Data and Expressions

Chapter 2

5TH EDITION

Lewis & Loftus

java

Software Solutions

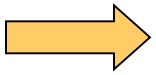
Foundations of Program Design

PEARSON Addison Wesley

Data and Expressions

- **Let's explore some other fundamental programming concepts**
- **Chapter 2 focuses on:**
  - **character strings**
  - **primitive data**
  - **the declaration and use of variables**
  - **expressions and operator precedence**
  - **data conversions**
  - **accepting input from the user**
  - **Java applets**
  - **introduction to graphics**

Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

Graphics

Applets

Drawing Shapes

Character Strings

- **A string of characters can be represented as a *string literal* by putting double quotes around the text:**
- **Examples:**

```
"This is a string literal."
"123 Main Street"
"X"
```

- **Every character string is an object in Java, defined by the `String` class**
- **Every string literal represents a `String` object**

The println Method

- **In the `Lincoln` program from Chapter 1, we invoked the `println` method to print a character string**
- **The `System.out` object represents a destination (the monitor screen) to which we can send output**

```
System.out.println ("Whatever you are, be a good one.");
```

in C++, they use
cout << 'Hello' << endl;

object

method name

information provided to the method (parameters)

The print Method

- **The `System.out` object provides another service as well**

javac Countdown.java
java Countdown

- **The `print` method is similar to the `println` method, except that it does not advance to the next line**
- **Therefore anything printed after a `print` statement will appear on the same line**
- **See Countdown.java (page 61)**

```
//********************************************************************
//  Countdown.java       Author: Lewis/Loftus
//
//  Demonstrates the difference between print and println.
//********************************************************************
public class Countdown
{
   //-----------------------------------------------------------------
   //  Prints two lines of output representing a rocket
countdown.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.print ("Three... ");
      System.out.print ("Two... ");
      System.out.print ("One... ");
      System.out.print ("Zero... ");

      System.out.println ("Liftoff!"); // appears on first output line

      System.out.println ("Houston, we have a problem.");
   }
}
```

String Concatenation

- **The *string concatenation operator* (+) is used to append one string to the end of another**

```
"Peanut butter " + "and jelly"
```

- **It can also be used to append a number to a string**
- **A string literal cannot be broken across two lines in a program**
- **See Facts.java (page 63)**

```
//********************************************************************
//  Facts.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the string concatenation operator and the
//  automatic conversion of an integer to a string.
//********************************************************************

public class Facts
{
  //-----------------------------------------------------------------
  //  Prints various facts.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    // Strings can be concatenated into one long string
    System.out.println ("We present the following facts for your "
                        + "extracurricular edification:");

    System.out.println ();

    // A string can contain numeric digits
    System.out.println ("Letters in the Hawaiian alphabet: 12");

    // A numeric value can be concatenated to a string
    System.out.println ("Dialing code for Antarctica: " + 672);

    System.out.println ("Year in which Leonardo da Vinci invented "
                        + "the parachute: " + 1515);

    System.out.println ("Speed of ketchup: " + 40 + " km per year");
  }
}
```

String Concatenation

- **The + operator is also used for arithmetic addition**

  1 + 2 = 3

  "Cat" + "fish" = "Catfish"

  40 + "winks" = "40" + "winks" = "40winks"

  adding number and string, the number becomes a string

- **The function that it performs depends on the type of the information on which it operates**
- **If both operands are strings, or if one is a string and one is a number, it performs string concatenation**
- **If both operands are numeric, it adds them**
- **The + operator is evaluated left to right, but parentheses can be used to force the order**
- **See Addition.java (page 64)**

break

```
//********************************************************************
// Addition.java       Author: Lewis/Loftus
//
// Demonstrates the difference between the addition and string
// concatenation operators.
//********************************************************************
public class Addition
{
  //-----------------------------------------------------------------
  // Concatenates and adds two numbers and prints the results.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    System.out.println ("24 and 45 concatenated: " + 24 + 45);

    System.out.println ("24 and 45 added: " + (24 + 45));
  }
}
```

Escape Sequences

- **What if we wanted to print a the quote character?**
- **The following line would confuse the compiler because it would interpret the second quote as the end of the string**

```
System.out.println ("I said "Hello" to you.");
```

- **An *escape sequence* is a series of characters that represents a special character**
- **An escape sequence begins with a backslash character (\)**

```
System.out.println ("I said \"Hello\" to you.");
```

need to use double back slash \\ if typing out pathnames in DOS(windows)

Escape Sequences

- **Some Java escape sequences:**

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

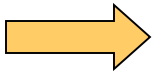
\v - vertical tab

enter key (goes to the new line and then goes to the beginning) (this is 2 separate steps)

- **See Roses.java (page 66)**

```
//********************************************************************
//  Roses.java       Author: Lewis/Loftus
//
//  Demonstrates the use of escape sequences.
//********************************************************************

public class Roses
{
   //-----------------------------------------------------------------
   //  Prints a poem (of sorts) on multiple lines.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("Roses are red,\n\tViolets are blue,\n" +
         "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
         "So I'd rather just be friends\n\tAt this point in our " +
         "relationship.");
   }
}
```

Outline



Character Strings

Variables and Assignment

look at the assignment

You MUST use javadoc style comments for all assignments
top of every method and class

Primitive Data Types

have to download JDK, can't do the hw aassignments in hills.... lame

Expressions

Data Conversion

Interactive Programs

Graphics

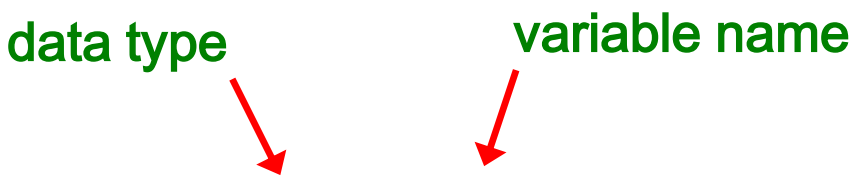
Applets

Drawing Shapes

8/29/17 end

Variables

- **A *variable* is a name for a location in memory**
- **A variable must be *declared* by specifying the variable's name and the type of information that it will hold**




```
int total;

int count, temp, result;
```

not a good style because one declaration per line is better - you can comment on what it does that way...

Multiple variables can be created in one declaration

Variable Initialization

- **A variable can be given an initial value in the declaration**

allocates memory and then initializes it

```
int sum = 0;
int base = 32, max = 149;
```

- **When a variable is referenced in a program, its current value is used**

- **See PianoKeys.java (page 68)**

```
//********************************************************************
//  PianoKeys.java       Author: Lewis/Loftus
//
//  Demonstrates the declaration, initialization, and use of an
//  integer variable.
//********************************************************************

public class PianoKeys
{
   //-----------------------------------------------------------------
   //  Prints the number of keys on a piano.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int keys = 88;

      System.out.println ("A piano has " + keys + " keys.");
   }
}
```

Assignment

- **An *assignment statement* changes the value of a variable**
- **The assignment operator is the = sign**

int total = 0

```
total = 55;
```

primitives can only hold one value at a time

- **The expression on the right is evaluated and the result is stored in the variable on the left**
- **The value that was in `total` is overwritten**
- **You can only assign a value to a variable that is consistent with the variable's declared type**
- **See Geometry.java (page 69)**

```
//********************************************************************
//  Geometry.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an assignment statement to change the
//  value stored in a variable.
//********************************************************************

public class Geometry
{
   //-----------------------------------------------------------------
   //  Prints the number of sides of several geometric shapes.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int sides = 7;  // declaration with initialization
      System.out.println ("A heptagon has " + sides + " sides.");

      sides = 10;  // assignment statement
      System.out.println ("A decagon has " + sides + " sides.");

      sides = 12;
      System.out.println ("A dodecagon has " + sides + " sides.");
   }
}
```

Constants

- **A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence**
- **As the name implies, it is constant, not variable**
- **The compiler will issue an error if you try to change the value of a constant**
- **In Java, we use the `final` modifier to declare a constant**

```
final int MIN_HEIGHT = 69;
```

this is the perferred constant keyword
final contants are in UPPER CASE

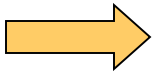
Constants

skipped

- **Constants are useful for three important reasons**
- **First, they give meaning to otherwise unclear literal values**
  - **For example, MAX_LOAD means more than the literal 250**
- **Second, they facilitate program maintenance**
  - **If a constant is used in multiple places, its value need only be updated in one place**
- **Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers**

Outline

skipped



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

Graphics

Applets

Drawing Shapes

Primitive Data

- **There are eight primitive data types in Java**
- **Four of them represent integers:**
  - `byte`, `short`, `int`, `long` whole numbers
    1 2 4 8 bytes
- **Two of them represent floating point numbers:**
  - `float`, `double` real numbers
- **One of them represents characters:**
  - `char`
- **And one of them represents boolean values:**
  - `boolean`

Numeric Primitive Data

java is a typed program - so there is no bit overflow: if it can only carry 3 numbers, and it is already 999, if she gets a $2 raise, then it will go over the typed amount of 3 sizess.

know how many bites each of these carry

- **The difference between the various numeric primitive types is their size, and therefore the values they can store:**

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	$+/- 3.4 \times 10^{38}$ with 7 significant digits	
`double`	64 bits	$+/- 1.7 \times 10^{308}$ with 15 significant digits	

CS -> ACM (software)
EE -> IEEE (hardware)

on guaranteed 7 decimals after the zero is valid
in a double, only 15

Characters

also primitive

single quote are for chars, double quotes are for strings

'a' is a char
"a" is a string

- **A `char` variable stores a single character**
- **Character literals are delimited by single quotes:**

```
'a'   'X'   '7'   '$'   ','   '\n'
```

this is stored as one character

- **Example declarations:**

```
char topGrade = 'A';

char terminator = ';', separator = ' ';
```

- **Note the distinction between a primitive character variable, which holds only one character, and a `String` object, which can hold multiple characters**

Character Sets

a character is one byte

- **A *character set* is an ordered list of characters, with each character corresponding to a unique number**
- **A `char` variable in Java can store any character from the *Unicode character set*** a superset of ascii
- **The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters**
- **It is an international character set, containing symbols and characters from many world languages**

in the beginning, they only had 127 characters becauase they didn't want to use negative numbers to represent the characters

now, after 127 is the extended.
Then, java went to 2 bytes

Characters

- **The *ASCII character set* is older and smaller than Unicode, but is still quite popular**
- **The ASCII characters are a subset of the Unicode character set, including:**

uppercase letters	A, B, C, …
lowercase letters	a, b, c, …
punctuation	period, semi-colon, …
digits	0, 1, 2, …
special symbols	&, |, \, …
control characters	carriage return, tab, ...

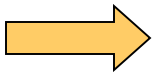
Boolean

- A `boolean` value represents a true or false condition
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

- A `boolean` variable can also be used to represent any two states, such as a light bulb being on or off

Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

Graphics

Applets

Drawing Shapes

Expressions

- **An *expression* is a combination of one or more operators and operands**
- ***Arithmetic expressions* compute numeric results and make use of the arithmetic operators:**

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	% modulus

only use remainder with whole numbers - becauase real numbers do not have remainder

- **If either or both operands used by an arithmetic operator are floating point, then the result is a floating point**

5.0 / 2 => gets promoted to 5.0/2.0 = 2.5
5/2.0 = 2.5

Division and Remainder

mod only matters when using it with real numbers

- **If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)**

```
14 / 3      equals      4
```
quotient

```
8 / 12      equals      0
```

- **The remainder operator (%) returns the remainder after dividing the second operand into the first**

```
14 % 3      equals      2

8 % 12      equals      8
```

Operator Precedence

- **Operators can be combined into complex expressions**

```
result = total + count / max - offset;
```

4 2 1 3

- **Operators have a well-defined precedence which determines the order in which they are evaluated**
- **Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation** precedence: x / + - string concat
- **Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order**

Operator Precedence

- **What is the order of evaluation in the following expressions?**

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

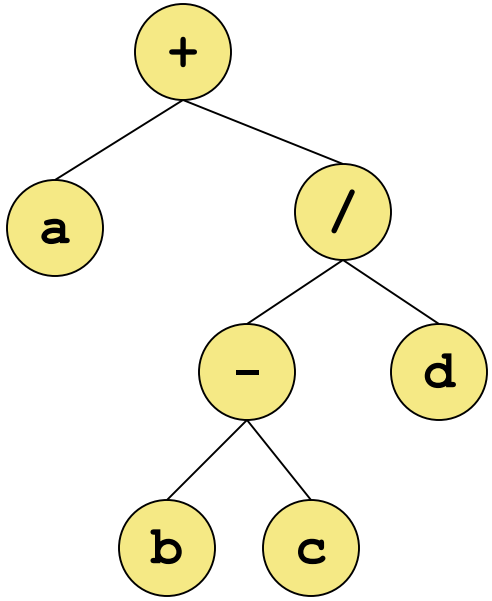
```
a / (b + c) - d % e
  2    1     4   3
```

```
a / (b * (c + (d - e)))
  4    3    2    1
```

Expression Trees

- **The evaluation of a particular expression can be shown using an *expression tree***
- **The operators lower in the tree have higher precedence for that expression**

```
a + (b - c) / d
```



wont need to know about this for until later

8/31/17 break

Assignment Revisited

- **The assignment operator has a lower precedence than the arithmetic operators**

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
```

4 1 3 2

Then the result is stored in the variable on the left hand side

Assignment Revisited

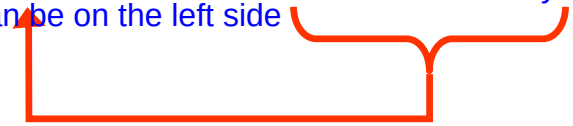
- **The right and left hand sides of an assignment statement can contain the same variable**

First, one is added to the original value of count

```
count  =  count + 1;
```

L value: only variable can be on the left side

R value - anything can be on this side



Then the result is stored back into count (overwriting the original value)

Increment and Decrement

- **The increment and decrement operators use only one operand**
- **The *increment operator* (++) adds one to its operand**
- **The *decrement operator* (--) subtracts one from its operand**
- **The statement**

```
count++;
```

is functionally equivalent to

```
count = count + 1;
```

Increment and Decrement

when only adding one, then where the + symbol is, it doesnt matter where they are. prefix or suffix

- **The increment and decrement operators can be applied in *postfix form*:**

```
count++
```

- **or *prefix form*:**

int num ++count; count - here, count gets added to 6 immediately

```
++count
```

need to see example of this

int x = 5;
int y = x++;
after everything happens, then x will get 6.

- **When used as part of a larger expression, the two forms can have different effects**

- **Because of their subtleties, the increment and decrement operators should be used with care**

Assignment Operators

- **Often we perform an operation on a variable, and then store the result back into that variable**
- **Java provides *assignment operators* to simplify that process**
- **For example, the statement**

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- **There are many assignment operators in Java, including the following:**

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Assignment Operators

- **The right hand side of an assignment operator can be a complex expression**
- **The entire right-hand expression is evaluated first, then the result is combined with the original variable**
- **Therefore**

```
result /= (total-MIN) % num;
```

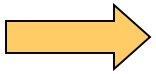
is equivalent to

```
result = result / ((total-MIN) % num);
```

Assignment Operators

- **The behavior of some assignment operators depends on the types of the operands**
- **If the operands to the += operator are strings, the assignment operator performs string concatenation**
- **The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)**

Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

Graphics

Applets

Drawing Shapes

Data Conversion

- **Sometimes it is convenient to convert data from one type to another**
- **For example, in a particular situation we may want to treat an integer as a floating point value**
- **These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation**

int can be automatically be promoted to a double, but double cannot be narrowed to an int
but you can cast

Data Conversion

- **Conversions must be handled carefully to avoid losing information**

promote

- ***Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)**

demote

- ***Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)**
- **In Java, data conversions can occur in three ways:**
  - **assignment conversion**
  - **promotion**
  - **casting**

Assignment Conversion

- ***Assignment conversion*** **occurs when a value of one type is assigned to a variable of another**
- **If `money` is a `float` variable and `dollars` is an `int` variable, the following assignment converts the value in `dollars` to a `float`**

float

```
money = dollars
```

int

10 dollars
10.0 money

- **Only widening conversions can happen via assignment** only promotion will happen automatically from the compiler

  java does not automatically demote numbers as C and C++ will do
- **Note that the value or type of `dollars` did not change**

Data Conversion

- ***Promotion*** **happens automatically when operators in expressions convert their operands**
- **For example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a floating point value to perform the following calculation:**

float / int

```
result = sum / count;
```

count only gets promoted for this expresion

Casting

note: casting a floating point to int truncates:
(int)12.25 => 12

- ***Casting*** **is the most powerful, and dangerous, technique for conversion**
- **Both widening and narrowing conversions can be accomplished by explicitly casting a value**
- **To cast, the type is put in parentheses in front of the value being converted**
- **For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:**

but watch out: evaluated first

(float)(total/ int) = 2.0 because the division was

int / int

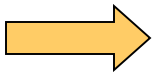
```
result = (float) total / count;
```

5 / 2 usually = 2

but the cast has higher presidence, so it is evaluated immediately to:

5.0 / 2.0 = 2.5

Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

Graphics

Applets

Drawing Shapes

jumped here from 8/31/17 break

Interactive Programs

- **Programs generally need input on which to operate**
- **The `Scanner` class provides convenient methods for reading input values of various types**
- **A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard**
- **Keyboard input is represented by the `System.in` object**

Reading Input

input in other languages:
in
get
next
scan
read

- **The following line creates a Scanner object that reads from the keyboard:**

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

"scan" now represents the keyboard

- **The `new` operator creates the `Scanner` object**
- **Once created, the `Scanner` object can be used to invoke various input methods, such as:**

answer = scan.nextLine();

in C++:
String answer = cin.getline();

reads through the carriage return but doesn't record it.

but nextInt/nextDouble stops at whitespaces, so does not consume newline(its left in input buffer)

so if u do a nextline after a nextInt, you will get the return character for that nextLine. pointer points at the \n after the nextInt, since it is not consumed

so if reading int before string, have to store the \n in a 'junk' variable

the OS opens 3 'files'

Unix	Java	
STDIN	System.in	keyboard
STDOUT	System.out	monitor/ console
STDERR	System.err	monitor/console

Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used

```
import java.util.Scanner;
```

must be inported before the class

- See Echo.java (page 88)

```
//********************************************************************
//  Echo.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the nextLine method of the Scanner class
//  to read a string from the user.
//********************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message + "\"");
   }
}
```

- The `nextLine` method reads all of the input until the end of the line is found
- The details of object creation and class libraries are discussed further in Chapter 3

8/31/17 ended here

9/5/17 start

data
homework #1 extended until thursday because she didnt talk about data conversion in class

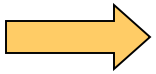
java is strongly typed language.
types don't change for the lifetime of program
those that are not strongly typed, the memory can
store anything, but those cannot be compiled - they
cannot generate an executable

Input Tokens

- **Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input**
- **White space includes space characters, tabs, new line characters**
- **The `next` method of the `Scanner` class reads the next input token and returns it as a string**
- **Methods such as `nextInt` and `nextDouble` read data of particular types**
- **See GasMileage.java (page 89)**

```
//********************************************************************
// GasMileage.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the Scanner class to read numeric data.
//********************************************************************

import java.util.Scanner;

public class GasMileage
{
  //-----------------------------------------------------------------
  //  Calculates fuel efficiency based on values entered by the
  //  user.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    int miles;
    double gallons, mpg;

    Scanner scan = new Scanner (System.in);

    System.out.print ("Enter the number of miles: ");
    miles = scan.nextInt();

    System.out.print ("Enter the gallons of fuel used: ");
    gallons = scan.nextDouble();

    mpg = miles / gallons;

    System.out.println ("Miles Per Gallon: " + mpg);
  }
}
```

Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

Graphics

Applets

Drawing Shapes

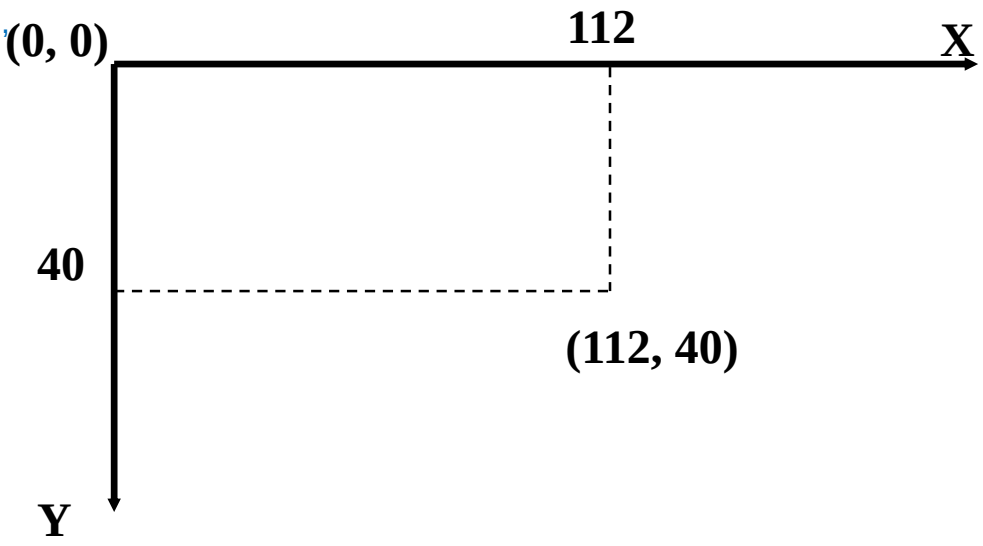
Introduction to Graphics

- **The last few sections of each chapter of the textbook focus on graphics and graphical user interfaces**
- **A picture or drawing must be digitized for storage on a computer**
- **A picture is made up of *pixels* (picture elements), and each pixel is stored separately**
- **The number of pixels used to represent a picture is called the *picture resolution***
- **The number of pixels that can be displayed by a monitor is called the *monitor resolution***

Coordinate Systems

- **Each pixel can be identified using a two-dimensional coordinate system**
- **When referring to a pixel in a Java program, we use a coordinate system with the origin in the top-left corner**

x,y are in the top left.
some systems correct this for you,
but java doesnt



Representing Color

- **A black and white picture could be stored using one bit per pixel (0 = white and 1 = black)**
- **A colored picture requires more information; there are several techniques for representing colors**
- **For example, every color can be represented as a mixture of the three additive primary colors Red, Green, and Blue**
- **Each color is represented by three numbers between 0 and 255 that collectively are called an *RGB value*** 24 bit color

The Color Class

- **A color in a Java program is represented as an object created from the `Color` class**
- **The `Color` class also contains several predefined colors, including the following:**

Object	RGB Value
`Color.black`	`0, 0, 0`
`Color.blue`	`0, 0, 255`
`Color.cyan`	`0, 255, 255`
`Color.orange`	`255, 200, 0`
`Color.white`	`255, 255, 255`
`Color.yellow`	`255, 255, 0`

the maximum for one byte is 256 numbers (but one is 0, so 0-255)
so the biggest number you can represent in one byte is 255

Outline

Character Strings

Variables and Assignment

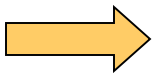
Primitive Data Types

Expressions

Data Conversion

Interactive Programs

Graphics



Applets we are not doing applets

Drawing Shapes

Applets

hot java

- **A Java application is a stand-alone program with a `main` method (like the ones we've seen so far)**
- **A Java *applet* is a program that is intended to transported over the Web and executed using a web browser**
- **An applet also can be executed using the appletviewer tool of the Java Software Development Kit**
- **An applet doesn't have a `main` method**
- **Instead, there are several special methods that serve specific purposes**

Applets

- The `paint` method, for instance, is executed automatically and is used to draw the applet's contents
- The `paint` method accepts a parameter that is an object of the `Graphics` class
- A `Graphics` object defines a *graphics context* on which we can draw shapes and text
- The `Graphics` class has several methods for drawing shapes

Applets

- **The class that defines an applet *extends* the `Applet` class**
- **This makes use of *inheritance*, which is explored in more detail in Chapter 8**
- **See Einstein.java (page 95)**
- **An applet is embedded into an HTML file using a tag that references the bytecode file of the applet**
- **The bytecode version of the program is transported across the web and executed by a Java interpreter that is part of the browser**

The HTML applet Tag

```
<html>

   <head>
      <title>The Einstein Applet</title>
   </head>

   <body>
      <applet code="Einstein.class" width=350 height=175>
      </applet>
   </body>

</html>
```

this opened up applet,enclosed and it wont touch local data

```
//********************************************************************
//  Einstein.java       Author: Lewis/Loftus
//
//  Demonstrates a basic applet.
//********************************************************************

import javax.swing.JApplet;
import java.awt.*;

public class Einstein extends JApplet
{
   //-----------------------------------------------------------------
   //  Draws a quotation by Albert Einstein among some shapes.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.drawRect (50, 50, 40, 40);    // square
      page.drawRect (60, 80, 225, 30);   // rectangle
      page.drawOval (75, 65, 20, 20);    // circle
      page.drawLine (35, 60, 100, 120);  // line

      page.drawString ("Out of clutter, find simplicity.", 110, 70);
      page.drawString ("-- Albert Einstein", 130, 100);
   }
}
```

Outline

Character Strings

Variables and Assignment

Primitive Data Types

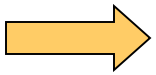
Expressions

Data Conversion

Interactive Programs

Graphics

Applets



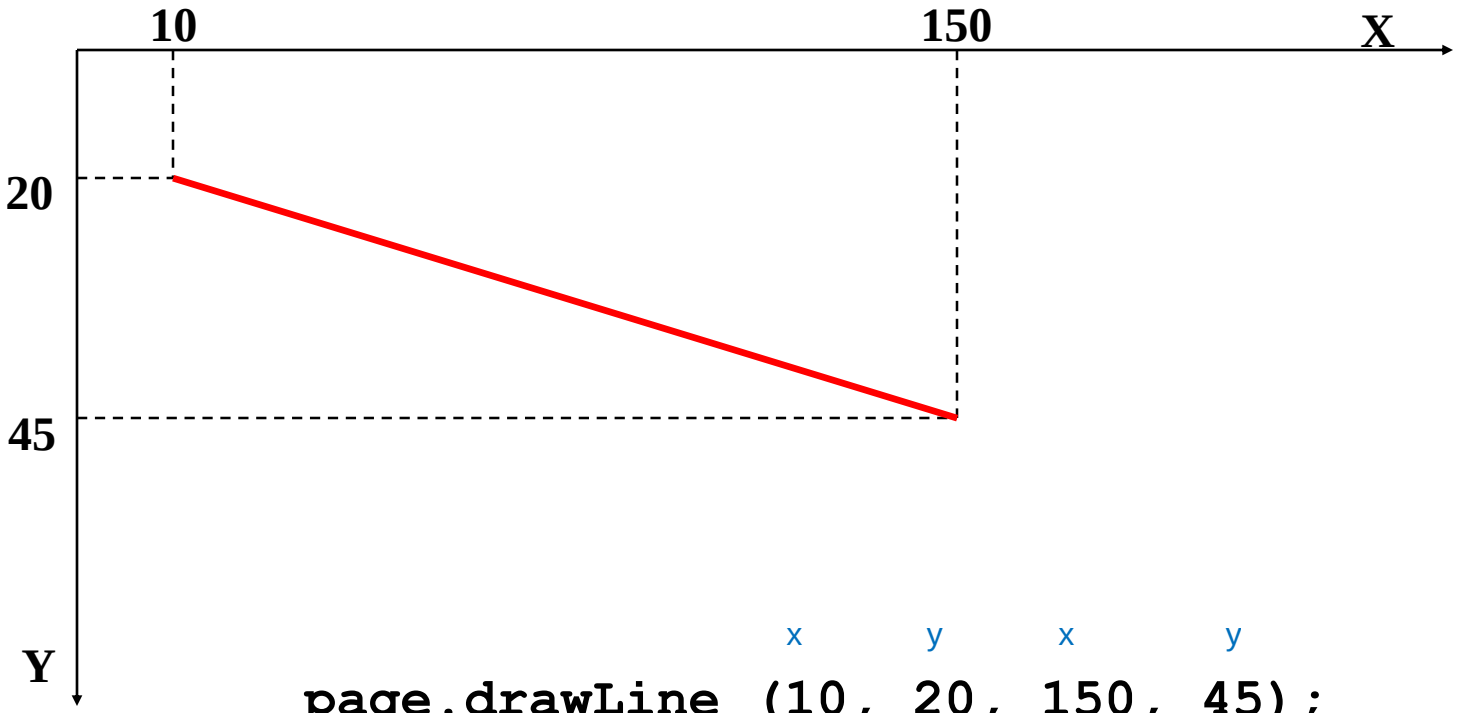
Drawing Shapes

Drawing Shapes

- **Let's explore some of the methods of the `Graphics` class that draw shapes in more detail**
- **A shape can be filled or unfilled, depending on which method is invoked**
- **The method parameters specify coordinates and sizes**
- **Shapes with curves, like an oval, are usually drawn by specifying the shape's *bounding rectangle***
- **An arc can be thought of as a section of an oval**

Drawing a Line

drawing a line

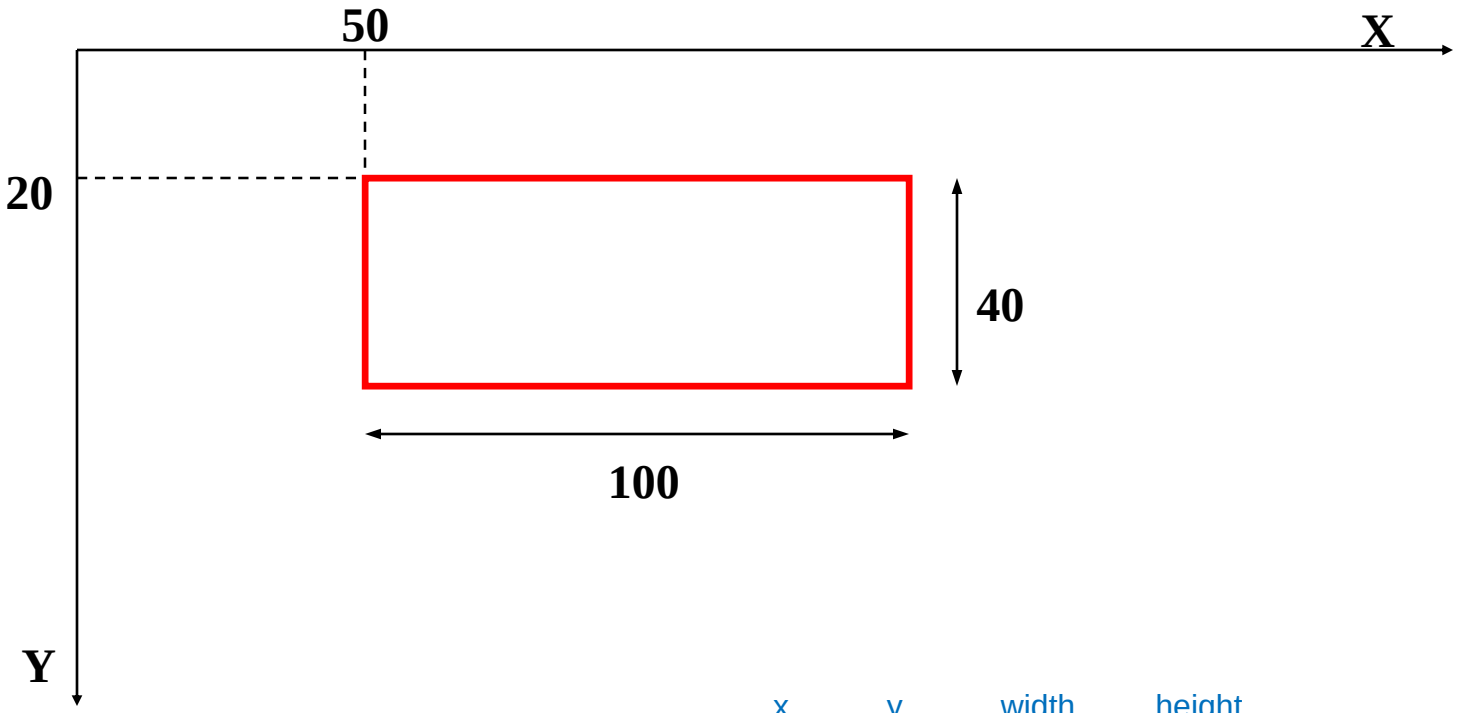



```
                    x     y     x     y
page.drawLine (10, 20, 150, 45);
```

or

```
page.drawLine (150, 45, 10, 20);
```

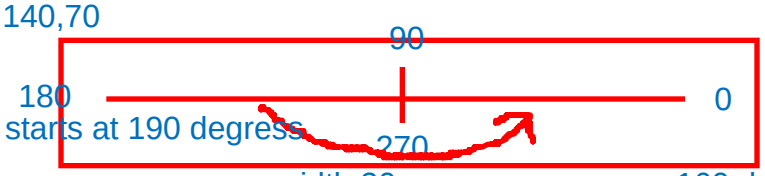
Drawing a Rectangle



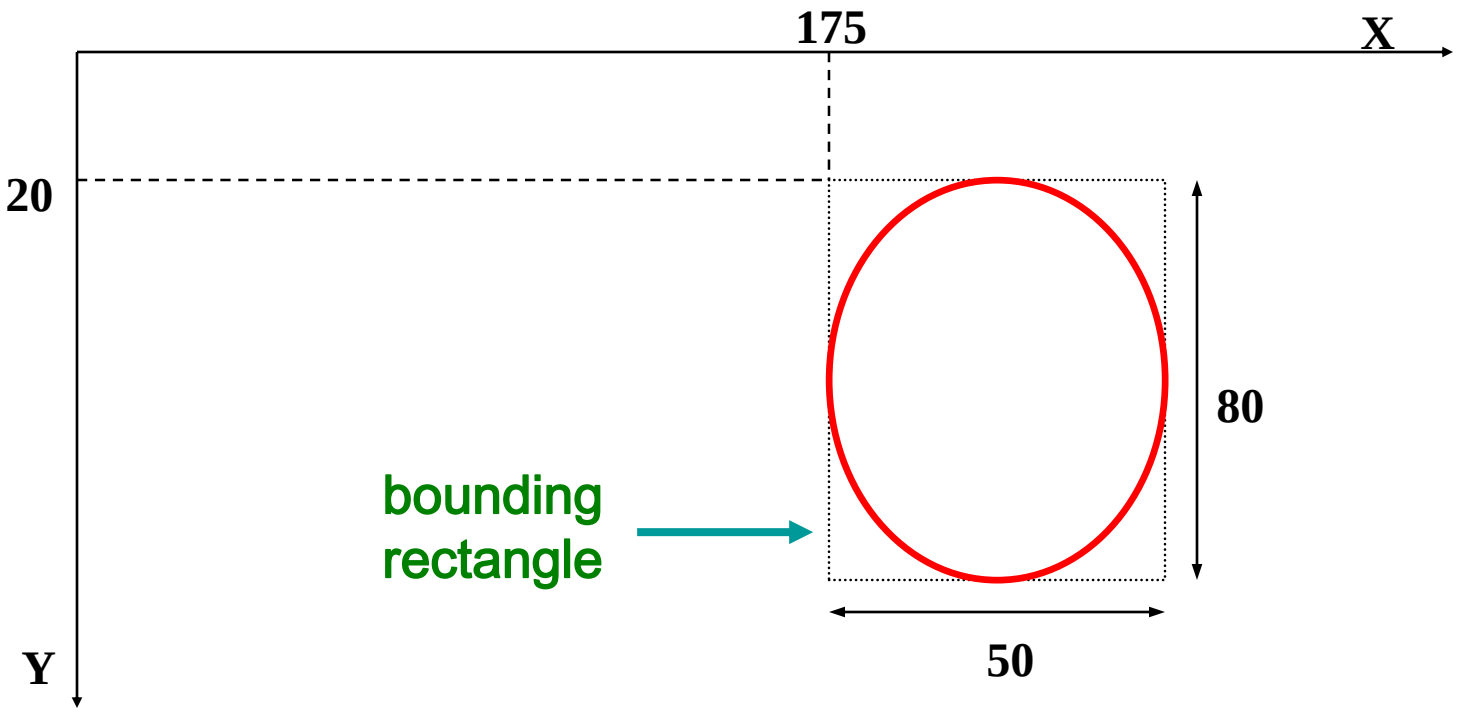
x y width height

```
page.drawRect (50, 20, 100, 40);
```

page,drawArc(140,70,20,10,190,160)



Drawing an Oval



x y width height of bounding retangle

```
page.drawOval (175, 20, 50, 80);
```

in circle, the width and height would be the same

Drawing Shapes

applets did not have a main

- **Every drawing surface has a *background color***
- **Every graphics context has a current *foreground color***
- **Both can be set explicitly**
- **See Snowman.java (page100)**

the one in the book is an applet, but we will use an application

```
//********************************************************************
//  Snowman.java       Author: Lewis/Loftus
//
//  Demonstrates basic drawing methods and the use of color.
//********************************************************************
import javax.swing.JApplet;
import java.awt.*;

public class Snowman extends JApplet
{
  //-----------------------------------------------------------------
  //  Draws a snowman.
  //-----------------------------------------------------------------
  public void paint (Graphics page)
  {
    final int MID = 150;
    final int TOP = 50;

    setBackground (Color.cyan);

    page.setColor (Color.blue);
    page.fillRect (0, 175, 300, 50);  // ground

    page.setColor (Color.yellow);
    page.fillOval (-40, -40, 80, 80);  // sun

    page.setColor (Color.white);
    page.fillOval (MID-20, TOP, 40, 40);      // head
    page.fillOval (MID-35, TOP+35, 70, 50);   // upper torso
    page.fillOval (MID-50, TOP+80, 100, 60);  // lower torso

    page.setColor (Color.black);
    page.fillOval (MID-10, TOP+10, 5, 5);   // left eye
    page.fillOval (MID+5, TOP+10, 5, 5);    // right eye

    page.drawArc (MID-10, TOP+20, 20, 10, 190, 160);   // smile

    page.drawLine (MID-25, TOP+60, MID-50, TOP+40);  // left arm
    page.drawLine (MID+25, TOP+60, MID+55, TOP+60);  // right arm

    page.drawLine (MID-20, TOP+5, MID+20, TOP+5);  // brim of hat
    page.fillRect (MID-15, TOP-20, 30, 25);        // top of hat
  }
}
```

9/7/17 end here

Summary

- **Chapter 2 focused on:**
  - **character strings**
  - **primitive data**
  - **the declaration and use of variables**
  - **expressions and operator precedence**
  - **data conversions**
  - **accepting input from the user**
  - **Java applets**
  - **introduction to graphics**