

# Chapter 2

## Elementary Programming

# Objectives

- To write Java programs to perform simple calculations
- To obtain input from the console using the Scanner class
- To use identifiers to name variables, constants, methods, and classes
- To use variables to store data
- To program with assignment statements and assignment expressions
- To use constants to store permanent data
- To declare Java primitive data types: byte, short, int, long, float, double, boolean, and char
- To use Java operators to write numeric expressions
- To cast value of one type to another type
- To know common errors in Java

# Introducing Programming with an Example

## Computing the Area of a Circle

This program computes the area of the circle.

# Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20.0;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

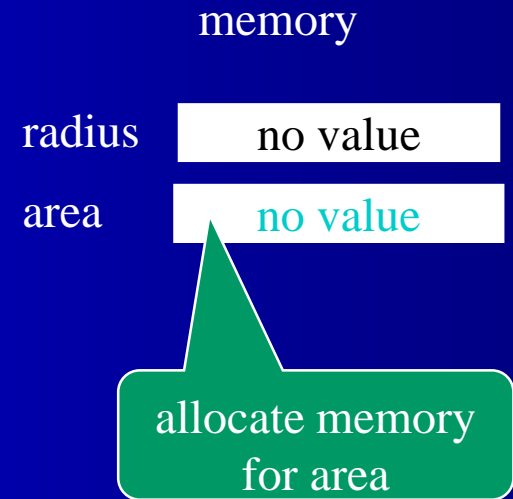
allocate memory  
for radius

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 = 20.0;  
  
        // 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 = 20.0;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

radius

area

assign 20 to radius

20.0

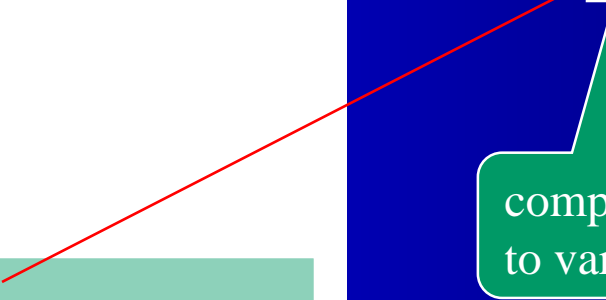
no value

# Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20.0;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

memory

|        |          |
|--------|----------|
| radius | 20.0     |
| area   | 1256.636 |



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 = 20.0;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

memory

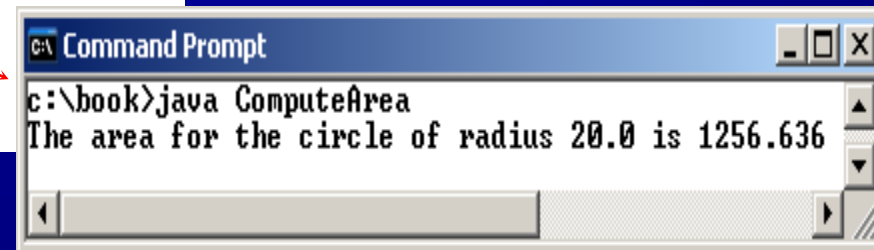
radius

20.0

area

1256.636

print a message to the  
console



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



# Reading Input from the Console

## 1. Create a Scanner object

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

## 2. Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value.

For example,

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter a double value: ");  
double doubleValue = input.nextDouble();
```

# Example 1

```
// Scanner is stored in java.util package
import java.util.Scanner;
public class ComputeAreaWithCnsoleInput
{
    public static void main(String[] args)
    {
        //create a Scanner object
        Scanner input = new Scanner(System.in);

        //Prompt the user to enter a radius
        System.out.print("Enter a number for radius (double): ");
        double radius = input.nextDouble();

        //Compute area
        double area = radius * radius * 3.14159;

        //Display results
        System.out.println("The area for the circle of radius "
                           + radius + " is " + area);
    }
}
```

## Example 2

```
// Scanner is stored in java.util package
import java.util.Scanner;
public class ComputeAverage
{
    public static void main(String[] args)
    {
        // create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter three numbers
        System.out.print("Enter three numbers (type double): ");
        double number1 = input.nextDouble();
        double number2 = input.nextDouble();
        double number3 = input.nextDouble();

        // Compute average
        double average = (number1 + number2 + number3) / 3.0;

        // Display results
        System.out.println("The average of " + number1 + " " +
                           number2 + " " + number3 + " is " + average);
    }
}
```

# Identifiers

- ❑ Identifier is a name for an element in the program, such as variable, class, and method.
- ❑ An identifier is a sequence of characters that consist of letters, digits, underscores (\_), and dollar signs (\$).
- ❑ An identifier must start with a letter, an underscore (\_), or a dollar sign (\$). It cannot start with a digit.
- ❑ An identifier cannot be a reserved word. (Java Keywords).
- ❑ An identifier can be of any length.

# Reserved Words

## Java reserved words:

|                       |                         |                        |                           |
|-----------------------|-------------------------|------------------------|---------------------------|
| <code>abstract</code> | <code>else</code>       | <code>interface</code> | <code>switch</code>       |
| <code>assert</code>   | <code>enum</code>       | <code>long</code>      | <code>synchronized</code> |
| <code>boolean</code>  | <code>extends</code>    | <code>native</code>    | <code>this</code>         |
| <code>break</code>    | <code>false</code>      | <code>new</code>       | <code>throw</code>        |
| <code>byte</code>     | <code>final</code>      | <code>null</code>      | <code>throws</code>       |
| <code>case</code>     | <code>finally</code>    | <code>package</code>   | <code>transient</code>    |
| <code>catch</code>    | <code>float</code>      | <code>private</code>   | <code>true</code>         |
| <code>char</code>     | <code>for</code>        | <code>protected</code> | <code>try</code>          |
| <code>class</code>    | <code>goto</code>       | <code>public</code>    | <code>void</code>         |
| <code>const</code>    | <code>if</code>         | <code>return</code>    | <code>volatile</code>     |
| <code>continue</code> | <code>implements</code> | <code>short</code>     | <code>while</code>        |
| <code>default</code>  | <code>import</code>     | <code>static</code>    |                           |
| <code>do</code>       | <code>instanceof</code> | <code>strictfp</code>  |                           |
| <code>double</code>   | <code>int</code>        | <code>super</code>     |                           |

# Variable Declaration

A *variable* is a name for a location in memory to store data of specific type.

A variable must be *declared* by specifying 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.

```
// declare and initialize  
int sum;  
sum = 0;  
int base = 32;  
double max = 149.75;
```

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

# Examples

```
// Compute the area
double radius; // declaration
double area;   // declaration
radius = 1.0;  // initialization
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius "
    +radius);
```

```
// Compute the area
double radius = 2.0; // declaration and initialization
double area = radius * radius * 3.14159; //same here
System.out.println("The area is " + area + " for radius "
    +radius);
```



# More Variables

Must declare all variables to let the program know what to store in the variables.

```
int grade;          // Declare grade as integer variable
double radius;     // Declare radius as double variable
float speed;        // Declare speed as float variable
char letter;        // Declare letter as character variable
boolean flag;       // Declare flag as boolean variable
short price;        // Declare price as short variable
long quantity;      // Declare quantity as long variable
```

# Variable Initialization Example

```
// Prints the number of keys on a piano.
public class PianoKeys
{
    public static void main (String[] args)
    {
        int keys = 88; //declare and initialize
        System.out.println ("A piano has " + keys + " keys.");
    }
}
```

**Output:** A piano has 88 keys.

# Declaring and Initializing in One Step

```
int x = 1;  
double d = 1.4;
```

**Same as:**


```
int x;  
x = 1;  
  
double d;  
d = 1.4;
```

# Assignment

An *assignment statement* changes the value of a variable

The assignment operator is the = sign

```
int total;  
total = 55;
```



The expression on the right is evaluated and its result is stored in the variable on the left.

The value that was in total is overwritten.

You can only assign a value to a variable that is consistent with the variable's declared type.

See program **Geometry.java** next slide.

# Assignment - Example

```
// Print the number of sides of several geometric shapes.
public class Geometry {
    public static void main (String[] args) {
        int sides = 7; // declare and initialize
        System.out.println ("A heptagon has " + sides + " sides.");

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

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

# Assignment Statement Examples

```
classSize = 40;    // Assign 40 to classSize  
radius = 3.0;      // Assign 3.0 to radius  
letter = 'A';      // Assign 'A' to letter  
answer = true;     // Assign true to answer  
  
//compute and assign to circleArea  
circleArea = radius * radius * Math.PI;
```

# Constants

A *constant* is an identifier that is similar to a variable except that it holds the same value during its entire existence

As the name implies, it is constant, not variable.

The compiler will issue an error if you try to change the value of a constant.

In Java, we use the **final** modifier to declare a constant, such as:

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

```
final boolean DEFAULT_ANSWER = true;
```

# Constants

Constants are useful for three important reasons:

- First, they give meaning to otherwise unclear literal (numeric) values.
- Second, they facilitate program maintenance so you make the value change in one place.
- Third, they help avoid inadvertent errors by other programmers.



# Constants - Examples

Format:

```
final datatype CONSTANT_NAME = Value;
```

Examples:

```
final int CLASS_SIZE = 40;
```

```
final double MATH_PI = 3.14159;
```

```
final char FAIL_GRADE = 'F';
```

```
final boolean FLAG = true;
```

# Primitive Data Types

There are eight primitive data types in Java:

Four of them represent **integer** numbers:

**byte, short, int, long**

Two of them represent **floating point** numbers:

**float, double**

One of them represents **characters**:

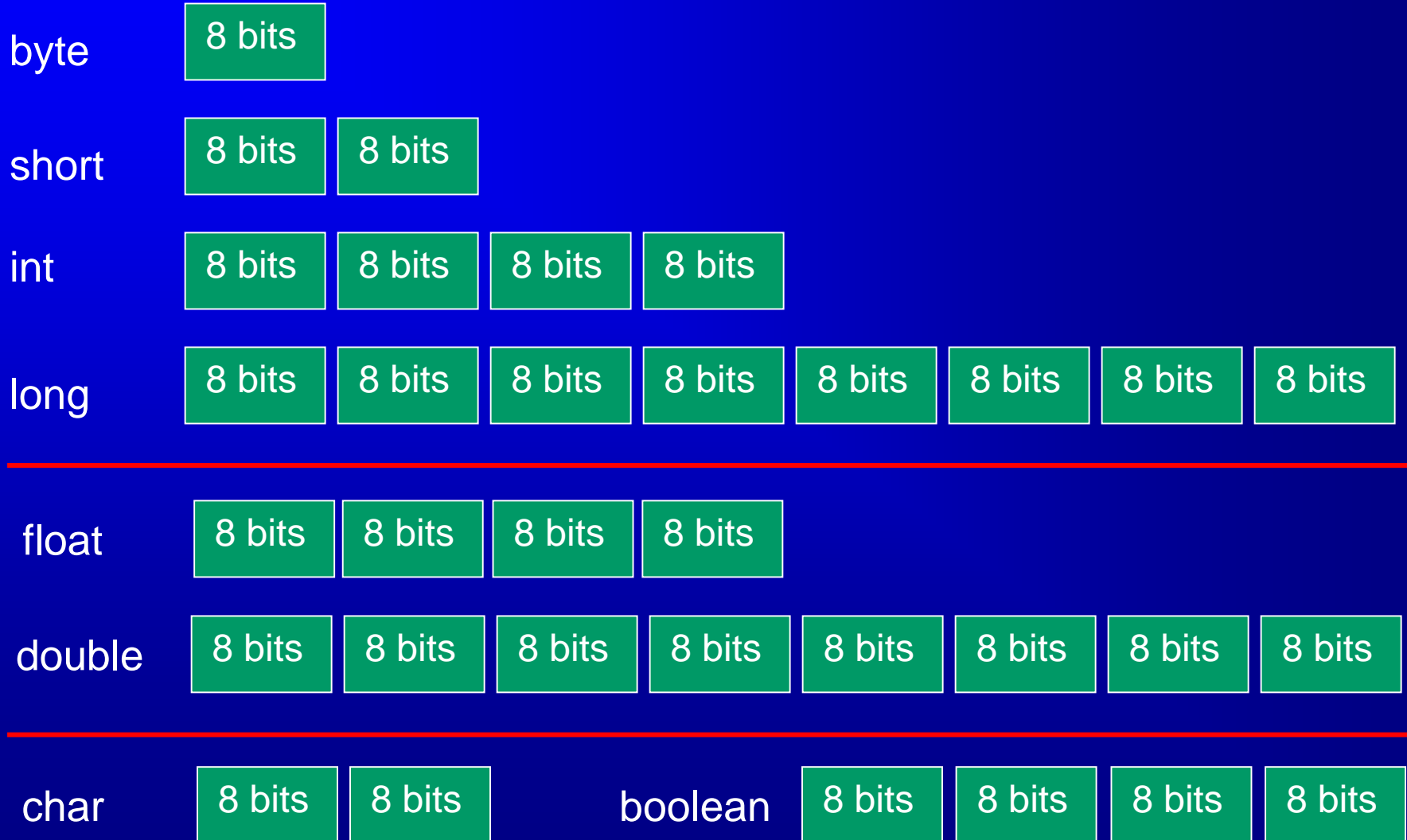
**char**

And one of them represents **boolean** values:

**boolean**

Everything else in Java is of type **object**.

# Sizes of Primitive Numeric Types



# Values of Primitive Numeric Types

The difference between the various numeric primitive types is their memory size:

| <u>Type</u> | <u>Storage</u> | <u>Min Value</u>                                     | <u>Max Value</u>     |
|-------------|----------------|--|----------------------|
| 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 |                      |
| boolean     | 32 bits        | Either <code>true</code> or <code>false</code>       |                      |

# Numeric Operators

| Name | Meaning        | Example      | Result |
|------|----------------|--------------|--------|
| +    | Addition       | $34 + 1$     | 35     |
| -    | Subtraction    | $34.0 - 0.1$ | 33.9   |
| *    | Multiplication | $300 * 30$   | 9000   |
| /    | Division       | $1.0 / 2.0$  | 0.5    |
| %    | Remainder      | $20 \% 3$    | 2      |

# Integer Division and Remainder

`5 / 2` yields an integer 2

`5.0 / 2` yields a double value 2.5

`5 % 2` yields 1 (remainder value)

## **Note**

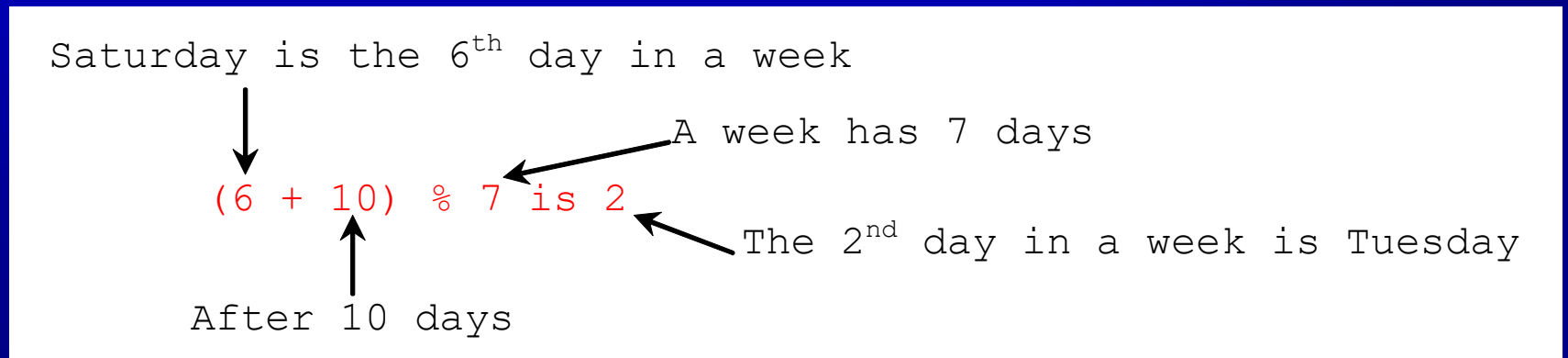
`5 / 2.0` OR `5.0 / 2` ==> 2.5

# Remainder Operator - Example

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd.

Suppose today is Saturday. You and your friends are going to meet in 10 days from today. What day is would that be?

You can find that out using the following expression:



# Number Literals

A *literal* is a **constant value** that appears directly in the program. For example, 40, 1000000, 5.0, true, and A are literals in the following statements:

```
int classSize = 40;  
long largeNumber = 1000000;  
double increment = 5.0;  
boolean defaultAnswer = true;  
char letterGrade = 'A';
```



# Arithmetic Expressions

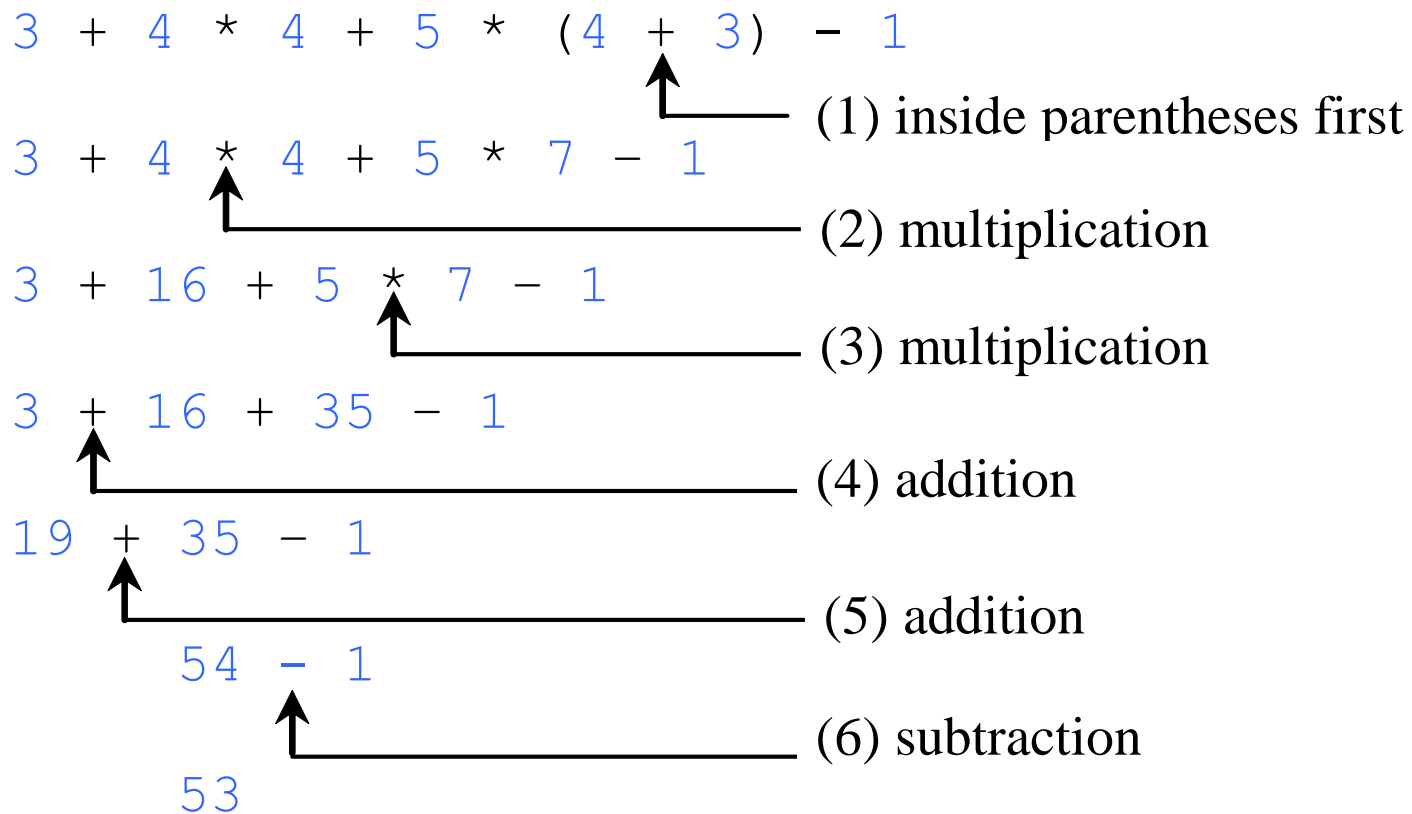
$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

is translated to

$$(3+4*x) / 5 - 10 * (y-5) * (a+b+c) / x + 9 * (4/x + (9+x) / y)$$

# How to Evaluate an Expression

Java uses arithmetic precedence rule for evaluating expression.



# How to Evaluate an Expression

What is the order of evaluation in the following expressions?

$$a + b + c + d + e$$

1 2 3 4

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

3 1 4 2

$$a / (b + c) - d \% e$$

2 1 4 3

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

4 3 2 1

# Problem: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\frac{5}{9})(fahrenheit - 32)$$

```
Double celsius = (5.0/9.0) * (fahrenheit - 32);
```

# Shortcut Assignment Operators

| <i>Operator</i> | <i>Example</i>        | <i>Equivalent</i> <small>(recommended use)</small> |
|-----------------|-----------------------|--|
| <code>+=</code> | <code>i += 8</code>   | <code>i = i + 8</code>                             |
| <code>-=</code> | <code>f -= 8.0</code> | <code>f = f - 8.0</code>                           |
| <code>*=</code> | <code>i *= 8</code>   | <code>i = i * 8</code>                             |
| <code>/=</code> | <code>i /= 8</code>   | <code>i = i / 8</code>                             |
| <code>%=</code> | <code>i %= 8</code>   | <code>i = i % 8</code>                             |

# Increment and Decrement Operators

| Operator     | Name          | Description   |
|--------------|---------------|---|
| <u>++var</u> | preincrement  | The expression (++var) increments <u>var</u> by 1 and evaluates to the <i>new</i> value in <u>var</u> <i>after</i> the increment. |
| <u>var++</u> | postincrement | The expression (var++) evaluates to the <i>original</i> value in <u>var</u> and increments <u>var</u> by 1.                       |
| <u>--var</u> | predecrement  | The expression (--var) decrements <u>var</u> by 1 and evaluates to the <i>new</i> value in <u>var</u> <i>after</i> the decrement. |
| <u>var--</u> | postdecrement | The expression (var--) evaluates to the <i>original</i> value in <u>var</u> and decrements <u>var</u> by 1.                       |

# Increment and Decrement Operators, cont.

```
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;
```

# Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also makes them *complex and difficult to read*.

Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this:

```
int i = 10;  
int k = ++i + i;
```

Output:

```
i = 11  
k = 22
```



# Type Conversion

Conversions must be handled carefully to avoid losing information.

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

*Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as `int` to `short`)

In Java, data conversions can occur in three ways:

- *assignment conversion*
- *promotion (temporary conversion)*
- *casting (explicit conversion)*

# Assignment Conversion

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

If `money` is a `float` variable and `dollars` is an `int` variable, the following assignment converts the value in `dollars` to a `float`.

```
double money = 0.0;  
int dollars = 10;  
money = dollars;    // money has value 10.0  
dollars = money;    // ERROR!!!!
```

Only widening conversion can happen via assignment.

Note that the value or type of `dollars` did not change.

# Promotion Conversion

*Promotion* happens automatically (and temporarily) when operators in expressions convert their operands.

Example :

```
int count = 5;
double sum = 20.0;
double result;
result = sum/count; //result contains 4.0

result = count/sum; //result contains 0.25
```

The value of `count` is temporarily promoted (converted) to a floating point value to perform the calculation. `count` is still of type integer.

# Casting Conversion

*Casting* is the most powerful, and dangerous, technique for conversion.

Both **widening** and **narrowing** conversions can be accomplished by explicit casting.

To cast, the **type name** is put in parentheses in front of the value being converted.

## Example

```
int total = 20, count = 5;
double result1, result2;
. . .
result1 = (float)total/count; //casting and promotion
result2 = (float)(total/count); //casting only
```

# Casting Conversion

How do we solve the problem of  $3 / 2$  having a result of 1.

You can make one of the values floating point by adding .0 to it (conversion by promotion)

`3.0 / 2`

`3 / 2.0`

The result type will then be floating point value (1.5)

Or

You can cast one of the values to either float or double (conversion by casting)

`(double) 3 / 2`

`3 / (float) 2`

# Conversion Rules

When performing an operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.

# Type Casting Examples

## Implicit casting

```
double d = 3; //type widening with assignment
```

## Explicit casting

```
int i = (int)3.0; //explicit type narrowing
```

```
int i = (int)3.9; //fraction part is truncated
```

What is wrong here? `int x = 5 / 2.0; //syntax error`

range increases



byte, short, int, long, float, double

# Reading Characters

```
//Characters are read as strings
```

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

```
System.out.print("Enter a character: ");
```

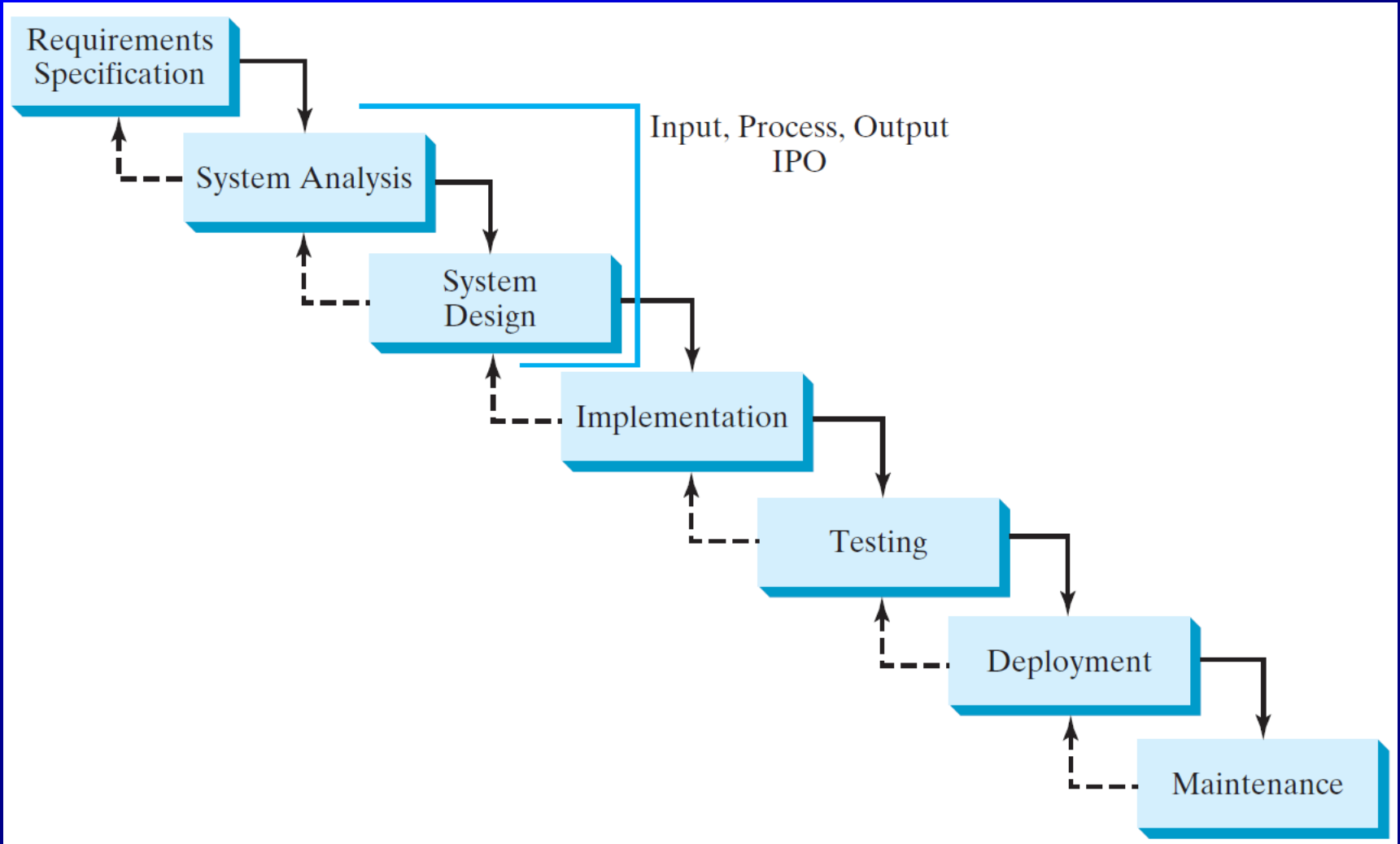
```
String s = input.nextLine(); //must press the Enter key
```

```
char ch = s.charAt(0);
```

```
System.out.println("The entered character is " + ch);
```



# Software Development Process



# Software Development Process

In this course, you need to start with:

1. Understand the problem.
2. Identify what the input and outputs.
3. Decide how to process the inputs to produce the outputs (algorithm).
4. Write down the logical steps of the algorithm.
5. Translate the algorithmic steps into Java code.
6. Type code, compile, fix errors, run, and test the program for correctness of outputs.

# Problem: Monetary Units

This program allows the user to enter the amount representing dollars and cents (in decimal) and output a report listing the monetary equivalence in coins (dollars, quarters, dimes, nickels, and pennies). The program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

# Problem: Monetary Units

Development steps:

1. Understand the problem: *The program reads an amount of money such as \$1.41 and prints out the change.*
2. Ask the user to enter an amount (of type double).
3. Convert the amount to pennies (cents) (easier to work with integer type). Use casting to convert double type to int type.
3. Divide the amount by 100 to get # of dollar coins (using / operator).
4. Determine the remaining change (use remainder operator %).
5. Repeat steps 3 and 4 to get the quarters, dimes, and nickels.
6. The final remaining change amount is the number of pennies.
7. Display the outputs with proper labels.

# Problem: Monetary Units

```
import java.util.Scanner;
public class ComputeChange {
    public static void main(String[] args) {
        // Create a Scanner
        // Read amount from user (double, e.g., 2.37)
        // Convert input amount to cents (int, e.g., 237)
        // Other code

        // Find the number of one dollars
        int numberOfOneDollars = remainingAmount / 100;
        remainingAmount = remainingAmount % 100;

        // Find the number of quarters in the remaining amount
        int numberOfQuarters = remainingAmount / 25;
        remainingAmount = remainingAmount % 25;

        // Other code...
        // Display results
        System.out.println("Your amount " + amount + " consists of");
        System.out.println("      " + numberOfOneDollars + " dollars");
        . . .
    }
}

// See listing 2.10, page 63 for complete code in the
// recommended textbook
```

# Common Errors

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

- Common Error 2: Overflow Errors

```
Example:  byte x = 127; //max byte value  
         x = x + 5;    //causes overflow
```

- Common Error 3: Round-off Errors

```
Example:  double x= 1.0 - 0.9 //expect 0.1  
         // output is 0.09999999999
```

# Common Errors

- Common Error 4: Unintended Integer Division

Example:

```
int Celsius = 40;  
int Fahrenheit = (int) ((5/9) * Celsius + 32);  
//output is always 32!!!
```

- Common Error 5: Redundant Input Objects (such as creating a scanner object for each input variable or each type!)

See recommended textbook, page 66, for examples.

End of Chapter 2