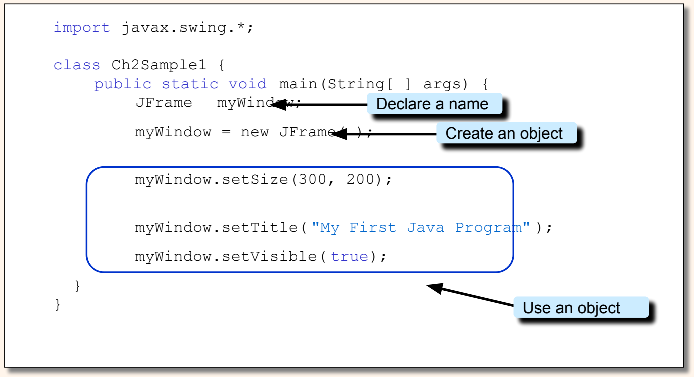
- Identify the basic components of Java programs
- Write simple Java programs
- Describe the difference between object declaration and creation
- Describe the process of creating and running Java programs
- Use the Date, SimpleDateFormat, String, and Scanner standard classes
- Develop Java programs, using the incremental development approach

The First Java Program

- The fundamental OOP concept illustrated by the program:

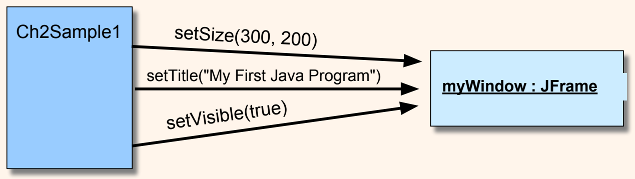
  An object-oriented program uses objects.
- This program displays a window on the screen.
- The size of the window is set to 300 pixels wide and 200 pixels high. Its title is set to My First Java Program.

Program Ch2Sample1

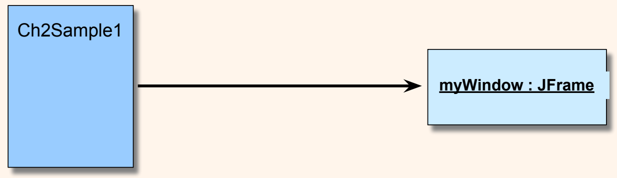


```
import javax.swing.*;

class Ch2Sample1 {
    public static void main(String[ ] args) {
        JFrame   myWindow;

        myWindow = new JFrame( );

        myWindow.setSize(300, 200);

        myWindow.setTitle("My First Java Program");

        myWindow.setVisible(true);
    }
}
```

Program Diagram for Ch2Sample1

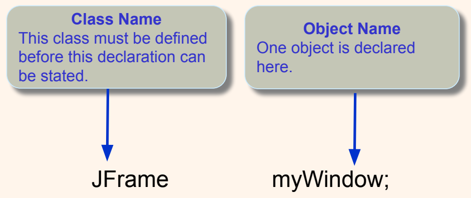


Dependency Relationship



Instead of drawing all messages, we summarize it by showing only the dependency relationship. The diagram shows that Ch2Sample1 "depends" on the service provided by myWindow.

Object Declaration

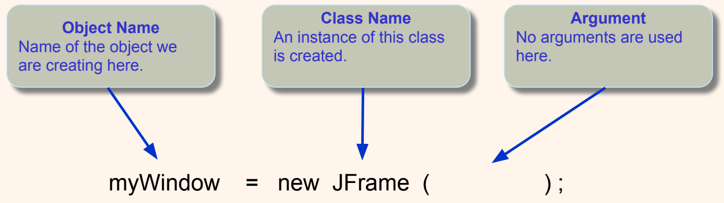

Class Name
This class must be defined before this declaration can be stated.
Object Name
One object is declared here.

```
JFrame        myWindow;
```

More Examples

```
Account customer;
Student jan, jim, jon;
Vehicle car1, car2;
```

Object Creation


Object Name
Name of the object we are creating here.
Class Name
An instance of this class is created.
Argument
No arguments are used here.

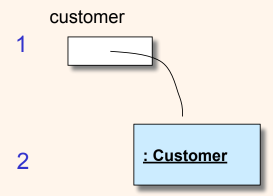
```
myWindow  =  new  JFrame (        ) ;
```

More Examples

```
customer = new Customer( );
jon= new Student("John Java");
car1  = new Vehicle( );
```

Declaration vs. Creation

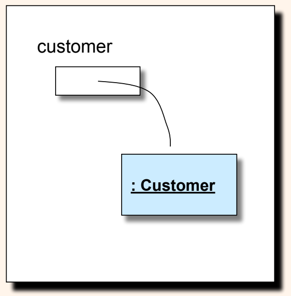
```
1  Customer   customer;
2  customer   =  new  Customer( );
```



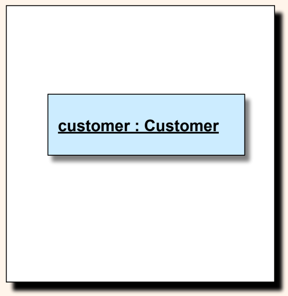
1. The identifier customer is declared and space is allocated in memory.

2. A Customer object is created and the identifier customer is set to refer to it.

State-of-Memory vs. Program



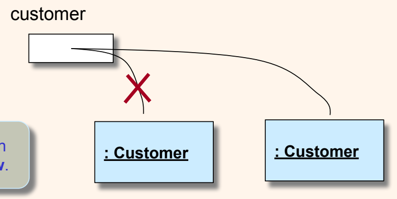
State-of-Memory Notation



Program Diagram Notation

Name vs. Objects

```
Customer   customer;
customer   =  new  Customer( );
customer   =  new  Customer( );
```



Created with the first **new**.

Created with the second **new**. Reference to the first Customer object is lost.

Sending a Message

Object Name
Name of the object to which we are sending a message.

Method Name
The name of the message we are sending.

Argument
The argument we are passing with the message.

myWindow . setVisible (true) ;

More Examples

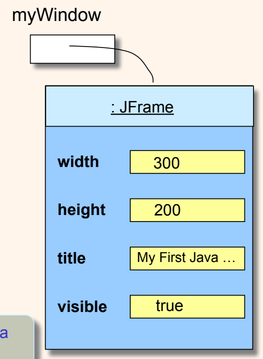
```
account.deposit( 200.0 );
student.setName("john");
car1.startEngine(  );
```

Execution Flow

Program Code

```
JFrame    myWindow;
myWindow  = new JFrame( );
myWindow.setSize(300, 200);
myWindow.setTitle
      ("My First Java Program");
myWindow.setVisible(true);
```

State-of-Memory Diagram



The diagram shows only four of the many data members of a JFrame object.

Program Components

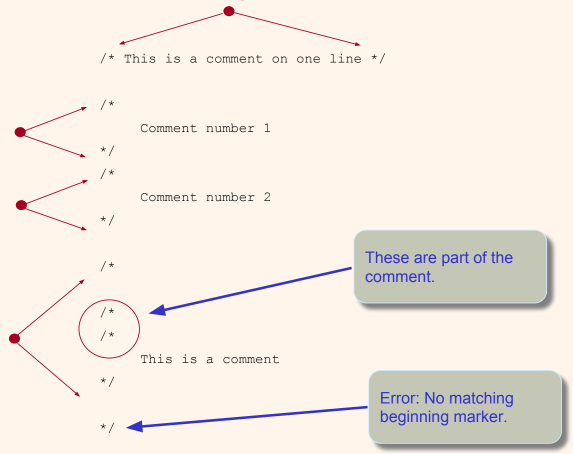
- A Java program is composed of
  - comments,
  - import statements, and
  - class declarations.

Program Component: Comment

```
/*
        Chapter 2 Sample Program: Displaying a Window

        File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
    public static void main(String[ ] args) {
        JFrame    myWindow;
        myWindow = new JFrame( );

        myWindow.setSize(300, 200);

        myWindow.setTitle("My First Java Program");
        myWindow.setVisible(true);

    }
}
```

Comment

Matching Comment Markers



```
/* This is a comment on one line */

/*
     Comment number 1
*/
/*
     Comment number 2
*/

/*

/*
/*
     This is a comment
*/

*/
```

These are part of the comment.

Error: No matching beginning marker.

Three Types of Comments

```
/*
    This is a comment with
    three lines of
    text.
*/
```

Multiline Comment

```
// This is a comment
// This is another comment
// This is a third comment
```

Single line Comments

```
/**
 * This class provides basic clock functions. In addition
 * to reading the current time and today's date, you can
 * use this class for stopwatch functions.
 */
```

javadoc Comments

Import Statement

```
/*
        Chapter 2 Sample Program: Displaying a Window

        File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
    public static void main(String[ ] args) {
        JFrame   myWindow;

        myWindow = new JFrame( );


        myWindow.setSize(300, 200);


        myWindow.setTitle("My First Java Program");

        myWindow.setVisible(true);

    }
}
```

Import Statement

Import Statement Syntax and Semantics

Package Name
Name of the package that contains the classes we want to use.

Class Name
The name of the class we want to import. Use asterisks to import all classes.

<package name> . <class name> ;

e.g. dorm . Resident;

More Examples

```
import      javax.swing.JFrame;
import      java.util.*;
import      com.drcaffeine.simplegui.*;
```

Class Declaration

```
/*
        Chapter 2 Sample Program: Displaying a Window

        File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
    public static void main(String[ ] args) {
        JFrame   myWindow;

        myWindow = new JFrame( );


        myWindow.setSize(300, 200);


        myWindow.setTitle("My First Java Program");

        myWindow.setVisible(true);

    }
}
```

Class Declaration

Method Declaration

```
/*
        Chapter 2 Sample Program: Displaying a Window

        File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1 {
    public static void main(String[ ] args) {
        JFrame   myWindow;

        myWindow = new JFrame( );

        myWindow.setSize(300, 200);

        myWindow.setTitle("My First Java Program");

        myWindow.setVisible(true);
    }
}
```

Method Declaration

Method Declaration Elements

Modifier | Modifier | Return Type | Method Name | Parameter

```
public    static    void       main(    String[ ] args  ){
     JFrame myWindow;

     myWindow = new JFrame( );

     myWindow.setSize(300, 200);

     myWindow.setTitle("My First Java Program");

     myWindow.setVisible(true);
}
```

Method Body

Template for Simple Java Programs

```
/*
   Chapter 2 Sample Program: Displaying a Window

   File: Ch2Sample2.java
*/

import javax.swing.*;

class Ch2Sample1                           {

    public static void main(String[ ] args) {
        JFrame   myWindow;

        myWindow = new JFrame( );

        myWindow.setSize(300, 200);

        myWindow.setTitle(
                "My First Java Program");

        myWindow.setVisible(true);
    }
}
```

Comment

Import Statements

Class Name

Method Body

Why Use Standard Classes

- **Don't reinvent the wheel**. When there are existing objects that satisfy our needs, use them.
- **Learning how to use standard Java classes** is the first step toward mastering OOP. Before we can learn how to define our own classes, we need to learn how to use existing classes
- We will introduce four standard classes here:
  - Scanner
  - String
  - Date
  - SimpleDateFormat.

Standard Output

- Using **print** of **System.out** (an instance of the **PrintStream** class) is a simple way to display a result of a computation to the user.

```
System.out.print("I Love Java");
```

```
I Love Java
```

The result appears on the console window. The actual appearance of the console window differs depending on the Java tool you use

Using the print Method

- The **print** method will continue printing from the end of the currently displayed output.

```
System.out.print("How do you do? ");
System.out.print("My name is ");
System.out.print("Jon Java. ");
```

```
How do you do? My name is Jon
Java.
```

Using the println Method

- The **println** method will skip to the next line after printing out its argument.

```
System.out.println("How do you do? ");
System.out.println("My name is ");
System.out.println("Jon Java. ");
```

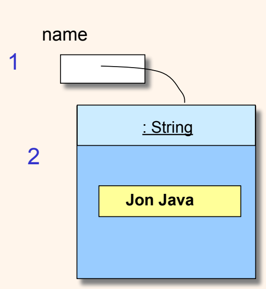
```
How do you do?
My name is
Jon Java.
```

String

- The textual values passed to the showMessageDialog method are instances of the String class.
- A sequence of characters separated by double quotes is a String constant.
- There are close to 50 methods defined in the String class. We will introduce three of them here: substring, length, and indexOf.
- We will also introduce a string operation called concatenation.

String is an Object

```
1  String   name;
2  name  =  new  String("Jon Java");
```

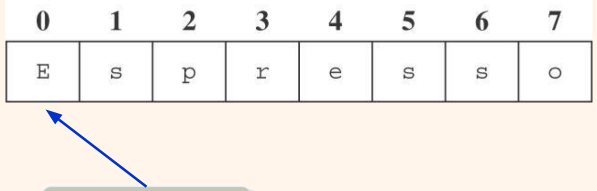


1. The identifier name is declared and space is allocated in memory.

2. A String object is created and the identifier name is set to refer to it.

String Indexing

```
String text;
text = "Espresso";
```



The position, or index, of the first character is 0.

Definition: substring

- Assume str is a String object and properly initialized to a string.
- str.substring(i, j) will return a new string by extracting characters of str from position i to j-1 where 0 ≤ i • length of str, 0 • j ≤ length of str, and i ≤ j.
- If str is "programming" , then str.substring(3, 7) will create a new string whose value is "gram" because g is at position 3 and m is at position 6.
- The original string str remains unchanged.

Examples: substring

```
String text = "Espresso";
```

```
text.substring(6,8)   ——→  "so"
text.substring(0,8)   ——→  "Espresso"
text.substring(1,5)   ——→  "spre"
text.substring(3,3)   ——→  ""
text.substring(4,2)   ——→  error
```

Definition: length

- Assume str is a String object and properly initialized to a string.
- str.length( ) will return the number of characters in str.
- If str is "programming" , then str.length( ) will return 11 because there are 11 characters in it.
- The original string str remains unchanged.

Examples: length

```
String str1, str2, str3, str4;
str1 = "Hello" ;
str2 = "Java" ;
str3 = "" ; //empty string
str4 = " " ; //one space
```

```
str1.length( )  →  5
str2.length( )  →  4
str3.length( )  →  0
str4.length( )  →  1
```

Definition: indexOf

- Assume str and substr are String objects and properly initialized.
- str.indexOf(substr) will return the first position substr occurs in str.
- If str is "programming" and substr is "gram" , then str.indexOf(substr) will return 3 because the position of the first character of substr in str is 3.
- If substr does not occur in str, then –1 is returned.
- The search is case-sensitive.

Examples: indexOf

```
String str;
str = "I Love Java and Java loves me." ;
```

3 7 21

```
str.indexOf( "J" )      →  7
str.indexOf( "love" )   →  21
str. indexOf( "ove" )   →  3
str. indexOf( "Me" )    →  -1
```

Definition: concatenation

- Assume str1 and str2 are String objects and properly initialized.
- str1 + str2 will return a new string that is a concatenation of two strings.
- If str1 is "pro" and str2 is "gram" , then str1 + str2 will return "program".
- Notice that this is an operator and not a method of the String class.
- The strings str1 and str2 remains the same.

Examples: concatenation

```
String str1, str2;
str1 = "Jon" ;
str2 = "Java" ;
```

```
str1 + str2                  →  "JonJava"

str1 + " " + str2            →  "Jon Java"

str2 + ", " + str1           →  "Java, Jon"

"Are you " + str1 + "?"      →  "Are you Jon?"
```

Date

- The Date class from the java.util package is used to represent a date.
- When a Date object is created, it is set to today (the current date set in the computer)
- The class has toString method that converts the internal format to a string.

```
Date today;
today = new Date( );

today.toString( );

      →  "Fri Feb 10 14:43:14 WET 2017"
```

SimpleDateFormat

- The SimpleDateFormat class from the **java.text** package allows the Date information to be displayed with various format.
- Table 2.1 page 62 shows the formatting options.

```
import java.text.*;
Date today = new Date( );
SimpleDateFormat sdf1, sdf2;
sdf1 = new SimpleDateFormat( "MM/dd/yy" );
sdf2 = new SimpleDateFormat( "MMMM dd, yyyy" );

sdf1.format(today);  →  "12/18/08"

sdf2.format(today);  →  "December 19, 2008"
```

Standard Input

- Using a **Scanner** object is a simple way to input data from the standard input **System.in**, which accepts input from the keyboard.
- First we need to associate a Scanner object to System.in as follows:

```
import java.util.Scanner;

Scanner scanner;

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

Reading from Standard Input

- After the Scanner object is set up, we can read data.
- The following inputs the first name (String):

```
System.out.print ("Enter your first name: ");
String firstName = scanner.next();
System.out.println("Nice to meet you, " +
                   firstName + ".");
```

```
Enter your first name: George ENTER
Nice to meet you, George.
```

1. Prompt is displayed
2. Data is entered
3. Result is printed

Problem Statement

- Problem statement:

  Write a program that asks for the user's first, middle, and last names and replies with their initials.

  Example:

input:	Andrew Lloyd Weber
output:	ALW

Overall Plan

- Identify the major tasks the program has to perform.
  - We need to know what to develop before we develop!
- Tasks:
  - Get the user's first, middle, and last names
  - Extract the initials and create the monogram
  - Output the monogram

Development Steps

- We will develop this program in two steps:

1. Start with the program template and add code to get input

2. Add code to compute and display the monogram

Step 1 Design

- The program specification states "get the user's name" but doesn't say how.
- We will consider "how" in the Step 1 design
- We will use JOptionPane for input
- Input Style Choice #1

  Input first, middle, and last names separately
- Input Style Choice #2

  Input the full name at once
- We choose Style #2 because it is easier and quicker for the user to enter the information

Step 1 Code

```
/*
    Chapter 2 Sample Program: Displays the Monogram
    File: Step1/Ch2Monogram.java
*/
import javax.swing.*;

class Ch2Monogram {
    public static void main (String[ ] args) {
        String name;
        name = JOptionPane.showInputDialog(null,
               "Enter your full name (first, middle, last):");
        JOptionPane.showMessageDialog(null, name);
    }
}
```

Step 1 Test

- In the testing phase, we run the program and verify that
  - we can enter the name
  - the name we enter is displayed correctly

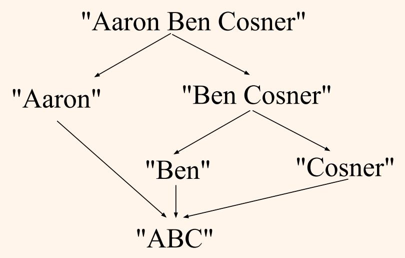
Step 2 Design

- Our programming skills are limited, so we will make the following assumptions:
  - input string contains first, middle, and last names
  - first, middle, and last names are separated by single blank spaces
- Example

John Quincy Adams	(okay)
John Kennedy	(not okay)
Harrison, William Henry	(not okay)

Step 2 Design (cont'd)

- Given the valid input, we can compute the monogram by
  - breaking the input name into first, middle, and last
  - extracting the first character from them
  - concatenating three first characters



Step 2 Code

```
/*
    Chapter 2 Sample Program: Displays the Monogram
    File: Step 2/Ch2MonogramStep2.java
*/
import javax.swing.*;

class Ch2Monogram {
    public static void main (String[ ] args) {
        String name, first, middle, last, space, monogram;

        space = " ";
        //Input the full name
        name = JOptionPane.showInputDialog (null,
          "Enter your full name (first, middle, last):");
```

Step 2 Code (cont'd)

```
        //Extract first, middle, and last names
        first = name.substring(0, name.indexOf(space));
        name = name.substring(name.indexOf(space)+1,
                name.length());

        middle = name.substring(0, name.indexOf(space));
        last = name.substring(name.indexOf(space)+1,
                name.length());

        //Compute the monogram
        monogram = first.substring(0, 1) +
                middle.substring(0, 1) + last.substring(0,1);
        //Output the result
        JOptionPane.showMessageDialog(null,
                "Your monogram is " + monogram);
    }
}
```

Step 2 Test

- In the testing phase, we run the program and verify that, for all valid input values, correct monograms are displayed.
- We run the program numerous times. Seeing one correct answer is not enough. We have to try out many different types of (valid) input values.

Program Review

- The work of a programmer is not done yet.
- Once the working program is developed, we perform a critical review and see if there are any missing features or **possible improvements**
- One suggestion
  - Improve the initial prompt so the user knows the valid input format requires single spaces between the first, middle, and last names