Week 5

(Chapter 6)

Introducing Methods

Classes have been used since the beginning of this class. However, until now, only the most rudimentary form of a class has been used. The classes created have primarily ***encapsulated*** the **init**, **paint**, and **main** methods, which have been used to demonstrate the basics of the Java syntax. As you will see (in chapters 8 & 9), methods and classes are substantially more powerful than the limited ones presented to far. Here in chapter 6, my focus is on how to create simple methods that can fit inside a class.

A method is conceptually similar to the subroutines or functions that you have seen in other programming languages. It is designed to receive data, process the data, and return a result. Sometimes methods are designed to do something that requires no input data, or output data (resulting in the use of the word **void**).

A class encapsulates (i.e., contains) two things: instance variables and methods. The topic of methods is a large one because Java gives them so much power and flexibility.

Here is the basic layout of a class:

class classname ***{***

type instance-variable1;

type instance-variable2;

*// …*

*type instance-variableN;*

***type*** *methodname1(parameter-list)* ***{***

# // body of method

***}***

***type*** *methodname2(parameter-list)* ***{***

# // body of method

***}***

***type*** *methodnameN(parameter-list)* ***{***

# // body of method

***}***

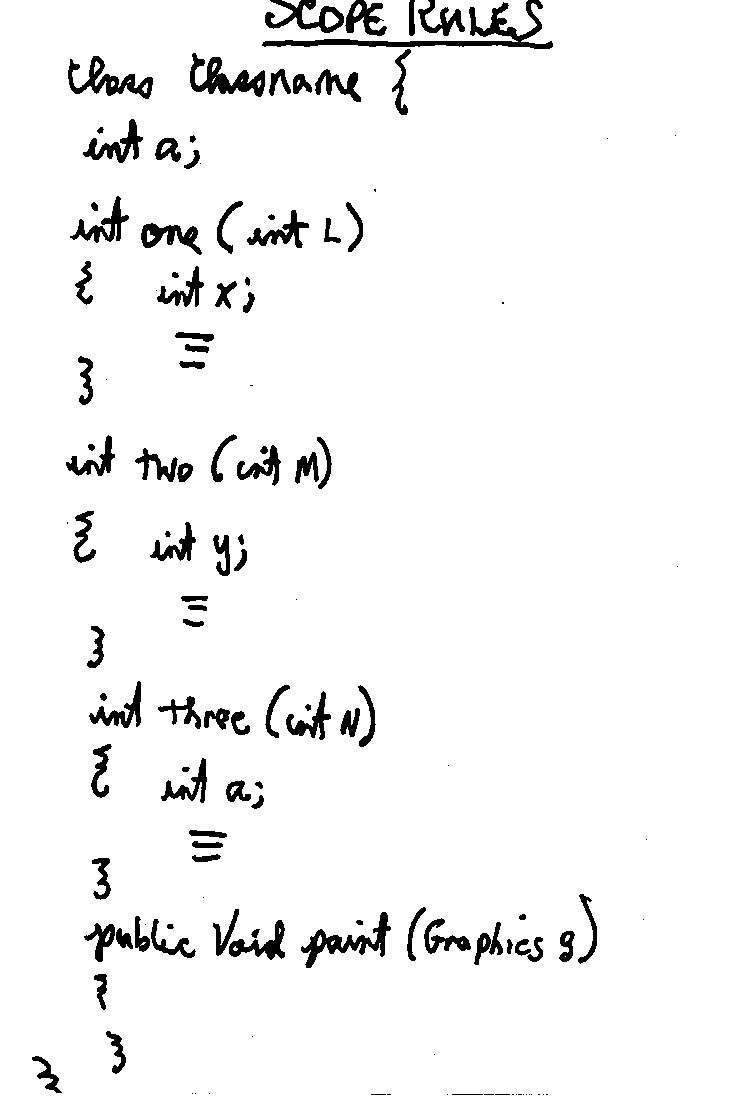
***}***

Here, ***type*** specifies the type of data returned by the method. If the method does not return a value, its return type must be **void**. The parameter-list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

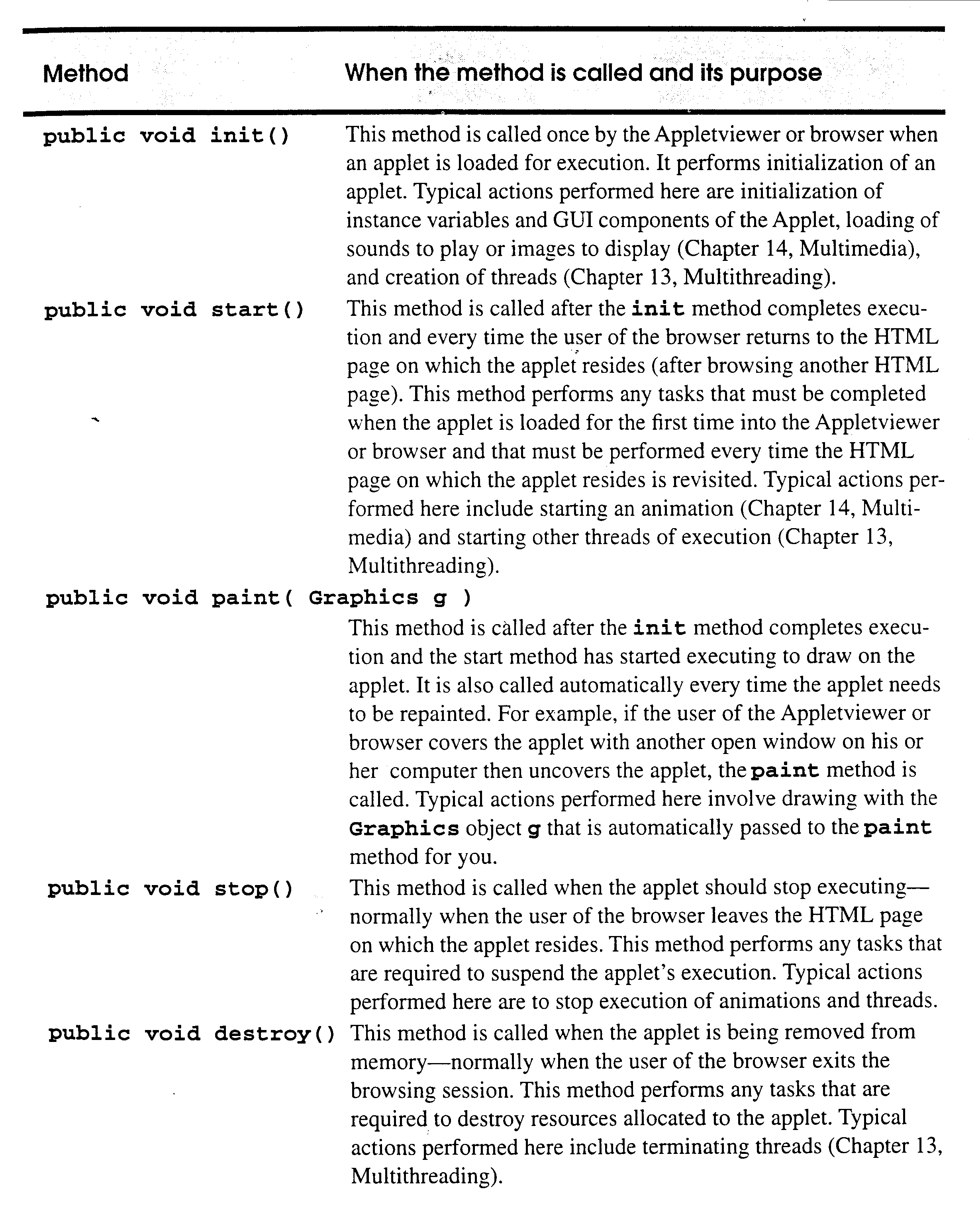
Methods that have a return type other than **void** return a value to the calling routine using the following form of the **return** statement:

return value

## First, a discussion on……



In Week 3, I discussed the key methods of the Applet class that are called automatically during the execution of an applet.



These Applet methods are defined by the Java API to do nothing unless you provide a definition in your applet’s class definition. You must define the first line of the method as shown above. Otherwise, the method will not get called automatically during the execution of the applet.

## Example 1

import java.awt.Graphics;

import java.applet.Applet;

public class BoxApp extends Applet {

double width;

double height;

double depth;

public void init( )

{ width = 3;

height = 6;

depth = 9;

}

public void paint (Graphics g)

{

g.drawString(“Volume =” + volume( ), 25, 25);

}

double volume( )

{

return width \* height \* depth;

}

}

## Example 2

import java.awt.Graphics;

import java.applet.Applet;

public class BoxApp2 extends Applet {

double width;

double height;

double depth;

void initialize(int x, int y, int z) // Must use int 3 times

{

width = x;

height = y;

depth = z;

}

public void paint (Graphics g)

{ initialize(3,6,9);

g.drawString(“Volume =” + volume( ), 25, 25);

}

double volume( )

{

return width \* height \* depth;

}

}

## Example 3

import java.awt.Graphics;

import java.applet.Applet;

public class BoxApp2 extends Applet {

double width;

double height;

double depth;

void initialize(int x, int y, int z) // Must use int 3 times

{ int width, height, depth; // Notice added declaration

width = x;

height = y;

depth = z;

}

public void paint (Graphics g)

{ initialize(3,6,9);

g.drawString(“Volume =” + volume( ), 25, 25);

}

double volume( )

{

return width \* height \* depth;

}

}

## Example 4

import java.awt.Graphics;

import java.applet.Applet;

public class BoxApp2 extends Applet {

double width;

double height;

double depth;

public void init(int x, int y, int z) // Overloaded Method (kind of)

{

width = x;

height = y;

depth = z;

}

public void paint (Graphics g)

{ init(3,6,9);

g.drawString(“Volume =” + volume(), 25, 25);

}

double volume( )

{

return width \* height \* depth;

}

}

## Example 5

import java.awt.Graphics;

import Java.applet.Applet;

public class Scoping extends Applet {

int x = 1; // instance variable

public void paint (Graphics g)

{

g.drawString(“See command line for output”, 25,25);

int x = 5; // local variable to paint

System.out.println(“local variable to paint is ” + x);

a( ); // a has automatic local x

b( ); // b uses instance variable x

a( ); // a reinitializes automatic local x

b( ); // instance variable x retains its value

System.out.println(“\nlocal x in paint is ” + x);

}

void a( )

{ int x = 25; // initialized each time a is called

System.out.println( “\nlocal x in a is ” + x + “after entering a ”);

++x;

System.out.println(“local x in a is ” + x + “before exiting a ”);

}

void b( )

{ System.out.println(“\ninstance variable x is ”+x+“ on entering b ”);

x \*= 10;

System.out.println(“instance variable x is ” + x + “ on exiting b ”);

}

}

## Example 6 - Recursion

The following code computes the sum of the numbers from 1 to 5 using a while loop:

sum = 0;

count = 1;

while (count <= 5)

sum = sum + count++;

The following code computes the sum of the numbers from 1 to 5 using a for loop:

for (int sum=0, int count=1; count<=5; sum=sum+count++) ;

**Here’s a recursive solution to this problem:**

public int sum(int current, int ending)

{

if (current = = ending)

return ending;

else

return current + sum(current+1, ending);

}

Here’s the calling statement:

System.out.println(“The sum of the #’s from 1 to 5=” + sum(1,5));

## Example 7

A simple coding example illustrating where we’ll be going with classes next week:

class Box {

double width;

double height;

double depth;

double volume( ) {

return width \* height \* depth;

}

}

class BoxDemo {

public static void main(String args[ ]) {

double vol;

Box first;

first = new Box( );

first.width = 3;

first.height = 6;

first.depth = 9;

System.out.println(“Volume =” + first.volume( ));

}

}

## Programming Assignment 6

The greatest common divisor (GCD) of two integers is the largest integer that evenly divides each of the two numbers. For example, the greatest common divisor of 12 and 18 is 6.

Write a method titled **gcd** that accepts two integers (as input parameters to the method), and **returns** the greatest common divisor of the two integers. Incorporate the method into an applet that obtains two integers from the user using **TextField**s. Display the result of **gcd** in the status bar, **and** on the applet window.

Run multiple tests, and submit a complete .java printout, and a display of multiple windows that clearly demonstrates that your applet works.

An aside, it makes good sense to call the **gcd** method at the point you are preparing your output message. You should not call **gcd** from your actionPerformed method.