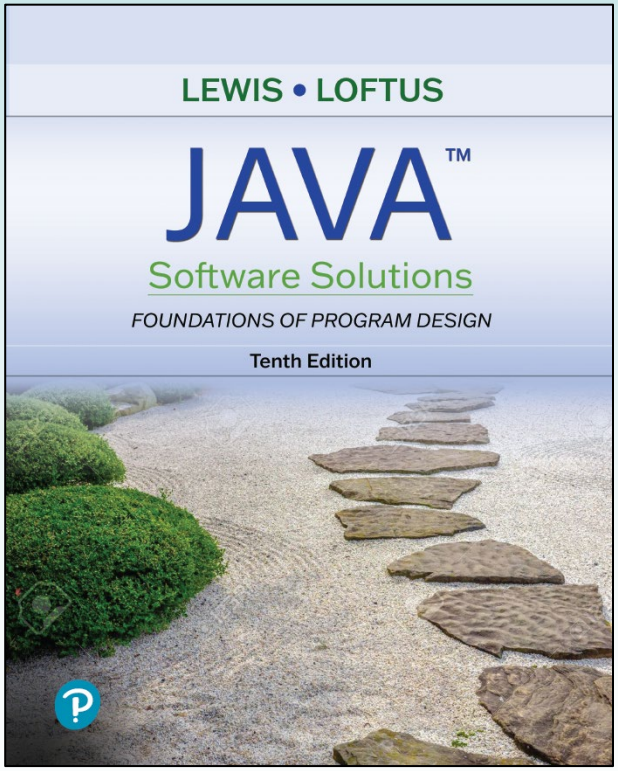
Chapter 2
Data and Expressions



Java Software Solutions

Foundations of Program Design

10th Edition

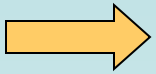
Edited:
06/11/2018
01/23/2022
09/07/2023
01/31/2024
08/07/2026

John Lewis
William Loftus

Data and Expressions

- Chapter 2 focuses on:
  - character strings
  - primitive data
  - declaring and using variables
  - expressions and operator precedence
  - data conversions
  - accepting input from the user

Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

☺ **Graphics**

§2.1 Character Strings

- A *string literal* is represented by putting double quotes around text
- Examples:

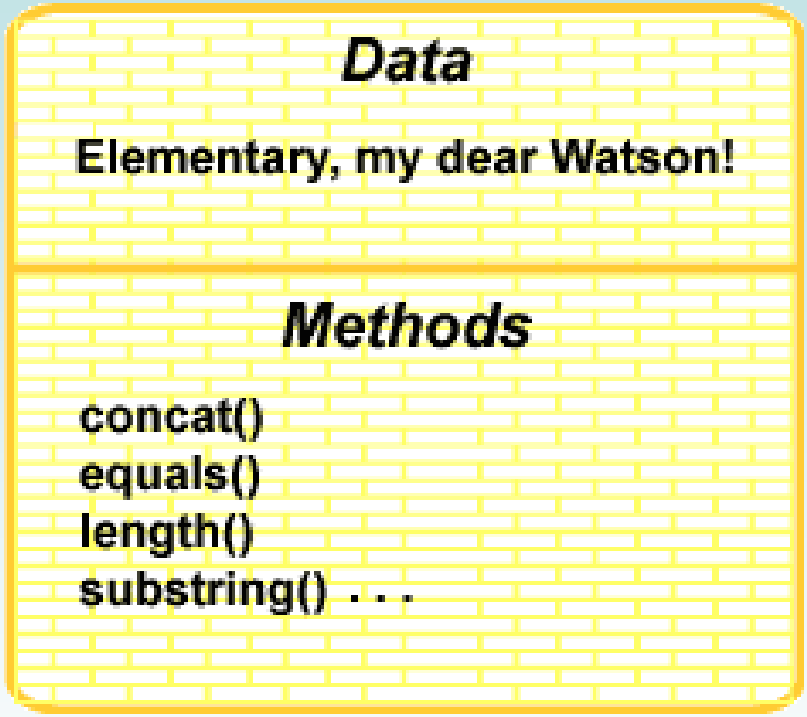
```
"This is a string literal."
"123 Main Street"
"X"
```

- Every string literal in your source code corresponds to a software object at run-time
- Every string literal is a **`String`** object
  - The name of the class is String
  - All string literals are instances of that class, but each on is a separate objects

Class `String`

- `String` is a class
- The class describes what data objects contain (state), and what they can do (behavior)
  - A class is a concept (described using Java)
  - An object is an instance of the concept (created when a program runs)
- For `String` the data is a sequence of characters
- The behavior is a collection of *methods*

An object is a block of memory (Yellow bricks in this illustration) that contain data and methods. All String objects have the same methods. You don't have to ask for them individually.

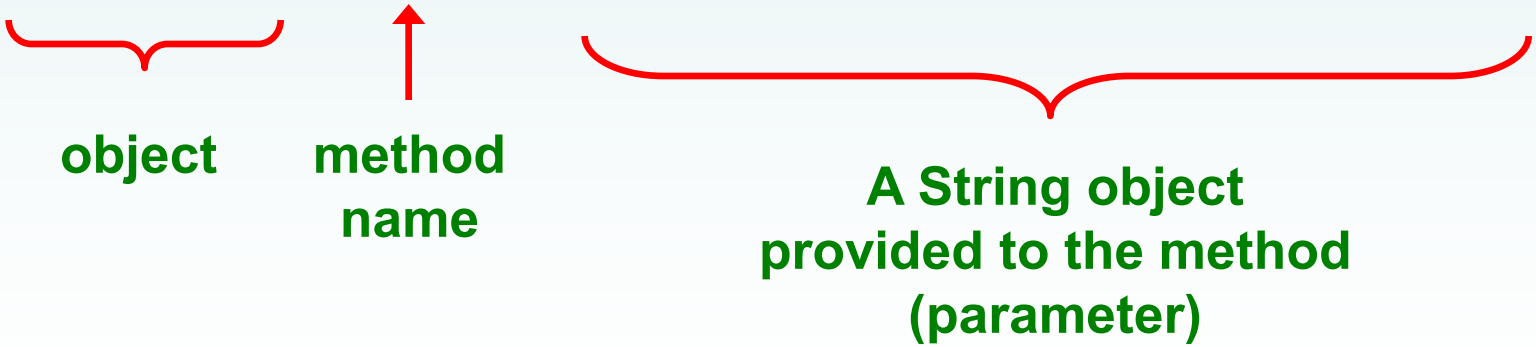


"Elementary, my dear Watson!"

The println Method

- The `Lincoln` program from Chapter 1, used the **`println`** method to print the characters contained in a **`String`**
- The **`System.out`** object represents the monitor screen

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



The print Method

- The `System.out` object has several methods that do output
- The `print` method is similar to the `println` method, except that it does not advance to the next line
  - Anything printed after a `print` statement will appear on the same line
- See `Countdown.java`

```java
//********************************************************************
//  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... "); // appears on first output line
      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.");
   }
}
```

Output

```
Three... Two... One... Zero... Liftoff!
Houston, we have a problem.
```

```
//****                                                              ****
//  Co
//
//  De
//****                                                              ****

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* (+) makes a new, bigger string by appending one string to the end of another

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

- It always creates a new string using data in the original strings
  - After the above has executed, there will be three strings: “Peanut butter ”, “and jelly”, “Peanut butter and jelly”
- It can also append a number to a string
- A string literal cannot be broken across two lines in a program

```
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.
      // The value is first used to create characters.
      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");
   }
}
```

Output

```
We present the following facts for your extracurricular edification:

Letters in the Hawaiian alphabet: 12
Dialing code for Antarctica: 672
Year in which Leonardo da Vinci invented the parachute: 1515
Speed of ketchup: 40 km per year
```

Overloading

- The + operator is also used for arithmetic addition
- What it does 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 a the order you want
- See `Addition.java`

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

Output

```
24 and 45 concatenated: 2445
24 and 45 added: 69
```

```
//**********                                                    **************
//  Addition.
//
//  Demonstra                                                   string
//  concatena
//*******************************************************************

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));
   }
}
```

Quick Check

What output is produced by the following?

```
System.out.println("X: " + 25);
System.out.println("Y: " + (15 + 50));
System.out.println("Z: " + 300 + 50);
```

Quick Check

What output is produced by the following?

```
System.out.println("X: " + 25);
System.out.println("Y: " + (15 + 50));
System.out.println("Z: " + 300 + 50);
```

```
X: 25
Y: 65
Z: 30050
```

Escape Sequences

- What if we wanted to print 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.");
```

Escape Sequences

- Some Java escape sequences:

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

- See `Roses.java`

```
//********************************************************************
//  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.");
   }
}
```

Output

```
Roses are red,
	Violets are blue,
Sugar is sweet,
	But I have "commitment issues",
	So I'd rather just be friends
	At this point in our relationship.
```

```
//****
//  Ro
//
//  De
//****

public
{
   //-
   //
   //-----------------------------------------------------------
   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.");
   }
}
```

Quick Check

Write a single `println` statement that produces the following output:

"Thank you all for coming to my home
tonight," he said mysteriously.

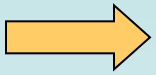
Quick Check

Write a single `println` statement that produces the following output:

"Thank you all for coming to my home
tonight," he said mysteriously.

```
System.out.println("\"Thank you all for " +
   "coming to my home\ntonight,\" he said " +
   "mysteriously.");
```

Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

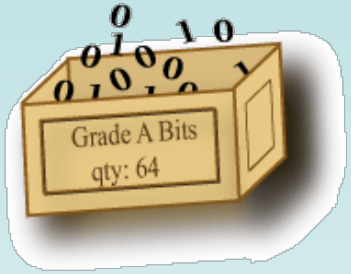
Data Conversion

Interactive Programs

☺ **Graphics**

§2.2 Variables

- A *variable* is a name for a location in memory that holds a value.
  - Think of it as a box that contains a value
  - The value in the box can change as a program runs



- A *variable declaration* specifies the variable's name and the type of information that it will hold

data type **variable name** **variable name**

```
int total;
int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initialization

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

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

- When a variable is used in a program, its current value is used
- A variable can only hold the type of information given in the declaration
- See `PianoKeys.java`

```java
//********************************************************************
//  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.");
   }
}
```

Output

```
A piano has 88 keys.
```

```
//********************                                    ********************
//  PianoKeys.java
//
//  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.");
   }
}
```

```
//********************************************************************
//  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 = 88.7;
      System.out.println("A piano has " + keys + " keys.");
   }
}
```

Assignment

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

```
total = 55;
```

- The value that was in `total` is replaced

Two Steps in Assignment

```
total = 55;
```

- Two steps:
  - i) evaluate expression on the right
  - ii) copy result into the variable
- The value that was in `total` is replaced
- You can only assign a value to a variable that is consistent with the variable's declared type

```java
//********************************************************************
//  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.");
   }
}
```

Output

```
A heptagon has 7 sides.
A decagon has 10 sides.
a dodecagon has 12 sides.
```

```
//**************                                          ***************
//  Geometry.ja
//
//  Demonstrate                                          change the
//  value store
//**************                                          ***************

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.");
   }
}
```

```
//********************************************************************
//  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.");

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

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

Constants

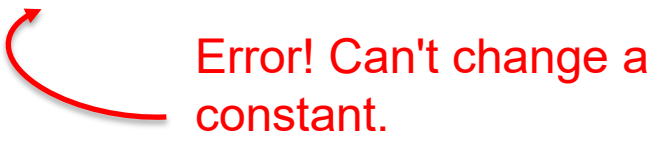
- A *constant* is similar to a variable except that its initial value does not change
- The compiler will not let you change the value of a constant
- Use the **`final`** modifier to declare a constant

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

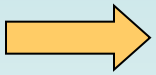
Constants

- Constants are useful for three important reasons:
  - First, they give meaning to otherwise unclear literal values
    - Example: `MAX_LOAD` means more than the literal 250
  - Second, they help program maintenance
    - If a constant is used in multiple places, its value need only be set in one place
    - Changes easily and consistently made
  - Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

```
//********************************************************************
//  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)
   {
      final 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.");
   }
}
```



Outline



Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

☺ **Graphics**

§2.3 Primitive Data

- "Primitive" means "fundamental"
- Data in Java is either primitive data or object data
- You can design any number of object types (called "classes")
  - You can have as many objects of each type as you need.
- The primitive types are built-in and you cannot add new ones
  - You can have as many primitive variables as you need. You just can't design a new primitive type.

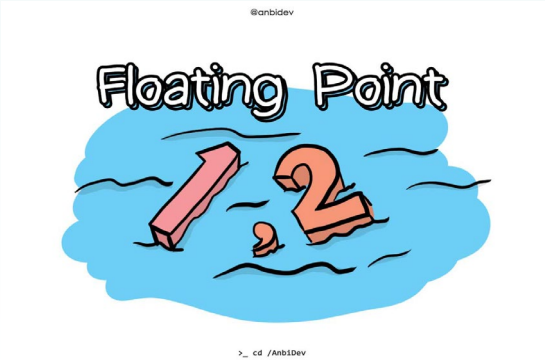
Primitive Data

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

Numeric Primitive Data

- The difference between the numeric primitive types is their size and 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	-9×10^{18}	9×10^{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



Characters

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

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

- Example declarations:

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

- Notes:
  - a primitive character variable holds only one character
  - a `String` object can hold many characters
  - a `String` literal uses double quotes: `"A Nice String "`
  - a `String` literal can hold a single character: `"X"`

Character Sets

- A *character set* is an ordered list of characters
- Each character corresponds to a unique binary pattern
- A **`char`** variable in Java can store any character from the *Unicode character set*
- The Unicode character set uses 16 bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages

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;
```

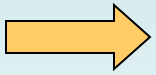
- A `boolean` variable can also be used to represent any two states, such as a light bulb being on or off

Outline

Character Strings

Variables and Assignment

Primitive Data Types



Expressions

Data Conversion

Interactive Programs

☺ **Graphics**

§2.4 Expressions

- An *expression* is a combination of one or more operators and operands that computes a value
- An *arithmetic expression* computes numeric results and makes use of the arithmetic operators:

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Remainder	**%**

- If either or both operands are floating point values, then the result is a floating point value

Division and Remainder

- If both operands for the division operator (/) are integers, the result is an integer

```
14 / 3   equals   4
8 / 12   equals   0
```

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

```
14 % 3   equals   2
8 % 12   equals   8
```

Quick Check

What are the results of the following expressions?

```
   12 / 2
12.0 / 2.0
   10 / 3
10 / 3.0
   4 / 10
4.0 / 10
   12 % 3
   10 % 3
   3 % 10
```

Quick Check

What are the results of the following expressions?

```
   12 / 2   =  6
12.0 / 2.0  =  6.0
   10 / 3   =  3
  10 / 3.0  =  3.33333333
   4 / 10   =  0
  4.0 / 10  =  0.4
   12 % 3   =  0
   10 % 3   =  1
   3 % 10   =  3
```

integer

floating point

Operator Precedence

- Operators can be combined into larger expressions

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

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated before addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right
- Parentheses change the evaluation order
  - Sub-expressions inside () are done first.

Order Makes a Difference

6 / 3 + 4 = 2 + 4 = 6

6 / (3 + 4) = 6 / 7 = 0

1 / 2 + 1 / 2 = 0 + 0 = 0

i. The two / are high precedence, so are done first, left to right

ii. The first division does integer division because both operands are int

iii. Then the second division uses integer division and computes integer 0

iv. Then the addition uses integer addition because both operands are integers

1 / 2 + 1.0 / 2 = 0 + 0.5 = 0.5

i. The two / are high precedence, so are done first, left to right

ii. The first division does integer division.

iii. Then the second division uses floating point division.

iv. Then the addition uses floating point addition because one operand is floating point.

Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
```

```
a + b * c - d / e
```

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

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

Quick Check

In what order are the operators evaluated in the following expressions?

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

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

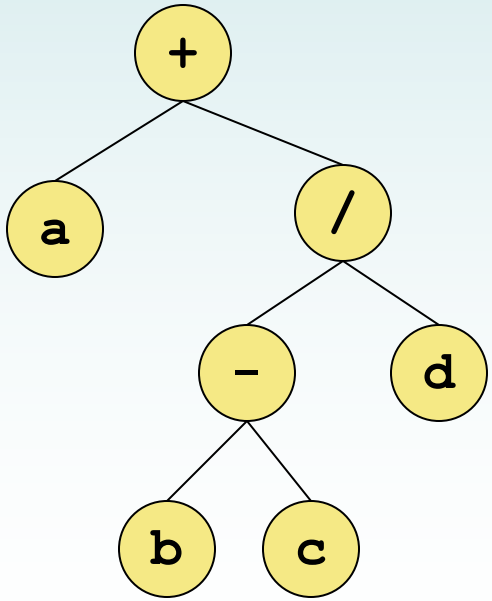
```
a / (b + c) - d % e
  2    1    4   3
```

```
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

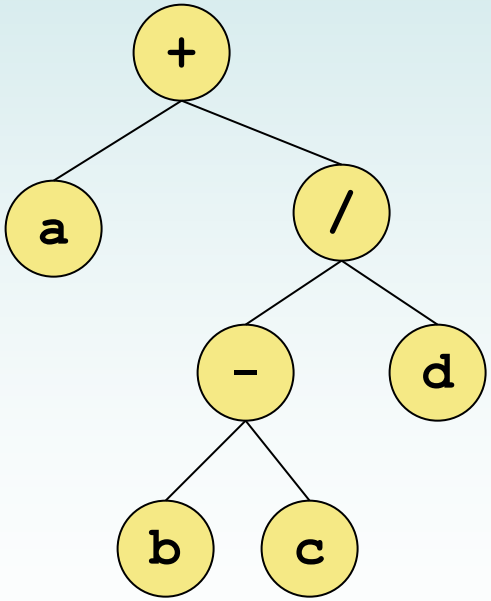
```
a + (b - c) / d
```

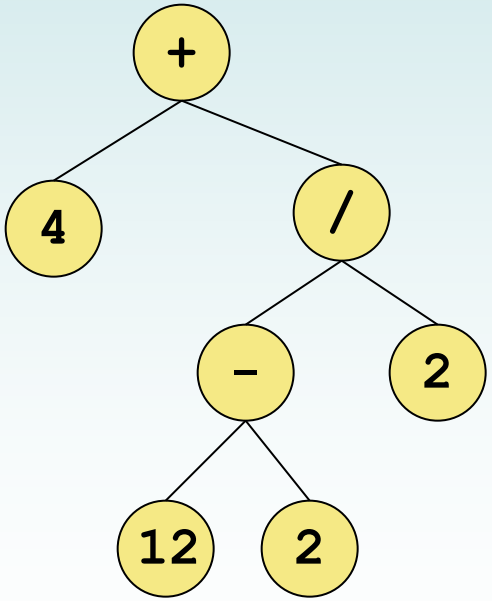


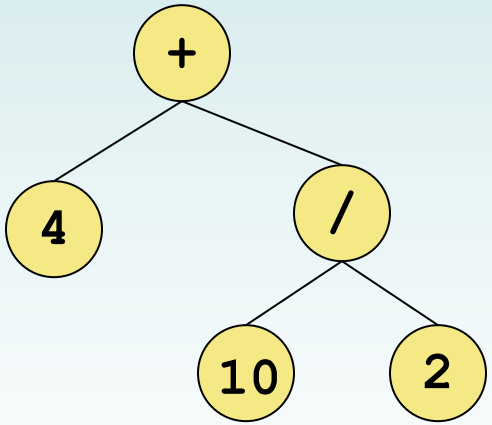
Values propagate upward

int a=4, b=12, c=2, d=2 ;

a + (b – c) / d



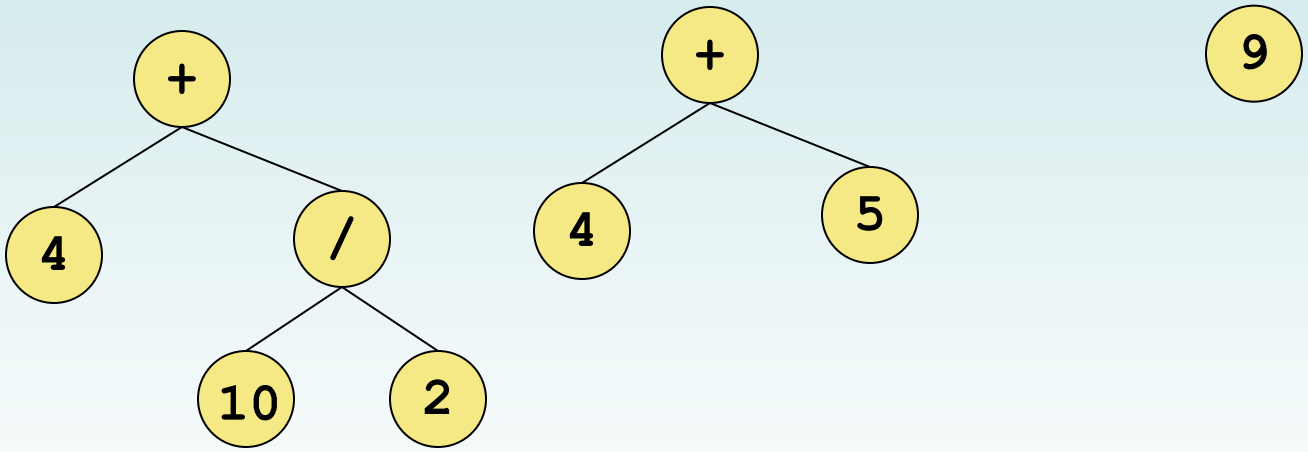




Values propagate upward

int a=4, b=12, c=2, d=2 ;

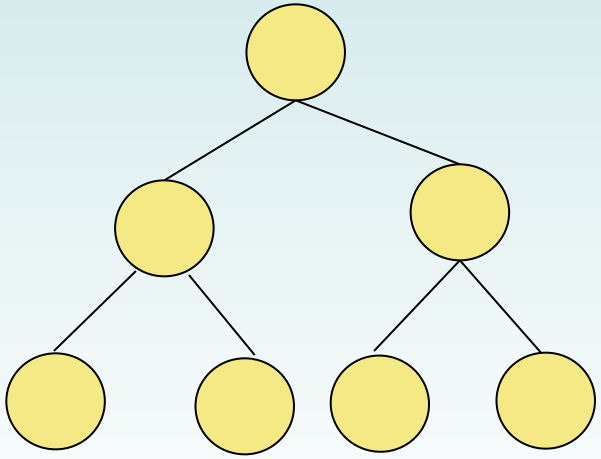
a + (b – c) / d



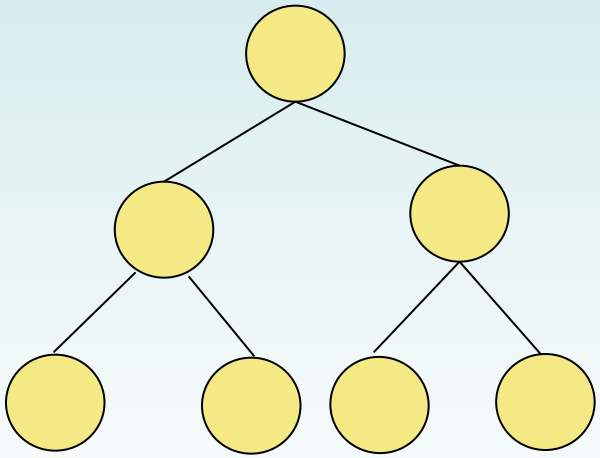
Puzzle: fill in the nodes of the tree

int a=4, b=12, c=2, d=2 ;

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



Variables and operators

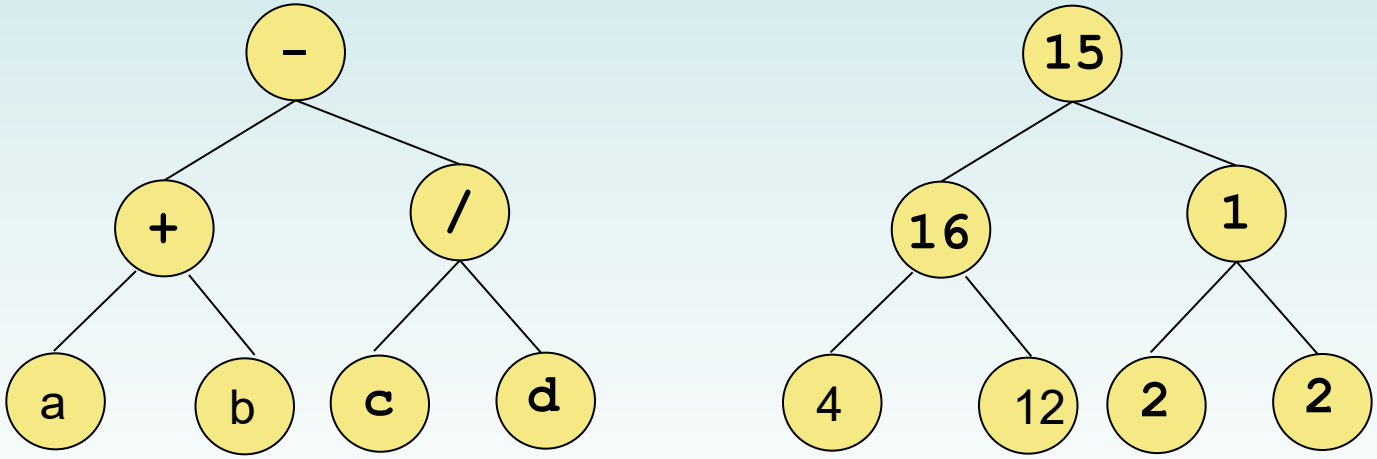


values

Puzzle

int a=4, b=12, c=2, d=2 ;

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



Assignment Revisited

- The assignment operator has lower precedence than the arithmetic operators
- Recall slide 32.

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;
```

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 statement

```
count++;
```

  is equivalent to

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

Increment and Decrement

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

```
count++
```

- or *prefix form*:

```
++count
```

- 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**

§2.5 Data Conversion

- Sometimes it is convenient to convert data from one type to another
- For example, 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 it is used in a computation

```
double x;
int j = 5;
x  = j/2.0  // value in j used to create 5.0 which is then
            // divided. But contents of j does not change
```

Data Conversion

- *Widening conversions* are safest because they mostly go from a small data type to a larger one (such as a `short` (16 bits) to an `int` (32 bits))
- *Narrowing conversions* can lose information because they mostly go from a larger 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

Data Conversion

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Assignment Conversion

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- Example:

```
int dollars = 20;
double money = dollars;
```

- Only widening conversions can happen via assignment
- The value and type of `dollars` did not change
  - The data in `money` will be the double-precision floating point version of 20

Promotion

- *Promotion* happens automatically when operators in expressions convert their operands
- Operations are always done with two operands of the same type
  - double added to double
  - int divided by int
  - string concatenated with string
- Operands are converted to higher type to make both operands the same type

Promotion

- Example:

```
int count = 12;
double sum = 490.27;
result = sum / count;
```

- The value copied from `count` is converted to a floating point double value to perform the division calculation
  - The data in `count` itself does not change

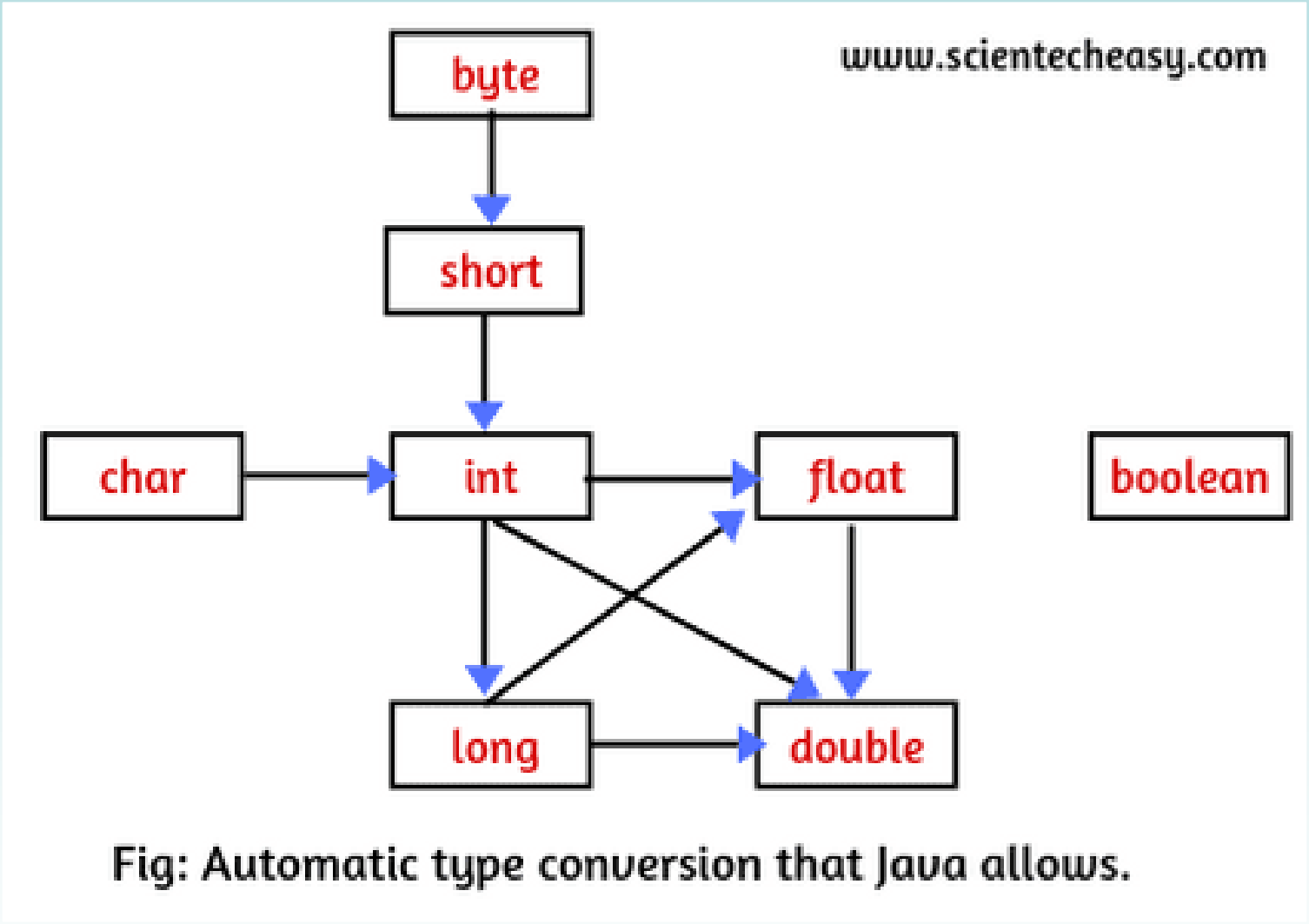


Fig: Automatic type conversion that Java allows.

You can follow several arrow from one type to another.
Eg. byte can be converted to int.

```
class TypeConversion
{
    public static void main ( String[] args )
    {
        byte b = 5;
        double x = b;
        System.out.println( "byte to double: " + x );

        char c = 'A';
        x = c;
        System.out.println( "char to double: " + x );
    }
}
```

byte to double: 5.0
char to double: 65.0

The bits that represent character 'A' are Interpreted as a number

Casting

- *Casting* is the most powerful, and most 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

```
int total = 50;
double result = (double)total / 6;
```

- Without the cast, the / would be integer division, then the `int` result would be converted to a `double` for the assignment

Outline

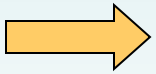
Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion



Interactive Programs

- ☺ **Graphics**

§2.6 Interactive Programs

- Programs need input
- Use the **`Scanner`** class for this
- A `Scanner` object reads input from various sources, including the keyboard
- Keyboard input is represented by the `System.in` object

Reading Input

- To create a `Scanner` object that reads from the keyboard:

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

- The `new` operator creates the `Scanner` object when the program runs
- The `Scanner` object has various input methods, such as:

```
answer = scan.nextLine();
```

Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program
- The **`nextLine`** method reads all of the input up to the end of the line
- Each time it is called it constructs a new `String` object that holds the characters in a line

```
//********************************************************************
//  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 + "\"");
   }
}
```

Sample Run

```
Enter a line of text:
You want fries with that?
You entered: "You want fries with that?"
```

```
//********************************************************************
//  Ec                                                                ***
//
//  De                                                                s
//  to
//********************************************************************

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 + "\"");
   }
}
```

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`

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

public class GasMileage
{
   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);
   }
}
```

```
//********************************************************************
//  GasMileage
//
//  Demonstrates
//********************************************************************
import java.u

public class
{
   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);
   }
}
```

Sample Run

```
Enter the number of miles: 328
Enter the gallons of fuel used: 11.2
Miles Per Gallon: 29.28571428571429
```

int miles is converted to double when used in the expression

Outline

Character Strings

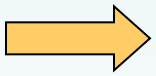
Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs



☺ Graphics

☺ Introduction to Graphics

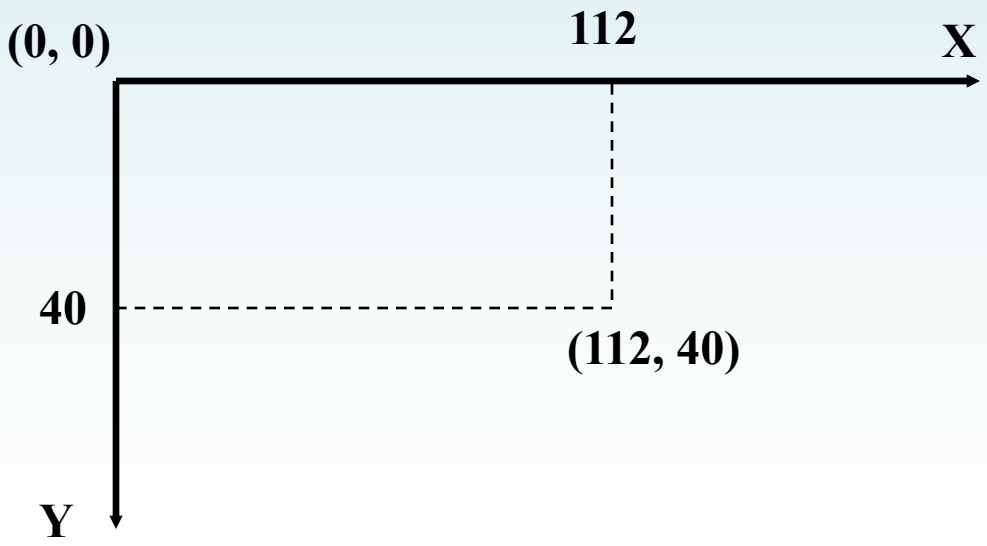
- The last few sections of each chapter of the textbook focus on graphics and graphical user interfaces
- A picture or drawing must be digitized for storage on a computer
- A picture is made up of *pixels* (picture elements), and each pixel is stored separately
- The number of pixels used to represent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

☺ Representing Images

- A digitized picture with a small portion magnified:

☺ 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



☺ 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
- 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*

☺ 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`

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
  - introduction to graphics