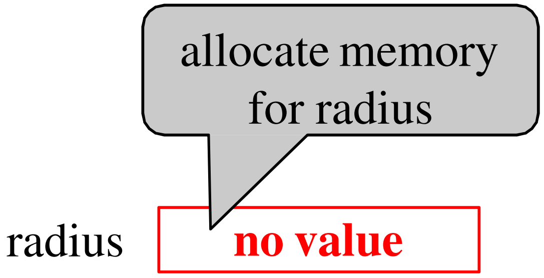
Chapter 2 Elementary Programming

Trace a Program Execution

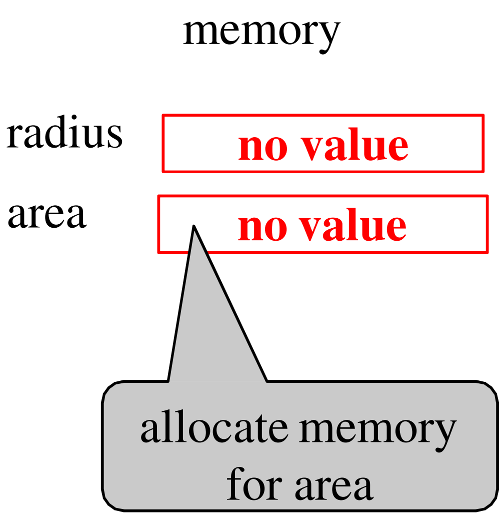
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // 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);
  }
}
```



allocate memory for 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);
  }
}
```



memory

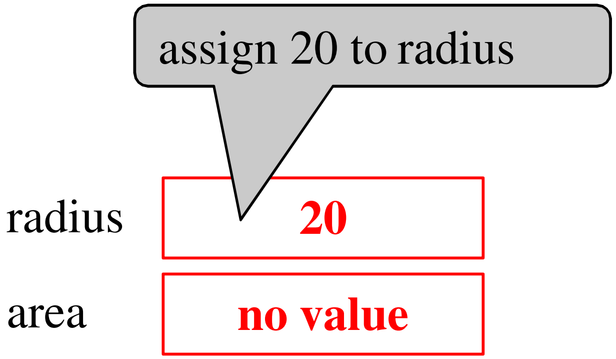
radius **no value**

area **no value**

allocate memory for area

Trace a Program Execution

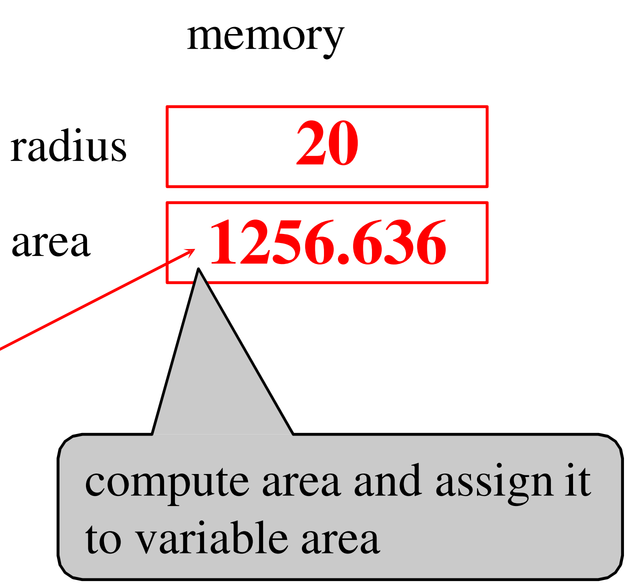
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // 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);
  }
}
```



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

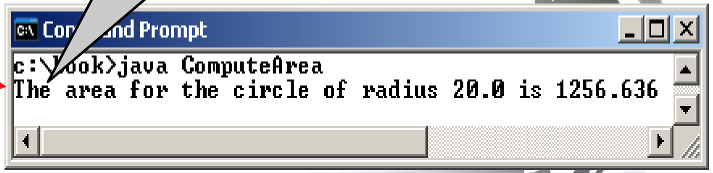
Trace a Program Execution

```
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



```
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 method nextDouble() to obtain to a double value. For example,

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

Reading Input from the Console

```
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.println("Enter three numbers: ");
double number1 = input.nextDouble();
double number2 = input.nextDouble();
double number3 = input.nextDouble();

// Compute average
double average = (number1 + number2 + number3) / 3;

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

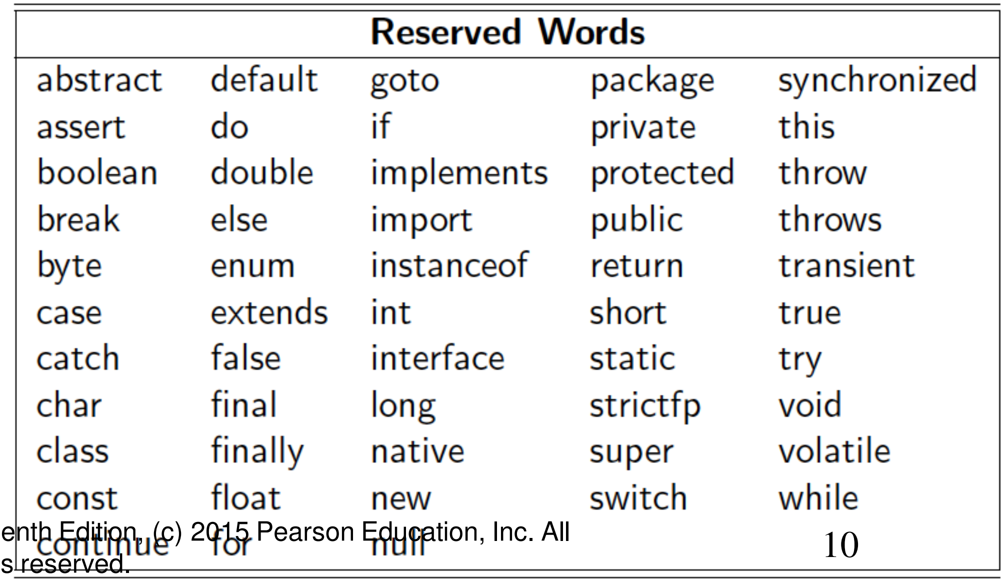
Identifiers

Identifiers are the names that identify the elements such as classes, methods, and variables in a program.

For instance, ComputeAverage, main, input, number1, number2, number3,

Identifiers

- 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.
- An identifier can be of any length.



Reserved Words

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

Declaring Variables

```
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

```
x = 1;               // Assign 1 to x;
radius = 1.0;        // Assign 1.0 to radius;
a = 'A';             // Assign 'A' to a;
```

Declaring and Initializing in One Step (same line)

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

Named Constants

A named constant is an identifier that represents a permanent value

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

```
final double PI = 3.14159;
final int SIZE = 3;
```

Named Constants

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConstant {
public static void main(String[] args) {
final double PI = 3.14159; // Declare a constant

// 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 radius = input.nextDouble();

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

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

Naming Conventions

- Choose meaningful and descriptive names.
- **Variables** and **method** names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.

Naming Conventions, cont.

- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name ComputeArea.

- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant PI and MAX_VALUE

Numerical Data Types

Name	Range	Storage Size
byte	-2^{7} to $2^{7} - 1$ (-128 to 127) integer of the byte type	8-bit signed
short	-2^{15} to $2^{15} - 1$ (-32768 to 32767) integer of the short type	16-bit signed
int	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
long	-2^{63} to $2^{63} - 1$ integer of the long type (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

byte x= 127; // correct

byte x= 128; //incorrect

(Note that the range for a byte value is from –128 to 127.)

Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
double x  = input.nextDouble();
```

Method	Description
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

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 %

5 / 2 yields an integer 2 // int/int=int

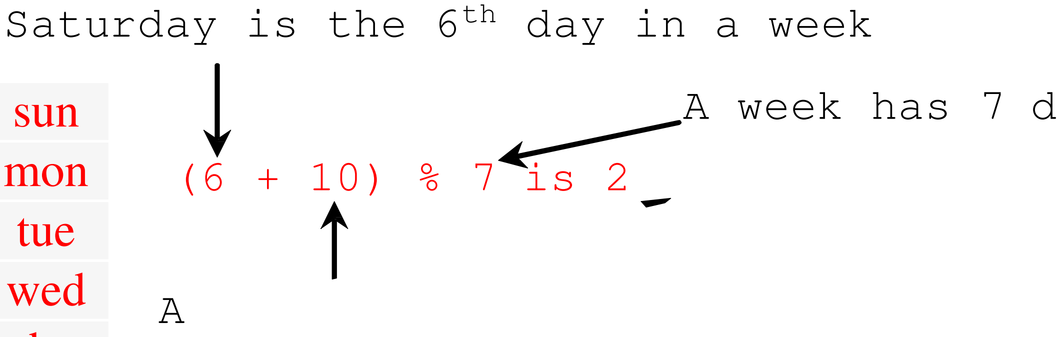
5.0 / 2 yields a double value 2.5 // double/int =double

5 % 2 yields 1 (the remainder of the division)

Remainder Operator

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 and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

0	sun
1	mon
2	tue
3	wed
4	thu
5	fri
6	sat



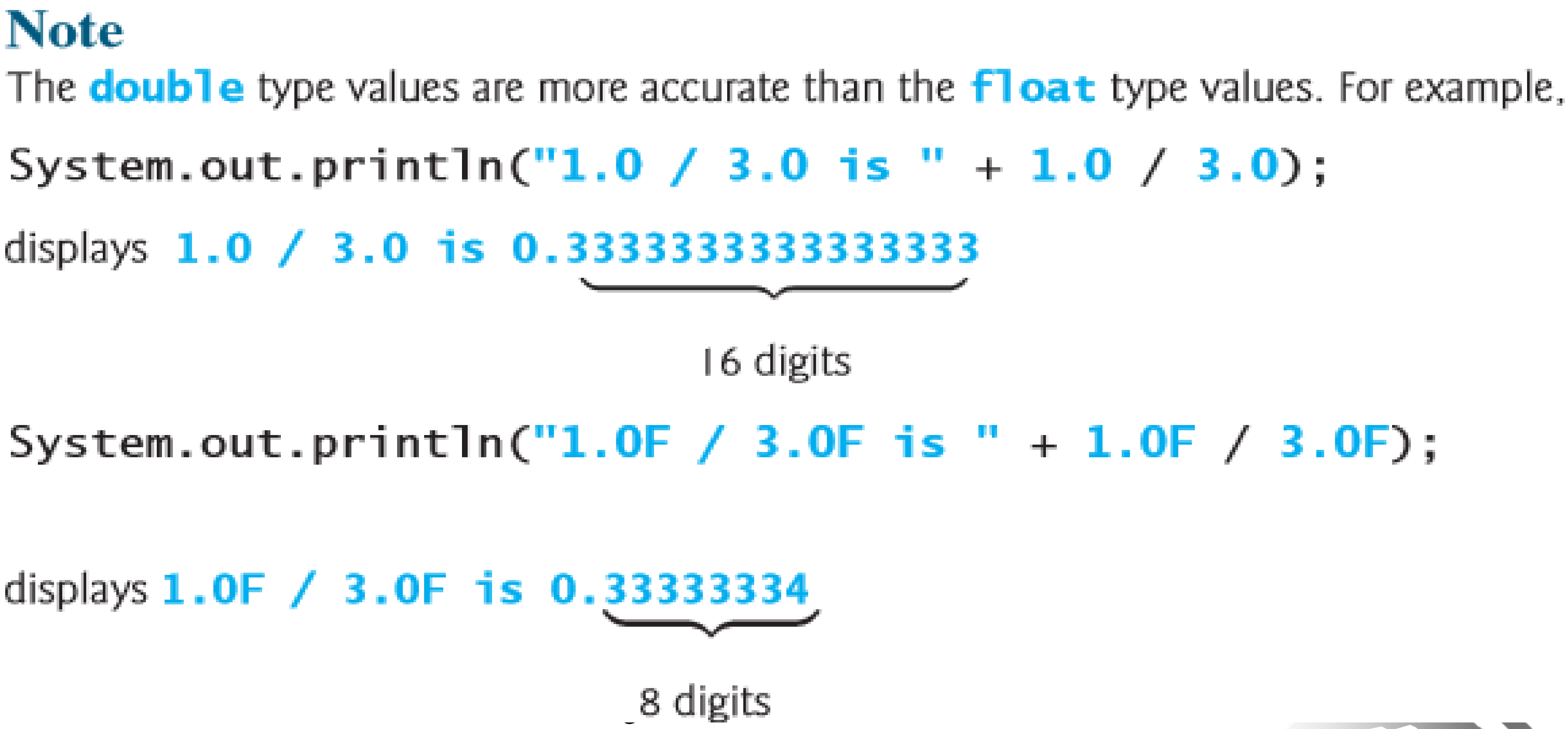
Saturday is the 6th day in a week

A week has 7 d

(6 + 10) % 7 is 2

A

double vs. float



Note

The `double` type values are more accurate than the `float` type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

8 digits

By default, a floating-point literal is treated as a double type value. For example, 5.0 is considered a double value, not a float value. You can make a number a float by appending the letter f or F,

A float value has 7 to 8 number of significant digits and a double value has 15 to 17 number of significant digits.

Arithmetic Expressions

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

In Java, it will be translated to

Arithmetic Expressions

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

In Java, it will be translated to

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

Operator Precedence

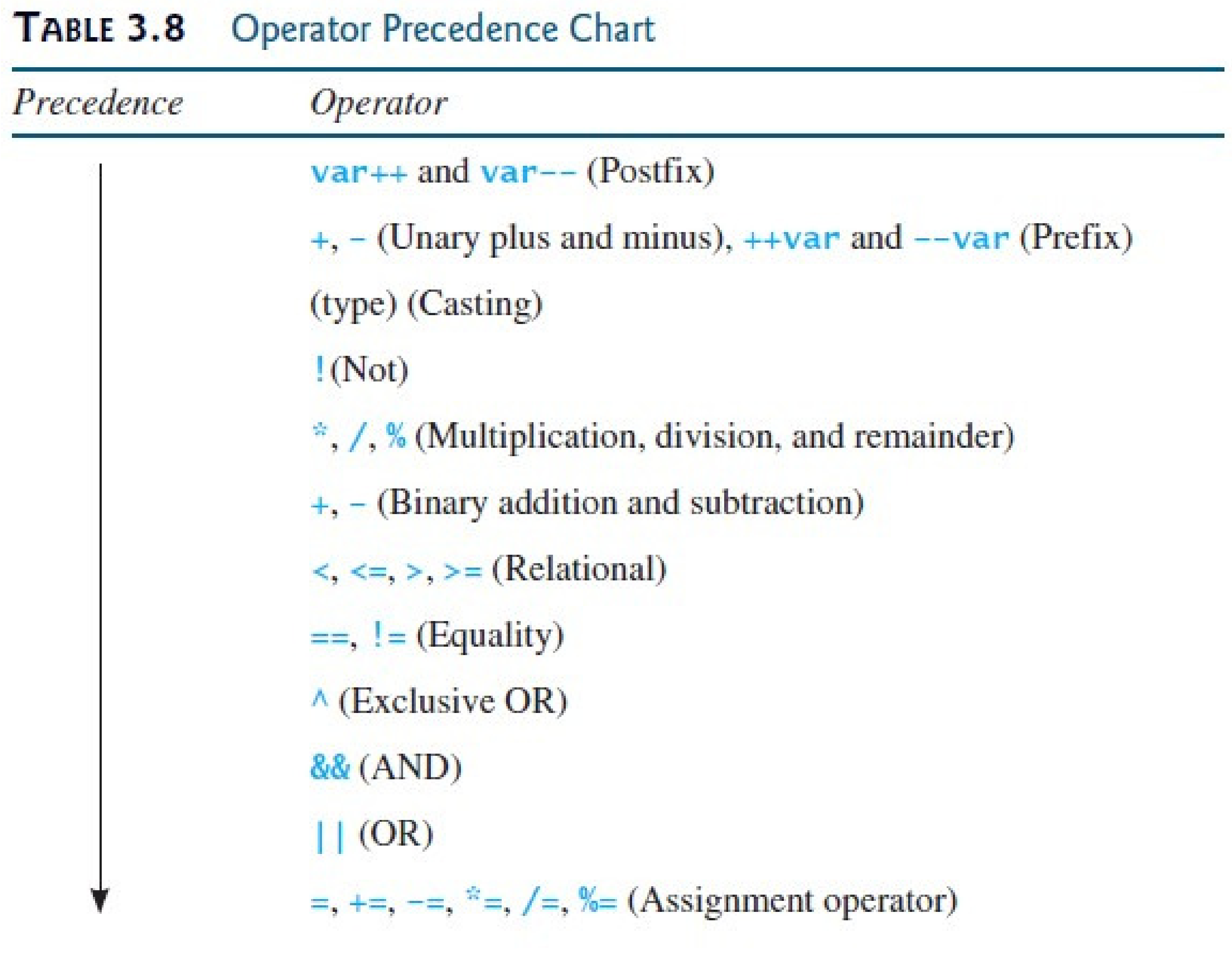


TABLE 3.8 Operator Precedence Chart

Precedence	*Operator*
	var++ and var-- (Postfix)
	+, - (Unary plus and minus), ++var and --var (Prefix)
	(type) (Casting)
	! (Not)
	*, /, % (Multiplication, division, and remainder)
	+, - (Binary addition and subtraction)
	<, <=, >, >= (Relational)
	==, != (Equality)
	^ (Exclusive OR)
	&& (AND)
	|| (OR)
	=, +=, -=, *=, /=, %= (Assignment operator)

Operator Precedence and Associativity

- The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.)

- When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

- If operators with the same precedence are next to each other, their associativity determines the order of evaluation. All binary operators except assignment operators are left-associative.

Operator Associativity

- When two operators with the same precedence are evaluated, the *associativity* of the operators determines the order of evaluation.

- All binary operators **except assignment** operators are ***left-associative***.

  a – b + c – d is equivalent to

- Assignment operators are ***right-associative***. Therefore, the expression

  a = b += c = 5 is equivalent to

Operator Associativity

- When two operators with the same precedence are evaluated, the *associativity* of the operators determines the order of evaluation.
- All binary operators **except assignment** operators are ***left-associative***.

  a – b + c – d is equivalent to **((a – b) + c) – d**
- Assignment operators are ***right-associative***. Therefore, the expression

  a = b += c = 5 is equivalent to **a = (b += (c = 5))**

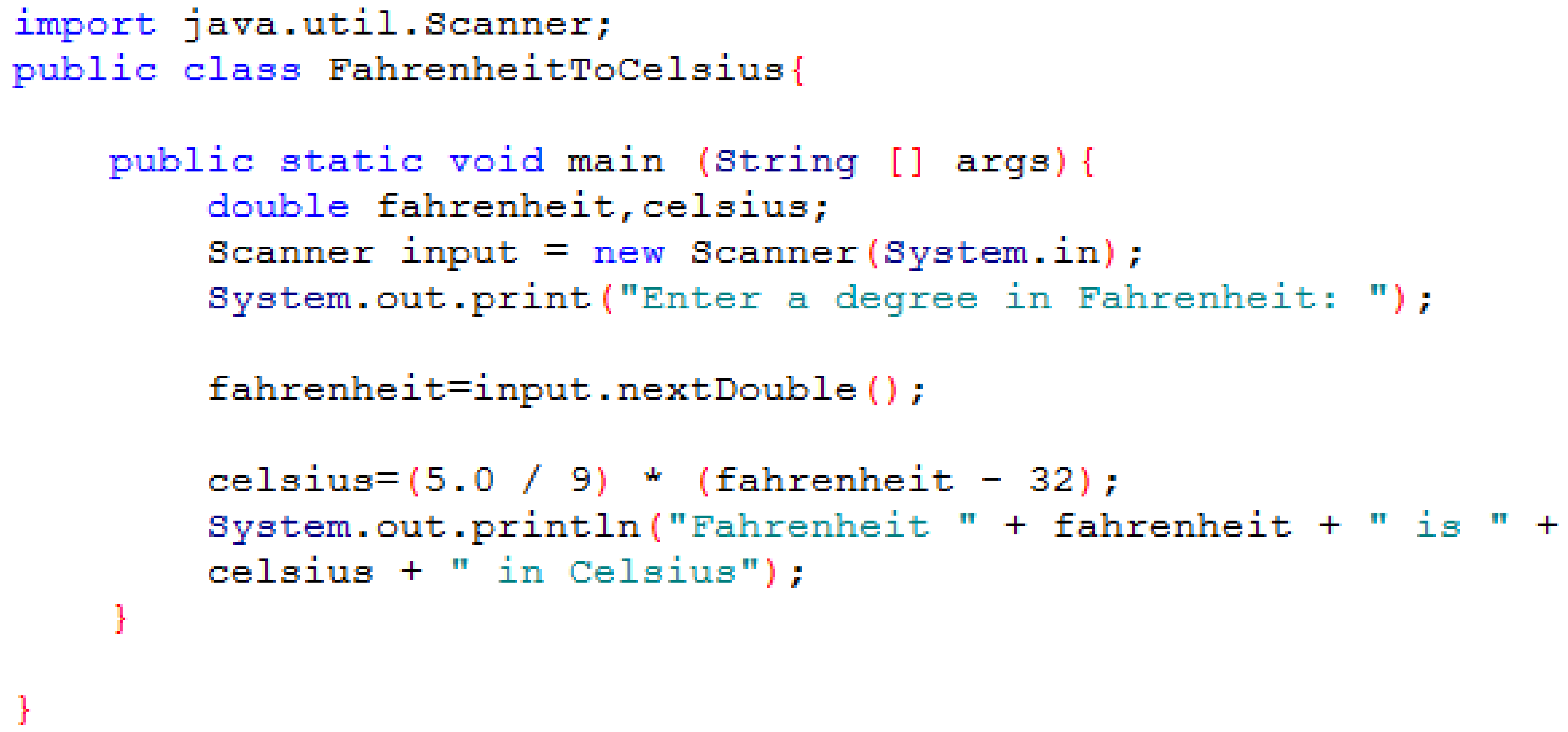
Problem: Converting Temperatures

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

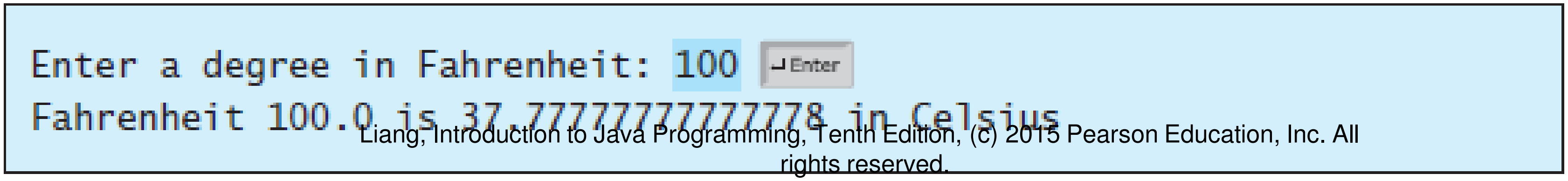
$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

Note: you have to write
celsius = (5.0 / 9) * (fahrenheit – 32)

Problem: Converting Temperatures



```
import java.util.Scanner;
public class FahrenheitToCelsius{

    public static void main (String [] args){
        double fahrenheit,celsius;
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a degree in Fahrenheit: ");

        fahrenheit=input.nextDouble();

        celsius=(5.0 / 9) * (fahrenheit - 32);
        System.out.println("Fahrenheit " + fahrenheit + " is " +
        celsius + " in Celsius");
    }

}
```



```
Enter a degree in Fahrenheit: 100 ↵Enter
Fahrenheit 100.0 is 37.77777777777778 in Celsius
```

Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
+=	Addition assignment	i += 8	i = i + 8
-=	Subtraction assignment	i -= 8	i = i - 8
*=	Multiplication assignment	i *= 8	i = i * 8
/=	Division assignment	i /= 8	i = i / 8
%=	Remainder assignment	i %= 8	i = i % 8

Increment and Decrement Operators

Operator	*Name*	*Description*	*Example (assume i = 1)*

`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;`
			`// j is 2, i is 2`

`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;`
			`// j is 1, i is 2`

`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;`
			`// j is 0, i is 0`

`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;`
			`// j is 1, i is 0`

Increment and Decrement Operators, cont.

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

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

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

Numeric Type Conversion

Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Conversion Rules

When performing a binary 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.

Conversion Rules: Example

If an integer and a floating-point number are involved in a binary operation, Java automatically converts the integer to a floating-point value.

Ex: **3 * 4.5** is same as **3.0 * 4.5**

Type Casting

Implicit casting

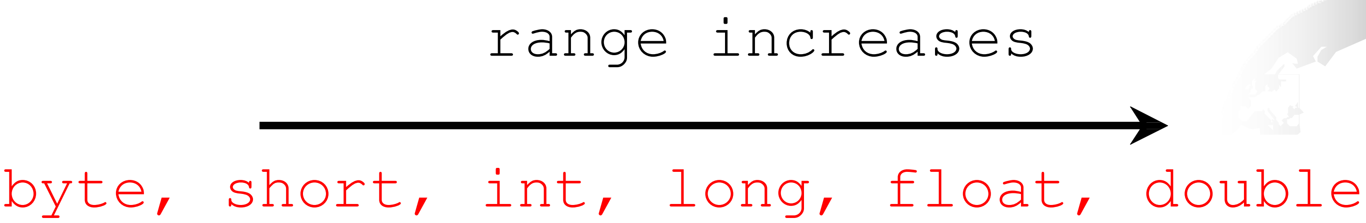
`double d = 3;` (type widening)

Explicit casting

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

`int i = (int)3.9;` (Fraction part is truncated)

What is wrong? int x = 5 / 2.0;



Type Casting: Examples

- System.out.println((**int**)**1.7**);

- System.out.println((double)1 / 2);

- System.out.println(1 / 2);

Type Casting: Examples

- System.out.println((**int**)**1.7**); // 1

displays **1**. When a **double** value is cast into an **int** value, the fractional part is truncated.

- System.out.println((double)1 / 2);

- System.out.println(1 / 2);

Type Casting: Examples

- System.out.println((**int**)**1.7**); // 1

displays **1**. When a **double** value is cast into an **int** value, the fractional part is truncated.

- System.out.println((double)1 / 2); // 0.5

displays 0.5, because 1 is cast to 1.0 first, then 1.0 is divided by 2.

- System.out.println(1 / 2);

Type Casting: Examples

- System.out.println((**int**)**1.7**); // 1

displays **1**. When a **double** value is cast into an **int** value, the fractional part is truncated.

- System.out.println((double)1 / 2); // 0.5

displays 0.5, because 1 is cast to 1.0 first, then 1.0 is divided by 2.

- System.out.println(1 / 2); // 0

displays 0, because 1 and 2 are both integers and the resulting value should also be an integer.

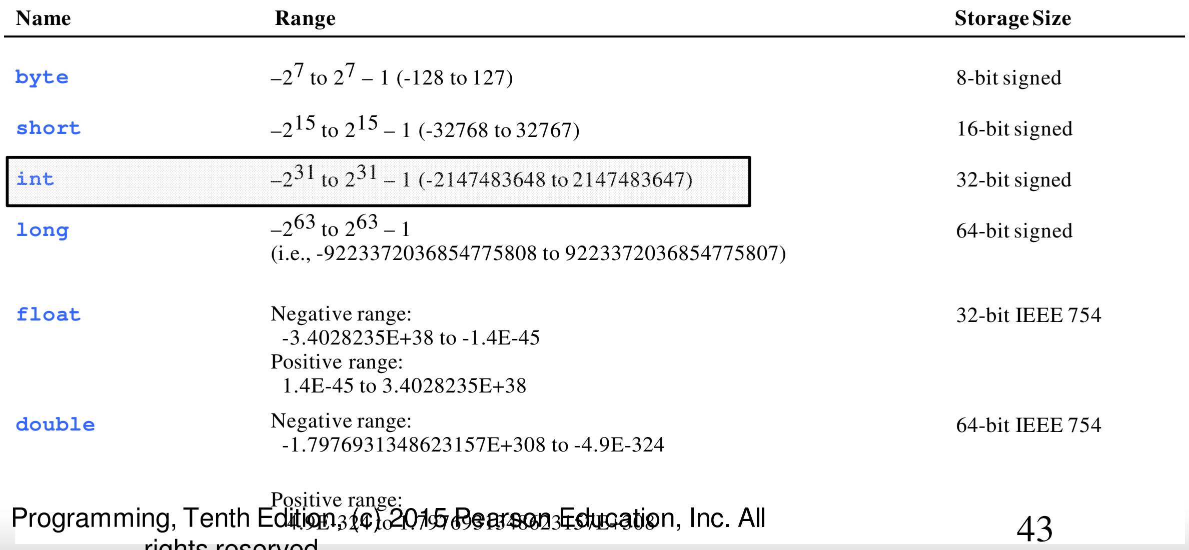
Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

```
double interestRate = 0.05;
double interest = interestrate * 45;
```

This code is wrong, because **interestRate** is assigned a value **0.05**; but **interestrate** has not been declared and initialized.

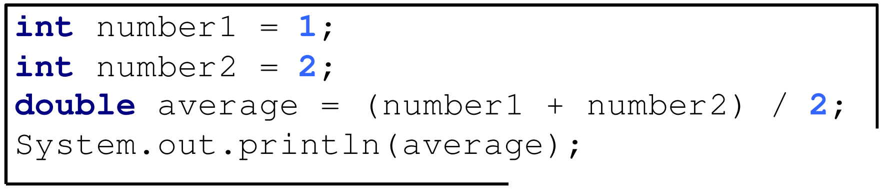
Common Error 2: Integer Overflow

```
int value = 2147483647 + 1;
// value will actually be -2147483648
```



Name	Range	Storage Size
`byte`	-2^{7} to $2^{7} - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Common Error 3: Unintended Integer Division



```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

```
int number1 = 1;
int number2 = 2;
double
```

the code in (a) displays that average is 1 **and the code in (b) displays that average is** 1.5

Common Error 4: Redundant Input Objects

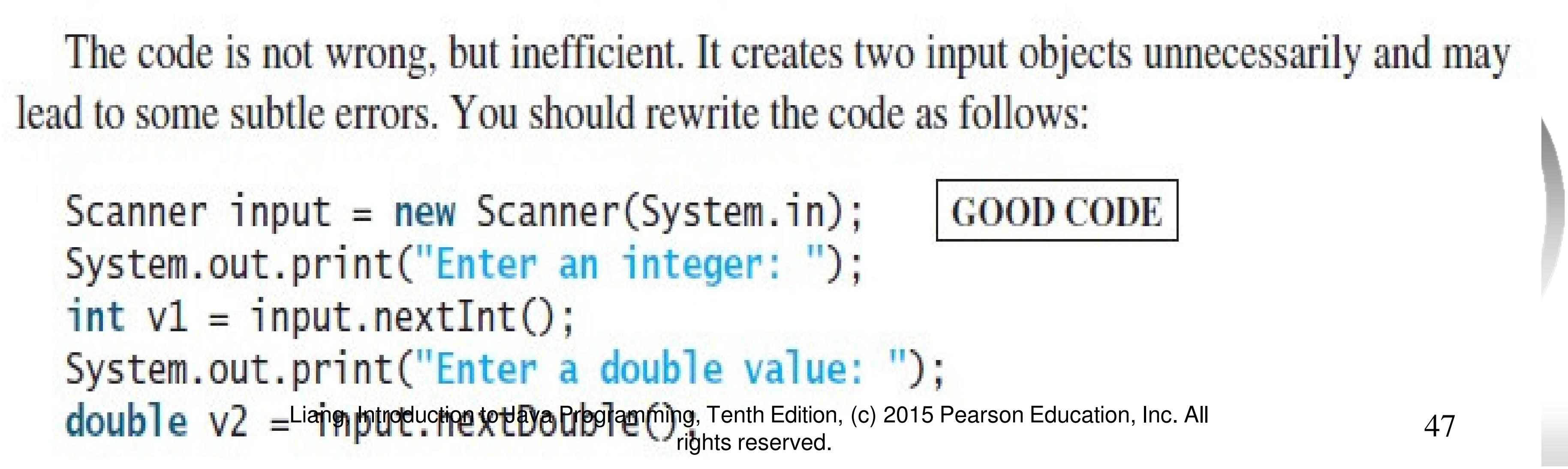
```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```

Common Error 4: Redundant Input Objects

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```

Common Error 4: Redundant Input Objects

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```



The code is not wrong, but inefficient. It creates two input objects unnecessarily and may lead to some subtle errors. You should rewrite the code as follows:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
System.out.print("Enter a double value: ");
double v2 = input.nextDouble();
```

GOOD CODE

Character Data Type

char letter = 'A'; (ASCII)

char numChar = '4'; (ASCII)

char letter = '\u0041'; (Unicode) **// Character A's Unicode is 0041**

char numChar = '\u0034'; (Unicode)

NOTE: The increment and decrement operators can also be used on **char** variables to get the next or preceding Unicode character. For example, the following statements display character **b**.

```
char ch = 'a';
System.out.println(++ch);
```

The String Type

- The char type only represents **one** character. To represent a string of characters, use the data type called **String**. For example:

String message = "Welcome to Java!";

- String is actually a predefined class in the Java library.
- The String type is not a primitive type. It is known as a ***reference*** *type*.

String Concatenation

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

Console Input

- You can use the **Scanner** class for console input.
- Java uses **System.in** to refer to the standard input device (i.e. Keyboard).

```
import java.util.Scanner;
public class Test{
   public static void main(String[] s){
      Scanner input = new
      Scanner(System.in);
      System.out.println("Enter    X : ");
       int x = input.nextInt();
       System.out.println("You entered: "+ x);
   }
}
```