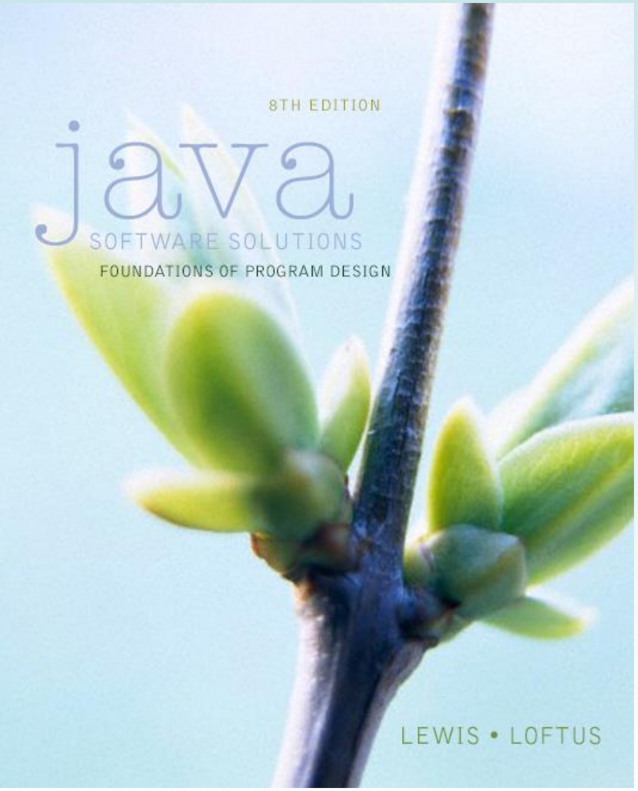
Chapter 2
Data and Expressions



Java Software Solutions

Foundations of Program Design

8th Edition

edits: 12/30/2016

John Lewis
William Loftus

Addison-Wesley
is an imprint of
PEARSON

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

☺ Applets

☺ Drawing Shapes

Character Strings

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

- Examples:

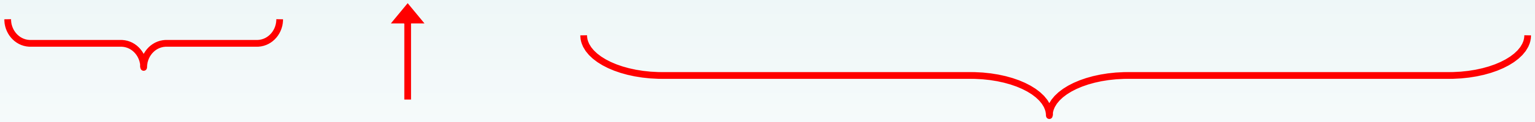
```
"This is a string literal."
"123 Main Street"
"X"
```

- Every string literal is an object in Java, defined by the `String` class
- Every string literal is 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.");
```



object

method name

information provided to the method (parameter)

The print Method

- The `System.out` object provides other services

- 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`

```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... ");
      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* (+) is used to make a new, bigger string by appending one string to the end of another

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

- It can also be used to append a number to a string
  - The number is converted to characters which are appended
- A string literal cannot be broken across two lines in a program
- See `Facts.java`

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

continue

continue

```
      // 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");
   }
}
```

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
```

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

String Concatenation

- 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 the order
- 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));
   }
}
```

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

Output

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

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`

```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

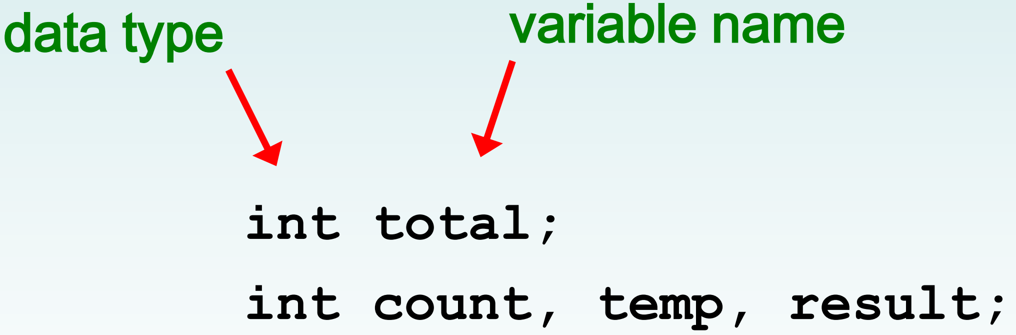
Applets

Drawing Shapes

Variables

- A *variable* is a name for a location in memory that holds a value.

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



data type

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 an expression, 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.");
   }
}
```

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

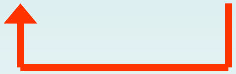
Output

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

Assignment

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

```
total = 55;
```



- 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`

```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.");
   }
}
```

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

Output

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

Outline

Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

☺ **Graphics**

☺ **Applets**

☺ **Drawing Shapes**

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”)
- The primitive types come built-in and you cannot add new ones
  - of course, you can have as many primitive variables as you need. You just can’t add 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	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	$+/- 3.4 \times 10^{38}$	with 7 significant digits
`double`	64 bits	$+/- 1.7 \times 10^{308}$	with 15 significant digits

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 = ' ';
```

- Note:
  - a primitive character variable, holds only one character
  - a `String` object, can hold many characters

Character Sets

- A *character set* is an ordered list of characters, with each character corresponding 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

☺ 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	%

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

Division and Remainder

- If both operands to 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 / 4
    10 / 4.0
      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 / 4  =  2
  10 / 4.0  =  2.5
    4 / 10  =  0
  4.0 / 10  =  0.4
    12 % 3  =  0
    10 % 3  =  1
    3 % 10  =  3
```

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, but parentheses can change the evaluation order

Order Makes a Difference

```
6 / 3 + 4   ==  2 + 4   ==   6

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


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

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

The first sub-expression uses integer division.

== means equality

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
```

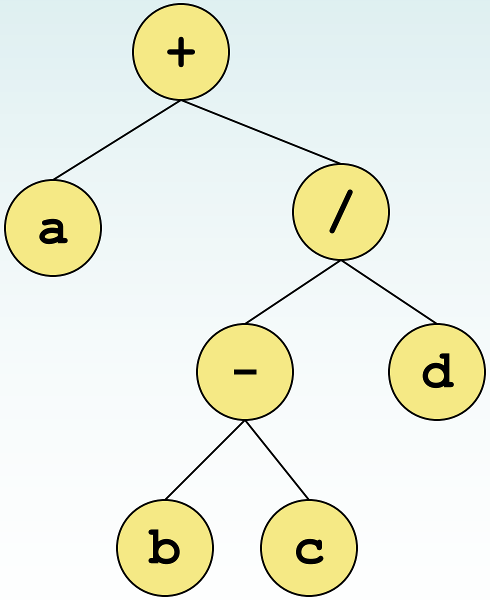
```
a / (b + c) - d % e
  2    1    4   3
```

```
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

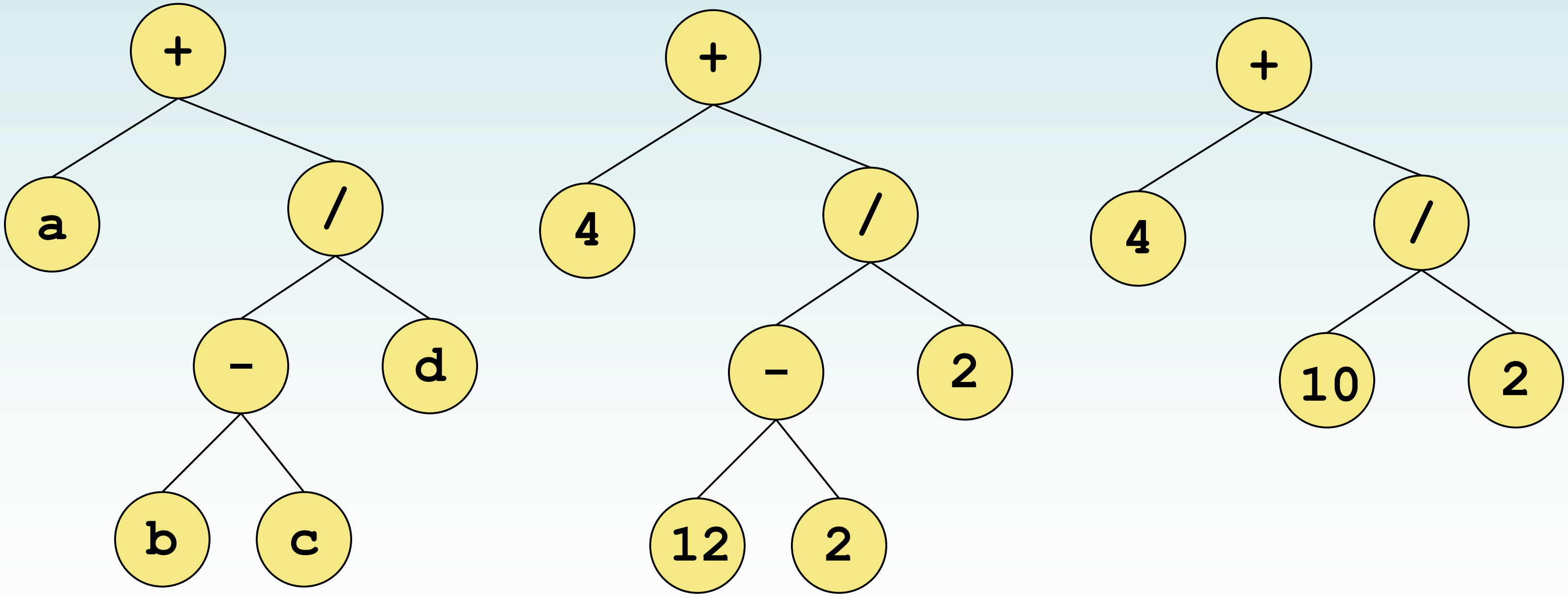
```
a + (b - c) / d
```



Values propagate upward

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

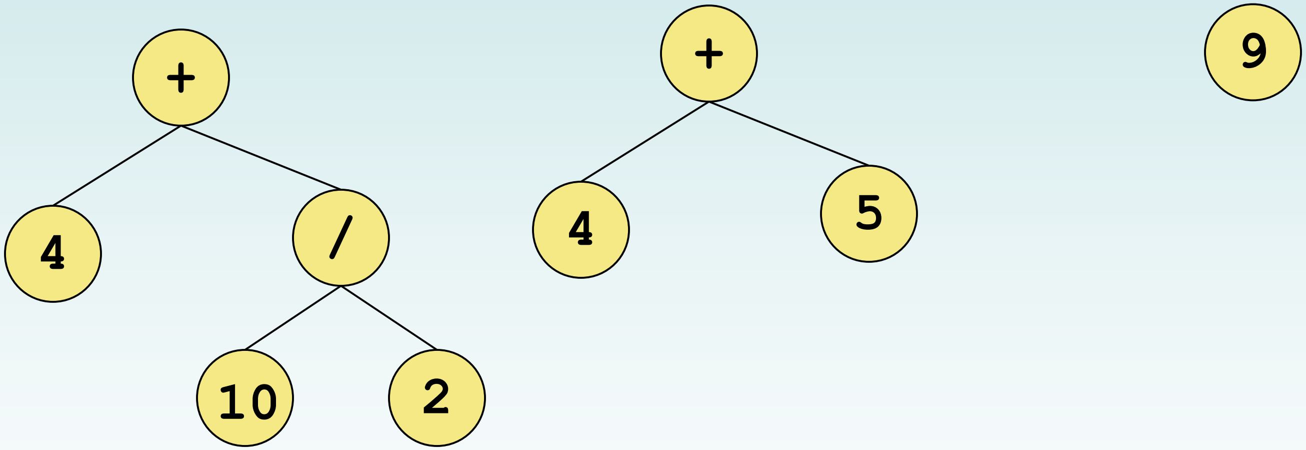
```
a + (b - c) / d
```



Values propagate upward

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

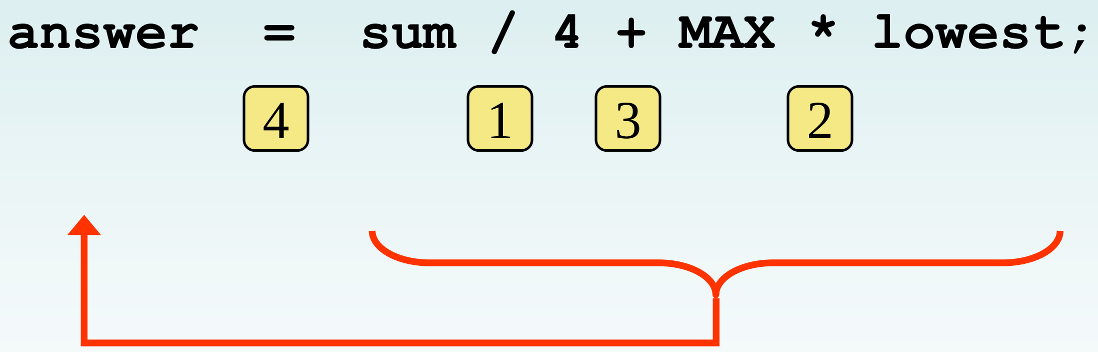
```
a + (b - c) / d
```



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;
```

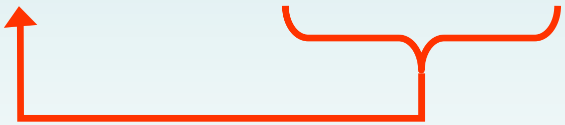
Then the result is stored in the variable on the left hand side

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 functionally 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

☺ Applets

☺ Drawing Shapes

Data Conversion

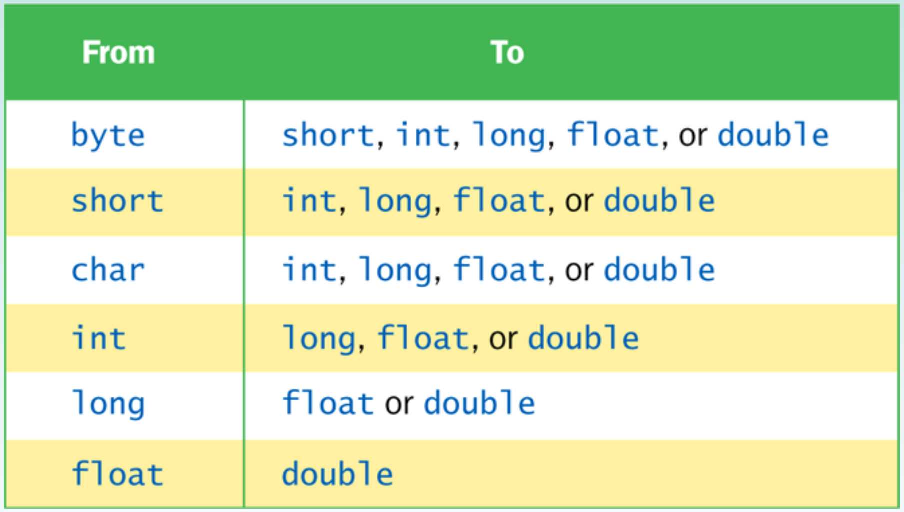
- Sometimes it is convenient to convert data from one type to another
- For example, in a particular situation we may want to treat an integer as a floating point value
- These conversions do not change the type of a variable or the value that's stored in it
  - they only convert a value as it is used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

Data Conversion

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

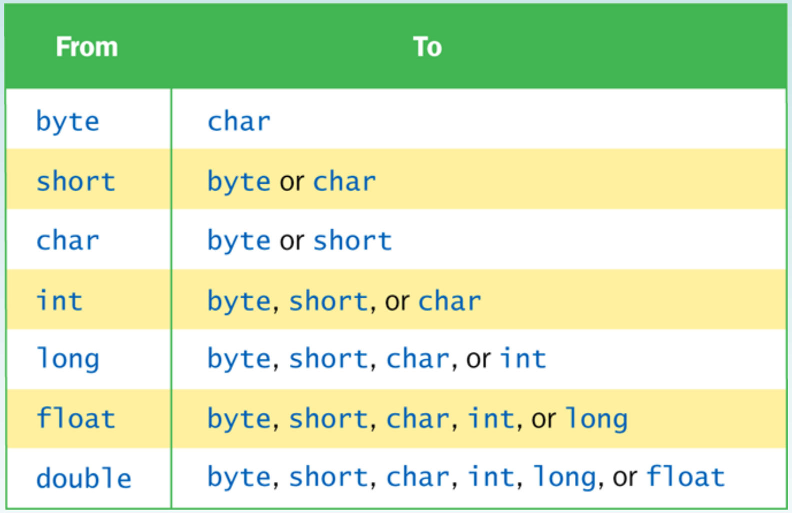
Data Conversion

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
  - What changes is the data in `money`.

Promotion

- *Promotion* happens automatically when operators in expressions convert their operands

- Example:

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

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

Casting

- *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

```
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**

☺ **Applets**

☺ **Drawing Shapes**

Interactive Programs

- Programs need input
- The `Scanner` class provides convenient methods for reading input values of various types
- 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
- 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 to be used
- 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
- See `Echo.java`
- The details of object creation and class libraries are discussed further in Chapter 3

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

Sample Run

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

Outline

Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

➡ ☺ Graphics

☺ Applets

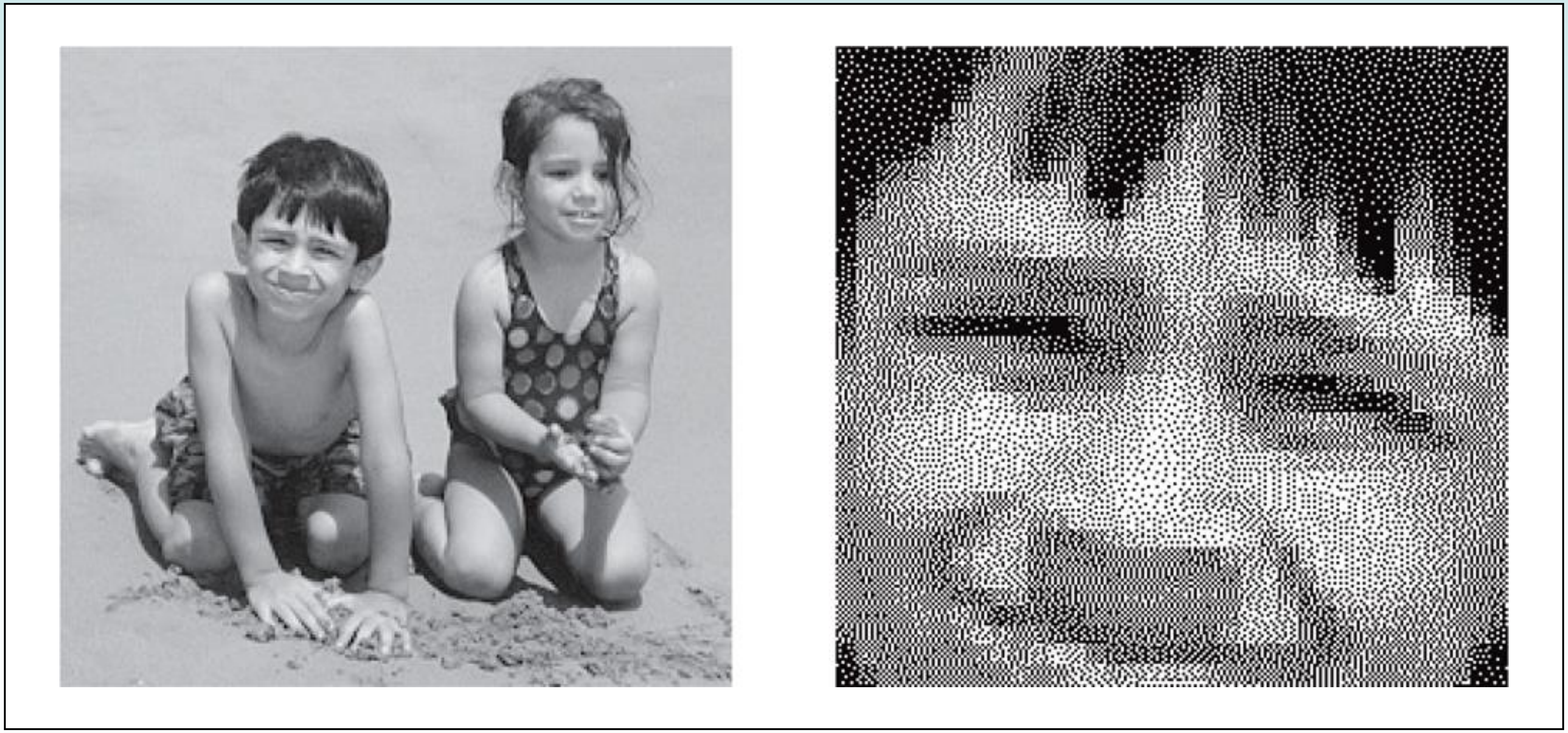
☺ Drawing Shapes

☺ 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*

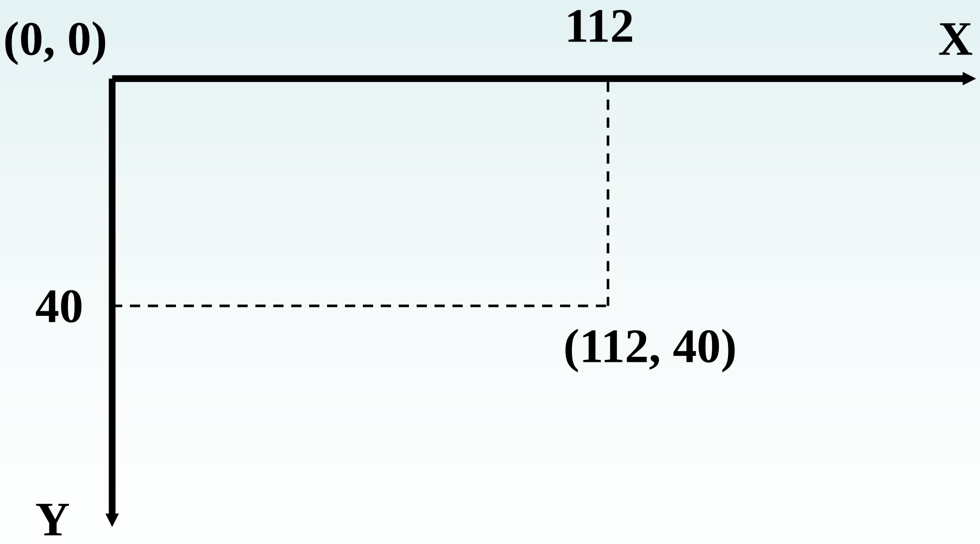
☺ Representing Images

- A digitized picture with a small portion magnified:
  - actually, the magnified picture should show each pixel with one gray level, not dots, as here



☺ Coordinate Systems

- Each pixel can be identified using a two-dimensional coordinate system
- The origin is the top-left corner
- The indexes are always whole numbers



☺ Representing Color

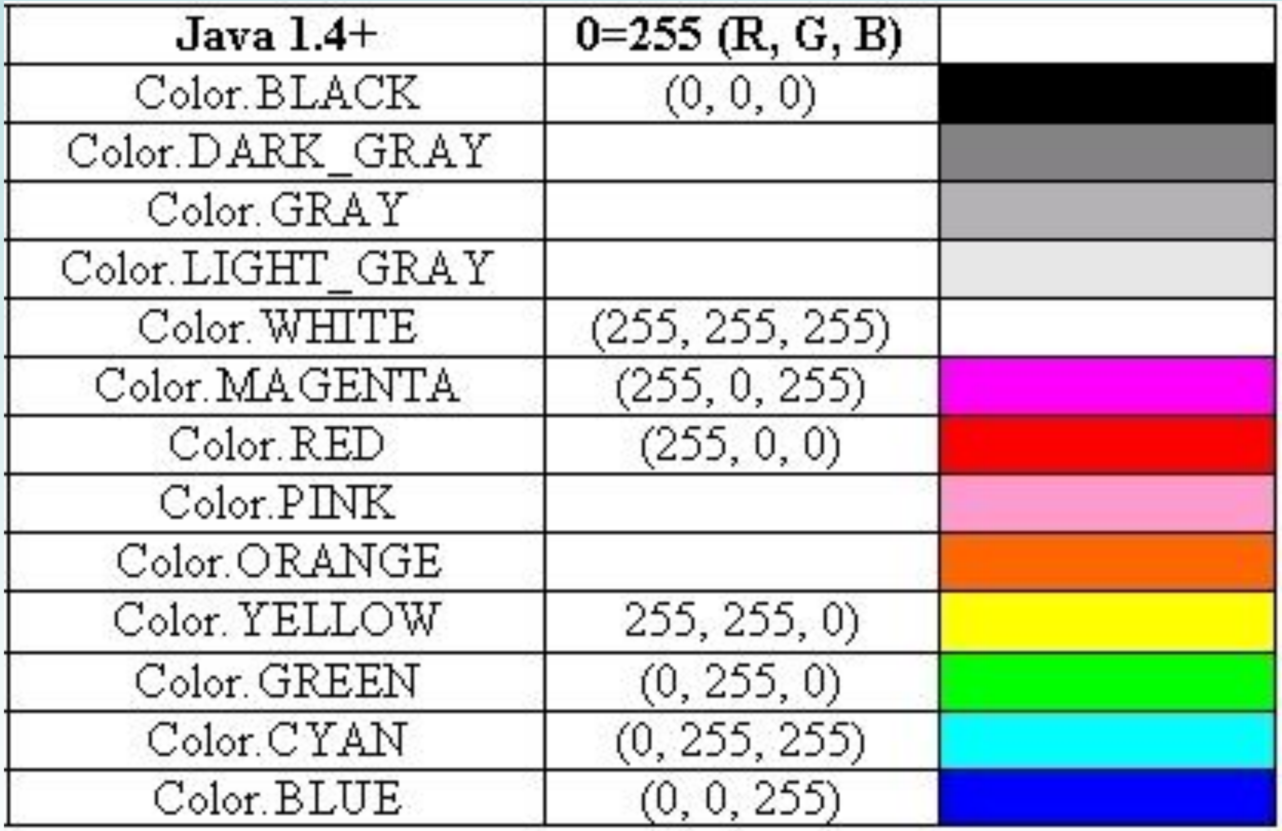
- A black and white picture could be stored using one bit per pixel (0 = white and 1 = black)
- A colored picture requires more information; there are several techniques for representing colors
- Every color can be represented as a mixture of the three additive primary colors Red, Green, and Blue
- Each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



Java 1.4+	**0=255 (R, G, B)**	
Color.BLACK	(0, 0, 0)	⬛
Color.DARK_GRAY		▓
Color.GRAY		▒
Color.LIGHT_GRAY		░
Color.WHITE	(255, 255, 255)	⬜
Color.MAGENTA	(255, 0, 255)	🟪
Color.RED	(255, 0, 0)	🟥
Color.PINK		🟪
Color.ORANGE		🟧
Color.YELLOW	255, 255, 0)	🟨
Color.GREEN	(0, 255, 0)	🟩
Color.CYAN	(0, 255, 255)	🟦
Color.BLUE	(0, 0, 255)	🟦

☺ Color Constructors

Color(float r, float g, float b)

Creates an opaque RGB color with the specified red, green, and blue values in the range (0.0 - 1.0).

Color(int r, int g, int b)

Creates an opaque RGB color with the specified red, green, and blue values in the range (0 - 255).

… and several other constructors

Outline

Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

☺ Graphics

⇨ ☺ Applets

☺ Drawing Shapes

☺ Applets

- 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 be transported over the Web and executed using a web browser
- The browser is already running and calls the paint() method of the applet to paint part of the display
- Mostly, applets are not used any more

☺ Applets

- An applet also can be executed using the appletviewer tool of the Java SDK

```
C:\temp>appletviewer webPageFile.html
```

- The html file describes a web page and includes a request to run the applet
  - details below
- An applet doesn't have a `main` method
- Instead, there are several methods that the browser or appletviewer calls when needed.

☺ Applets

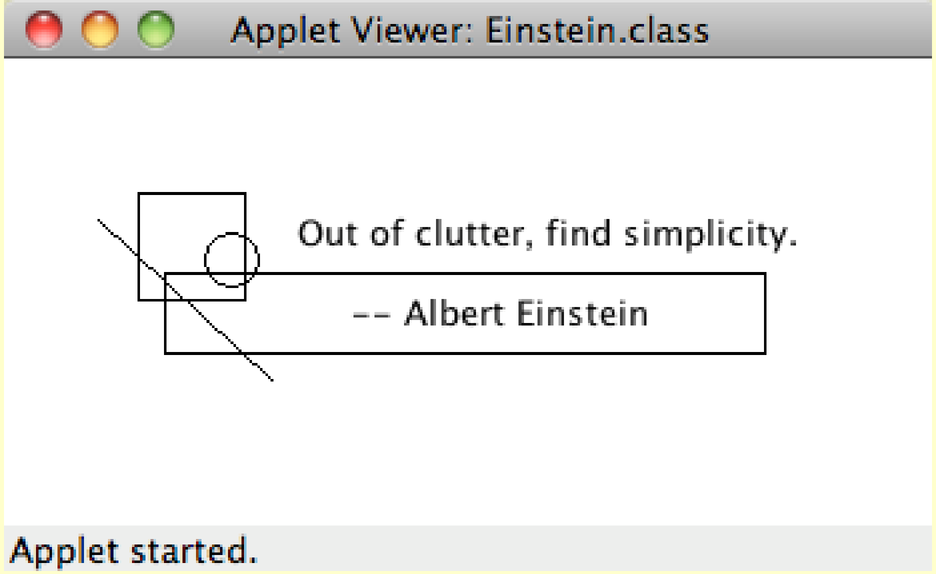
- The `paint` method is executed automatically whenever the applet's contents are drawn
- The `paint` method accepts a parameter that is an object of the `Graphics` class
- The web browser sets up this object and passes it to the applet
- A `Graphics` object defines a *graphics context* on which we can draw shapes and text
- The `Graphics` class has several methods for drawing shapes

☺ Applets

- We create an applet by *extending* the `JApplet` class
- The `JApplet` class is part of the `javax.swing` package
- This makes use of *inheritance*, which is explored in more detail in Chapter 8
- Applet classes must be declared public
- See `Einstein.java`

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

```
//*********                                                   **********
//  Einstei
//
//  Demonst
//*********                                                   **********

import java
import java

public clas
{
   //------                                                   ----------
   //  Draw                                                   s.
   //------                                                   ----------
   public v
   {
      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);
   }
}
```



Applet Viewer: Einstein.class

```
Out of clutter, find simplicity.

-- Albert Einstein
```

Applet started.

☺ The HTML applet Tag

- 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

```
<html>
   <head>
      <title>The Einstein Applet</title>
   </head>
   <body>
      <applet code="Einstein.class" width=350 height=175>
      </applet>
   </body>
</html>
```

☺ Running the Applet

- You need 3 files to execute this applet:
- Einstein.java
- Einstein.class – created by compiling the above
- Einstein.html – the HTML file (which could be named anything
- `C:\temp>appletviewer Einstein.html`
- this displays just the applet and skips the rest of the HTML
- BlueJ can also be used to do this.

Outline

Character Strings

Variables and Assignment

Primitive Data Types

Expressions

Data Conversion

Interactive Programs

☺ Graphics

☺ Applets

⇨ **☺ Drawing Shapes**

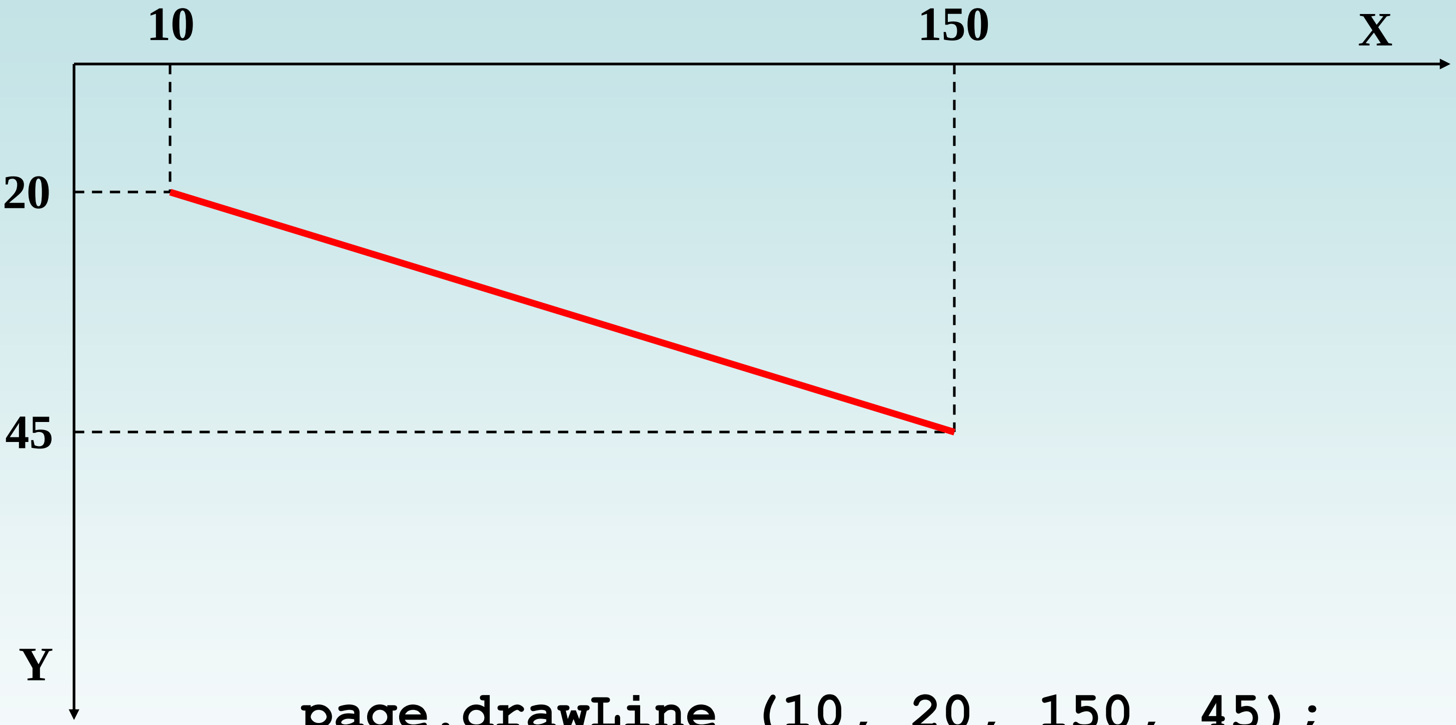
☺ Drawing Shapes

- Some methods of the `Graphics` class draw shapes
  - 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

Some Methods of the Graphics Class

- **public void drawString(String str, int x, int y):** is used to draw the specified string.
- **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
- **public void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
- **public void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
- **public void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
- **public void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
- **public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
- **public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
- **public void setColor(Color c):** is used to set the graphics current color to the specified color.

☺ Drawing a Line

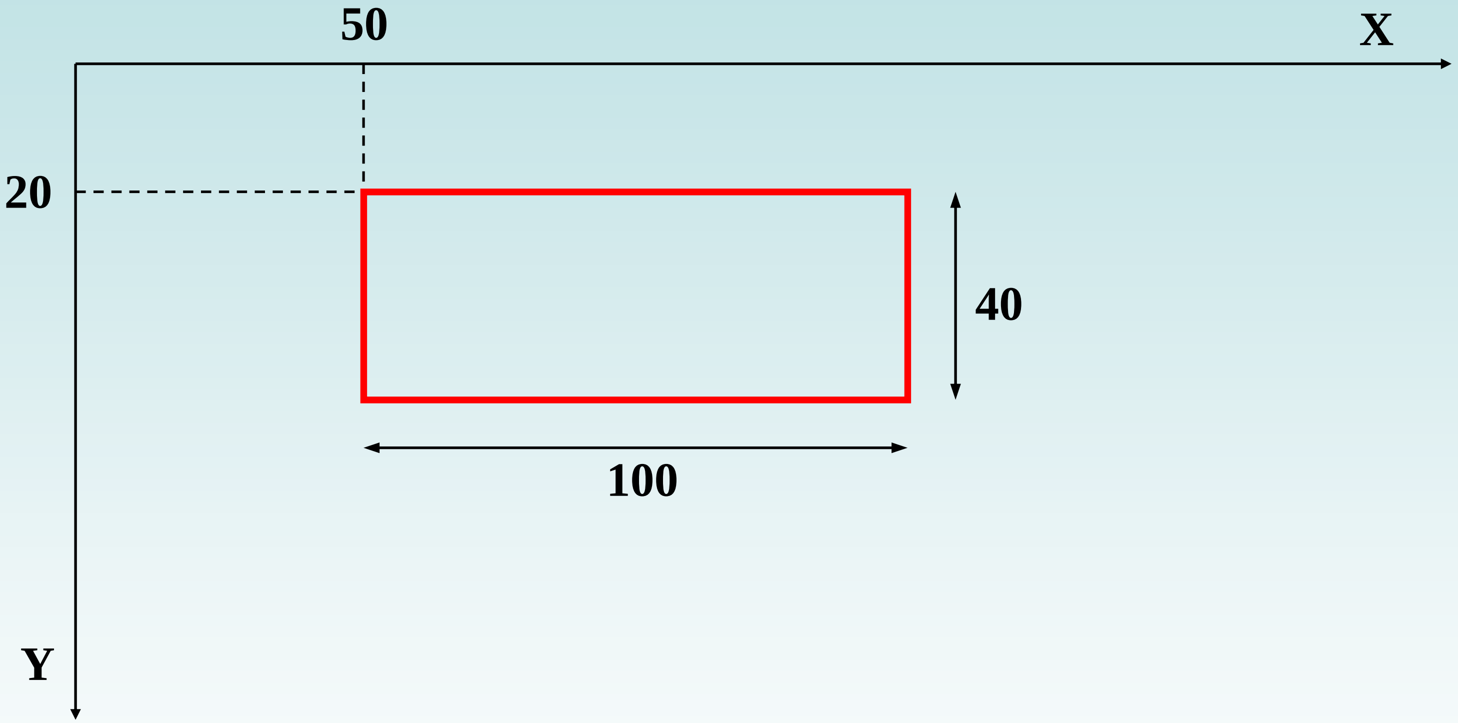


```
page.drawLine (10, 20, 150, 45);
```

or

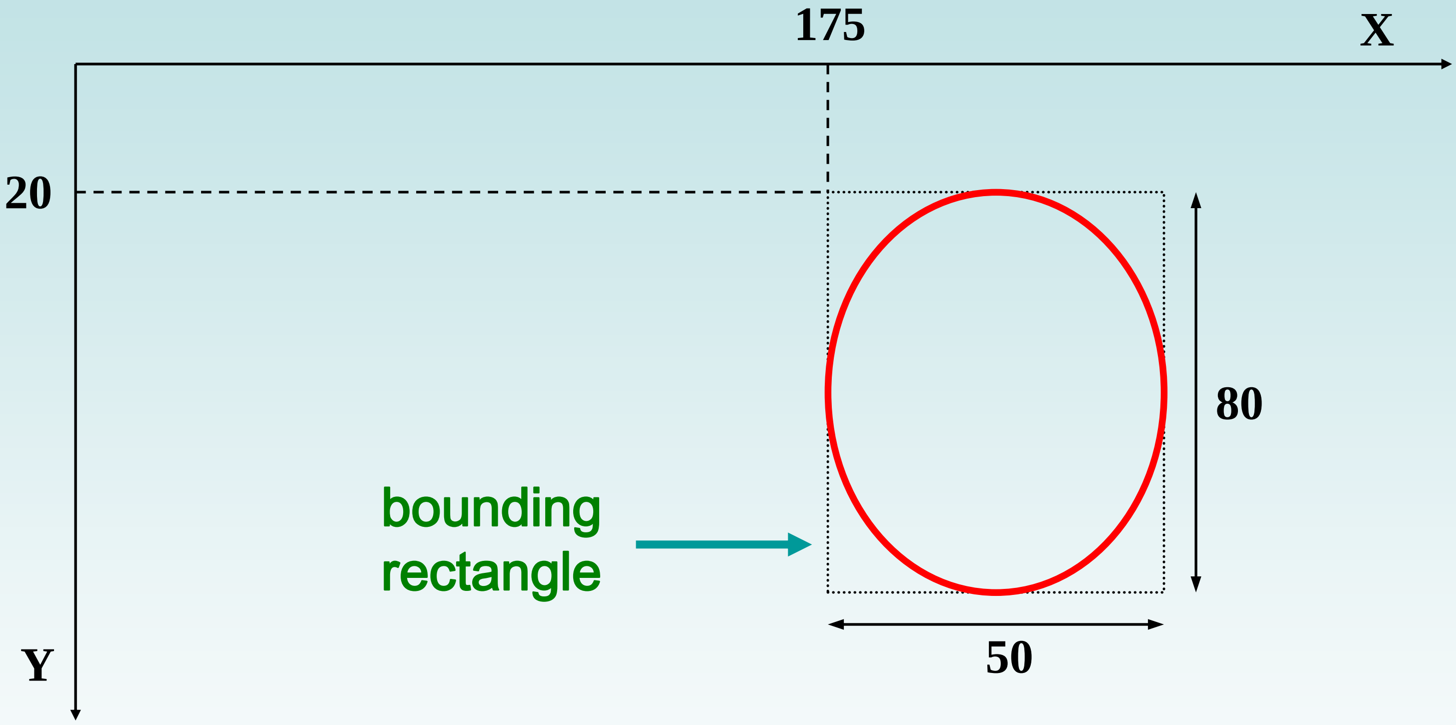
```
page.drawLine (150, 45, 10, 20);
```

☺ Drawing a Rectangle



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

☺ Drawing an Oval

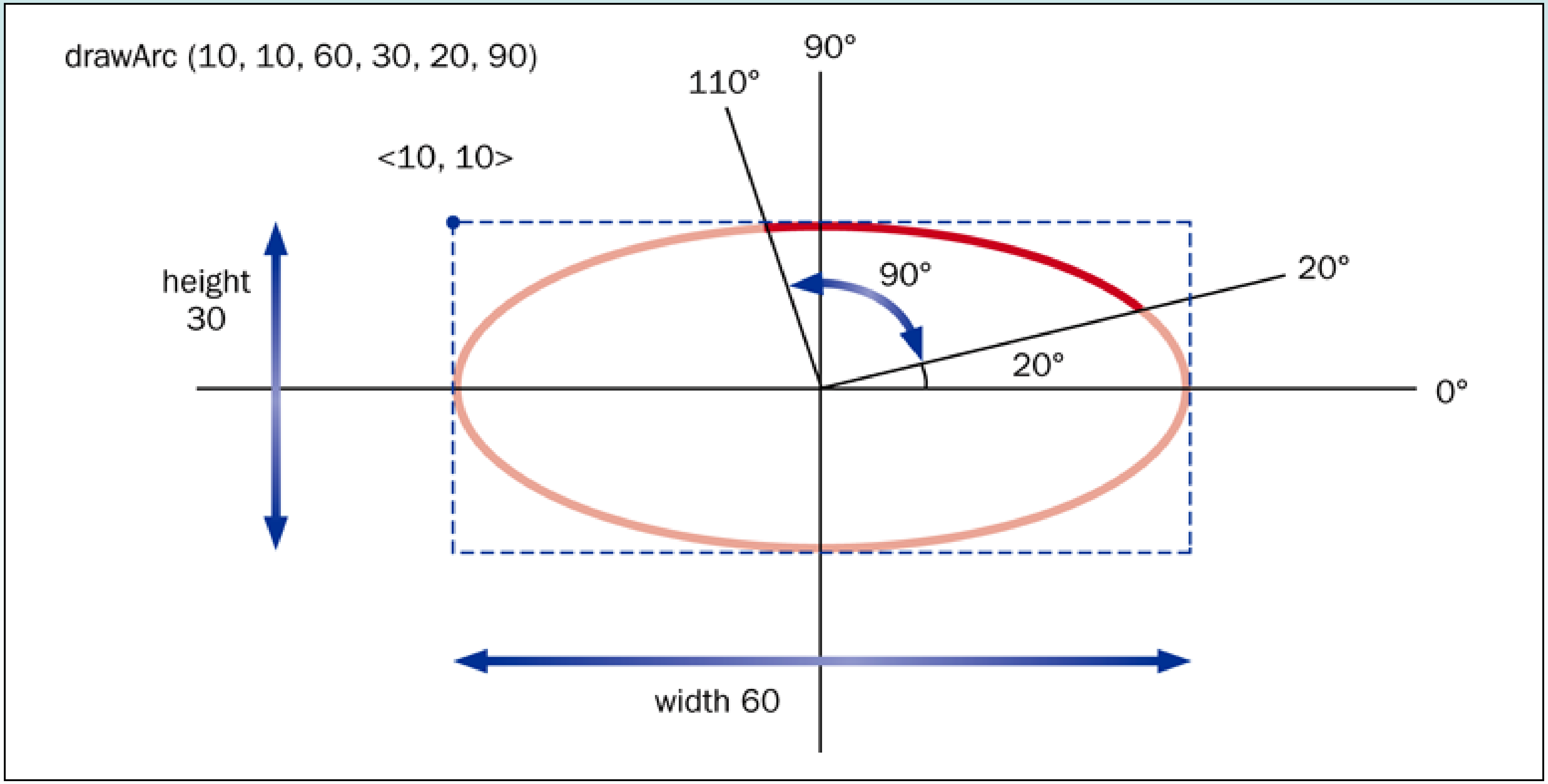


bounding rectangle

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

☺ Drawing an Arc

- An arc is defined by an oval, a start angle, and an arc angle:



☺ Drawing Shapes

- Every drawing surface has a *background color*
  - – like selecting a sheet of colored paper
- Every graphics context has a current *foreground color*
  - – like selecting a colored pen. You keep drawing with this color until you select another.
- Both can be set explicitly
- See `Snowman.java`

```java
//********************************************************************
//  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);       // select pen color
      page.fillRect(0, 175, 300, 50);  // draw the ground

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

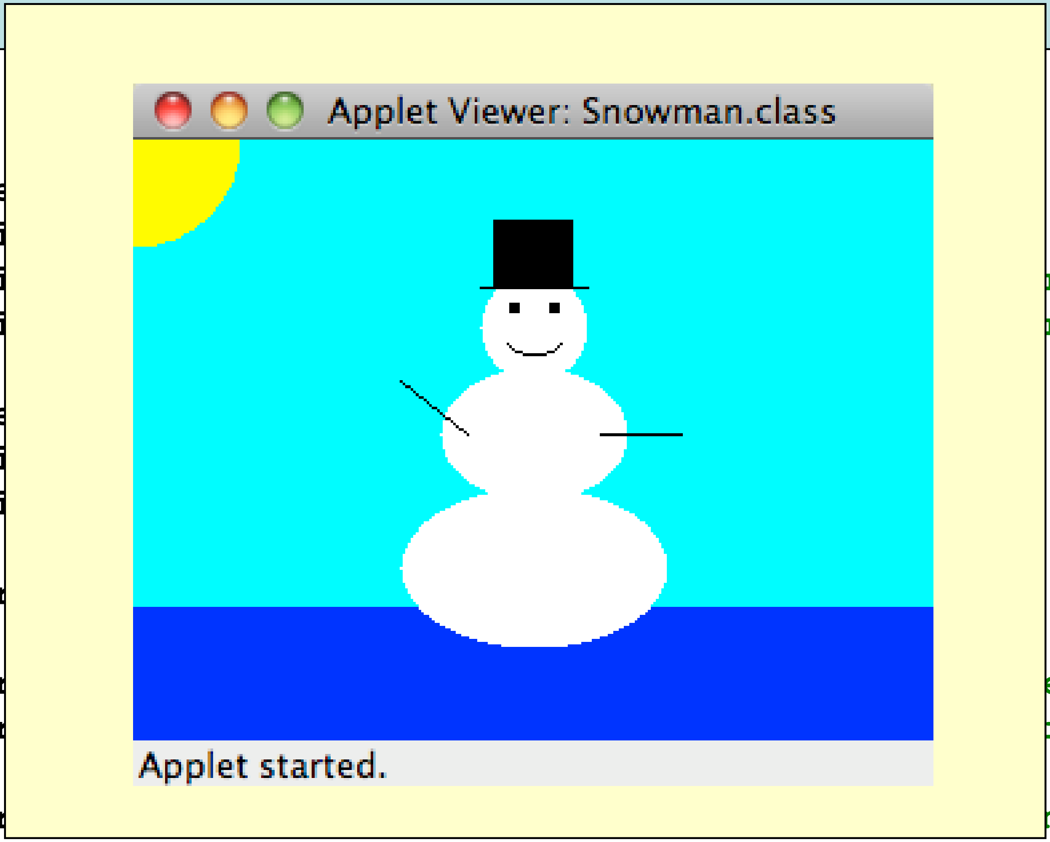
continued

continued

```
      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
   }
}
```

continued

```
      page.se
      page.fi
      page.fi                                              rso
      page.fi                                              rso

      page.se
      page.fi
      page.fi

      page.dr                                               smile

      page.dr                                              eft arm
      page.dr                                              ight arm

      page.dr                                              m of hat
      page.fillRect(MID-15, TOP-20, 30, 25);        // top of hat
   }
}
```



Applet started.

Copyright © 2014 Pearson Education, Inc.

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