

# Chapter 2:

# Primitive Data Types and Operations

# Variables

- *Variables* store data such as numbers and letters.
  - Think of them as places to store data.
  - They are implemented as memory locations.
- The data stored by a variable is called its *value*.
  - The value is stored in the memory location.
- Its value can be changed.

# Java Identifiers

- An *identifier* is a name, such as the name of a variable.
- Identifiers may contain only
  - Letters
  - Digits (0 through 9)
  - The underscore character (`_`)
  - And the dollar sign symbol (`$`) which has a special meaning
- The first character cannot be a digit.

# Java Identifiers

- Identifiers may not contain any spaces, dots (.), asterisks (\*), or other characters:

**7-11** `oracle.com` `util.*` (not allowed)

- Identifiers can be arbitrarily long.
- Since Java is *case sensitive*, `stuff`, `Stuff`, and `STUFF` are different identifiers.

# Keywords or Reserved Words

- Words such as **if** are called *keywords* or *reserved words* and have special, predefined meanings.
  - Cannot be used as identifiers.
  - See Appendix 1 for a complete list of Java keywords.
- Example keywords: **int**, **public**, **class**

# Primitive Data Types

**FIGURE 2.1** Primitive Type

Type Name	Kind of Value	Memory Used	Range of Values
<code>byte</code>	Integer	1 byte	-128 to 127
<code>short</code>	Integer	2 bytes	-32,768 to 32,767
<code>int</code>	Integer	4 bytes	-2,147,483,648 to 2,147,483,647
<code>long</code>	Integer	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>float</code>	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
<code>double</code>	Floating-point	8 bytes	$\pm 1.79769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$
<code>char</code>	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
<code>boolean</code>		1 bit	True or false

# Primitive Data Types

- Four integer types (**byte**, **short**, **int**, and **long**)
  - **int** is most common
- Two floating-point types (**float** and **double**)
  - **double** is more common
- One character type (**char**)
- One boolean type (**boolean**)

# Examples of Primitive Values

- Integer types

`0 -1 365 12000`

- Floating-point types

`0.99 -22.8 3.14159 5.0`

- Character type

`'a' 'A' '#' ' '`

- Boolean type

`true false`



# Declaring Variables

- A variable must be declared *only once* before it is used.
- When you *declare* a variable, you provide its type and name.
- Syntax:

```
datatype VARNAME;
```

```
int x;           // Declare x to be an  
                // integer variable;
```

```
double radius; // Declare radius to  
                // be a double variable;
```

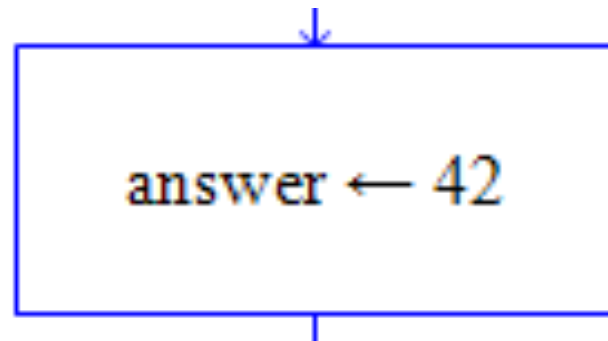
```
char a;        // Declare a to be a  
                // character variable;
```

# Assignment Statements

- An assignment statement is used to assign a value to a variable.

```
answer = 42 ;
```

- In Raptor, this is equivalent to the following block:



# Assignment Statements

- Syntax

**variable = expression**

where **expression** can be another variable, a *literal* (such as a number or character), or something more complicated which combines variables and literals using *operators* (such as + and -)

# Assignment Examples

```
amount = 3.99;
```

```
firstInitial = 'W';
```

```
score = numberOfCards + handicap;
```

```
eggsPerBasket = eggsPerBasket - 2;
```

# Assignment Evaluation

- The expression on the right-hand side of the assignment operator (=) is evaluated first.
- The result is used to set the value of the variable on the left-hand side of the assignment operator.

```
score = numberOfCards + handicap;  
eggsPerBasket = eggsPerBasket - 2;
```

# Declaring and assignment *in one step*

- Syntax:

```
datatype VARANTNAME = VALUE;
```

One Step	Two Steps
<code>int x = 1;</code>	<code>int x;</code> ← declaration <code>x = 1;</code> ← assignment
<code>double d = 1.4;</code>	<code>double d;</code> <code>d = 1.4;</code>

# Constants

- Adding the keyword *final* at the beginning of variable declaration, will make the value assigned to that variable final (i.e. cannot be changed later).

- Syntax:

```
final datatype CONSTANTNAME = VALUE;
```

- This is useful for defining constants such as:

```
final double PI = 3.14159;
```

```
final int SIZE = 3;
```

# Assignment Compatibilities

- Java is said to be *strongly typed*.
  - You can't, for example, assign a floating point value to a variable declared to store an integer.

```
int Var = 4.5; ❌
```

- Sometimes conversions between numbers are possible. For example:

```
double Var = 7;
```

is possible. The value in Var will be 7.0.



# Assignment Compatibilities

- A value of one type can be assigned to a variable of any type further to the right

**byte --> short --> int --> long  
--> float --> double**

- But not to a variable of any type further to the left.

# Type Casting

- A *type cast* temporarily changes the value of a variable from the declared type to some other type.

- For example,

```
double distance;
```

```
distance = 9.0;
```

```
int points;
```

```
points = (int)distance;
```

- Illegal without `(int)`
- When casting from double to int, any value to the right of the decimal point is *truncated* rather than *rounded*.

# Arithmetic Operators

- Arithmetic expressions can be formed using the **+**, **-**, **\***, and **/** operators together with variables or numbers referred to as *operands*.
  - **When both operands are of the same type, the result is of that type.**
  - **When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.**

# Arithmetic Operations

- Example

If `hoursWorked` is an `int` to which the value `40` has been assigned, and `payRate` is a `double` to which `8.25` has been assigned

`hoursWorked * payRate`

is a `double` with a value of `500.0`.

# The Division Operator

- The division operator (`/`) behaves as expected if one of the operands is a floating-point type.
- When both operands are integer types, the result is truncated, not rounded.
  - Hence, `99/100` has a value of `0`.

# The **mod** Operator

- The **mod** (%) operator is used with operators of integer type to obtain the remainder after integer division.
- 14 divided by 4 is 3 *with a remainder of 2*.
  - Hence, **14 % 4** is equal to **2**.
- The mod operator has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer.

# Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed

- examples:

`(cost + tax) * discount`

`cost + (tax * discount)`

- Without parentheses, an expressions is evaluated according to the *rules of precedence*.

# Precedence Rules

Operators	Precedence
( )	first (highest)
* , / , %	second
+ , -	third (lowest)

- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.



# Precedence Rules - example

Pass1:  $x = 4 / 2 + 8 * 4 - \underline{(5 + 2)} \% 3$

Pass2:  $x = \underline{4 / 2} + 8 * 4 - 7 \% 3$

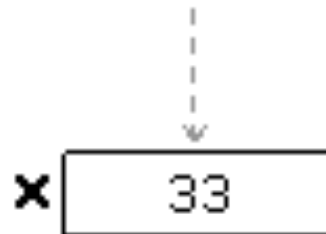
Pass3:  $x = 2 + \underline{8 * 4} - 7 \% 3$

Pass4:  $x = 2 + 32 - \underline{7 \% 3}$

Pass5:  $x = 2 + \underline{32} - 1$

Pass6:  $x = \underline{34} - 1$

Pass7:  $x = \underline{33}$



# Sample Expressions

Ordinary Math	Java (Preferred Form)
$rate^2 + delta$	<code>rate * rate + delta</code>
$2(salary + bonus)$	<code>2 * (salary + bonus)</code>
$\frac{1}{time + 3mass}$	<code>1 / (time + 3 * mass)</code>
$\frac{a - 7}{t + 9v}$	<code>(a - 7) / (t + 9 * v)</code>

# Shortcut Assignment Operators

- Assignment operators can be combined with the arithmetic operators: `-`, `*`, `/`, and `%`.

For example:

```
amount = amount + 5;
```

can be written as

```
amount += 5;
```

yielding the same results.

# Shortcut Assignment Operators

<i>Operator</i>	<i>Example</i>	<i>Shortcut</i>
<b>+=</b>	<b>i = i + 8</b>	<b>i += 8</b>
<b>-=</b>	<b>i = i - 8</b>	<b>i -= 8</b>
<b>*=</b>	<b>i = i * 8</b>	<b>i *= 8</b>
<b>/=</b>	<b>i = i / 8</b>	<b>i /= 8</b>
<b>%=</b>	<b>i = i % 8</b>	<b>i %= 8</b>

# Increment and Decrement Operators

- Used to increase (or decrease) the value of a variable by 1
- Easy to use, important to recognize
- The increment operator  
`count++` or `++count`
- The decrement operator  
`count--` or `--count`

# Increment and Decrement Operators

- equivalent operations

```
count++;
```

```
++count;
```

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

```
count--;
```

```
--count;
```

```
count = count - 1;
```

# Increment and Decrement Operators in Expressions

```
int i = 10;  
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i  
i = i + 1;
```

```
int i = 10;  
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;  
int newNum = 10 * i;
```

# The **String** Type

- Declaring & Assignment

```
String greeting;
```

```
greeting = "Hello!";
```

or

```
String greeting = "Hello!";
```

- Printing

```
System.out.println(greeting);
```



# Concatenation of Strings

- Two strings are *concatenated* using the **+** operator.

```
String greeting = "Hello";  
String sentence;  
sentence = greeting + " officer";  
System.out.println(sentence);
```

- Any number of strings can be concatenated using the **+** operator.

# Concatenating Strings and Integers

```
String solution;  
solution = "The answer is " + 42;  
System.out.println (solution);
```



The answer is 42

# Escape Characters

- How would you print

`"Java" refers to a language.`

- The compiler needs to be told that the quotation marks (") do not signal the start or end of a string, but instead are to be printed.

`System.out.println("\"Java\" refers to a language.");`

# Escape Characters

```
\\" Double quote.  
\' Single quote.  
\\ Backslash.  
\n New line. Go to the beginning of the next line.  
\t Tab. Add whitespace up to the next tab stop.
```

- Each escape sequence is a single character even though it is written with two symbols.

# Examples

```
System.out.println("abc\\def");
```



abc\def

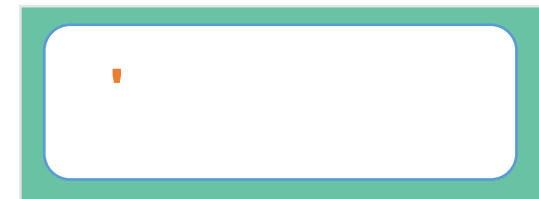
```
System.out.println("new\nline");
```



new  
line

```
char singleQuote = '\'';
```

```
System.out.println  
    (singleQuote);
```



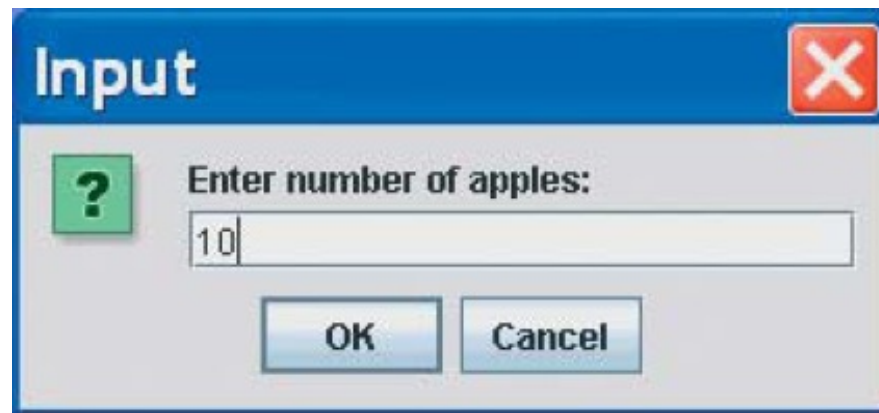
'

# Getting the input from the user

- Use `JOptionPane.showInputDialog()` to get an input from the user. For example, the statement:

```
String instr = JOptionPane.showInputDialog("Enter the number of apples");
```

shows the following dialog box:



- If the user entered the number `10` then clicked OK, the string `"10"` will be stored in the variable `instr`.

# Converting Strings to Integers

- The input returned from the input dialog box is always a string. If the user enters a numeric value such as **123**, it returns **"123"**. To obtain the input as a number, you have to convert the string into a number.

- To convert a string into an integer value, use the method **Integer.parseInt()** as follows:

```
int num = Integer.parseInt(instr) ;
```

Where **instr** is a numeric string such as **"123"**.

# Converting Strings to Doubles

- To convert a string into a double value, use the method `Double.parseDouble()` as follows:

```
double num = Double.parseDouble(instr) ;
```

Where `instr` is a numeric string such as `"123.45"`.



# Programming Errors

- **Syntax Errors**
  - Detected by the compiler
- **Runtime Errors**
  - Causes the program to abort
- **Logic Errors**
  - Produces incorrect or unexpected result

# Syntax Errors

```
public class ShowSyntaxErrors
{
    public static void main(String[] args)
    {
        i = 30;
        System.out.println(i + 4);
    }
}
```

# Runtime Errors

```
public class ShowRuntimeErrors
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1 / x;
    }
}
```

# Logic Errors

```
public class ShowLogicErrors
{
    public static void main(String[] args)
    {
        double a = 6;
        double b = 4;
        double average = a + b / 2;
        System.out.println("The average of " + a +
            " and " + b + " is: " + average);
    }
}
```

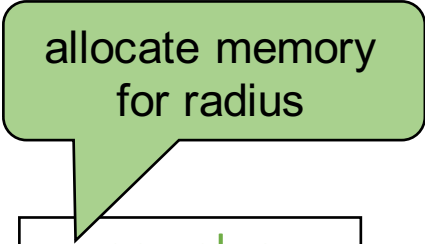
# Debugging

- Logic errors are called *bugs*.
- The process of finding and correcting logic errors is called *debugging*.
- The utility that helps you find these bugs is called a *debugger*.
- You can use the debugger to set a *breakpoint* at the line where you want the program execution to pause, then you could trace the program execution from that point step by step.

# Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

radius



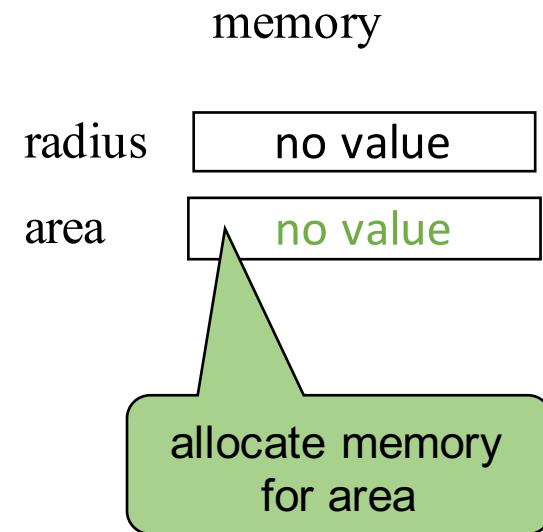
allocate memory  
for radius



no value

# Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```



# Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

radius

20

area

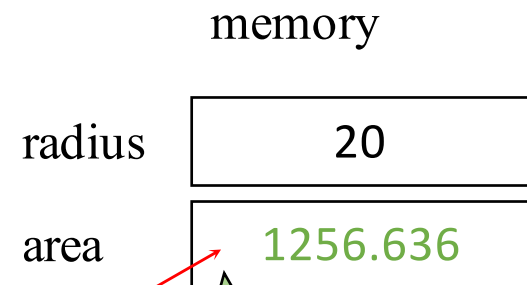
no value

assign 20 to radius



# Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```



compute area and assign  
it to variable area

# Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

memory

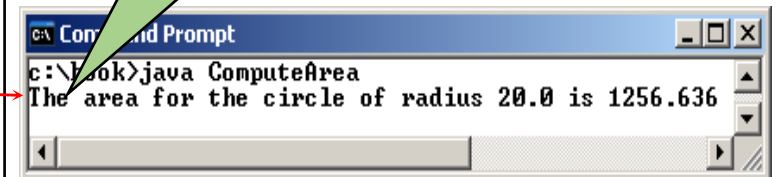
radius

20

area

1256.636

print a message to the console



```
CA Command Prompt  
c:\book>java ComputeArea  
The area for the circle of radius 20.0 is 1256.636
```