Lecture1 - Recap

A Java Program

```
//  comments about the class
public class MyProgram
{

   //  comments about the method
   public static void main(String[] args)
   {
                                               method header
                  method body

   }


}
```

Variables

- A *variable* is a name for a location in memory

- A variable must be *declared* by specifying its name and the type of information that it will hold

data type **variable name**

```
int total;

int count, temp, result;
```

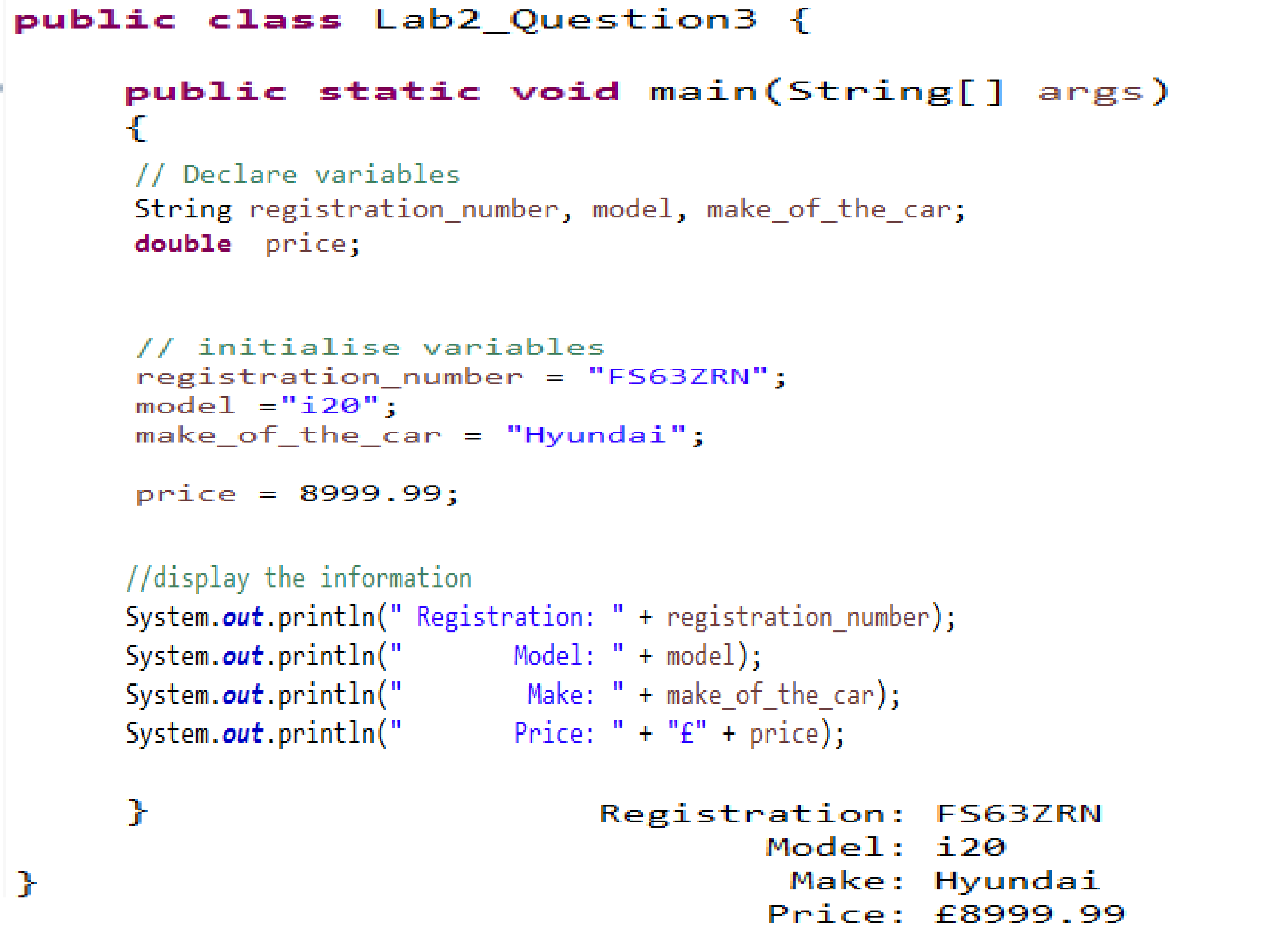
Multiple variables can be created in one declaration

Example

Write a java program to store the following information about a car:

- Registration number
- Model
- Price
- Make of the car.

Display the information on the console.



```java
public class Lab2_Question3 {

    public static void main(String[] args)
    {
     // Declare variables
     String registration_number, model, make_of_the_car;
     double  price;


     // initialise variables
     registration_number = "FS63ZRN";
     model ="i20";
     make_of_the_car = "Hyundai";

     price = 8999.99;

    //display the information
    System.out.println(" Registration: " + registration_number);
    System.out.println("        Model: " + model);
    System.out.println("         Make: " + make_of_the_car);
    System.out.println("        Price: " + "£" + price);

    }

}
```

```
 Registration: FS63ZRN
        Model: i20
         Make: Hyundai
        Price: £8999.99
```

Lecture 2

Learning Objectives

- Assignment statement and expressions
- Data conversions
- The `Scanner` class for interactive programs

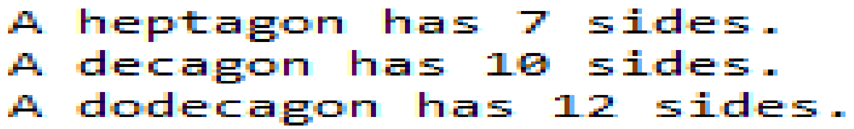
Assignment

- 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 the 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

```java
//********************************************************************
//  Geometry.java       Java Foundations
//
//  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.");
   }
}
```

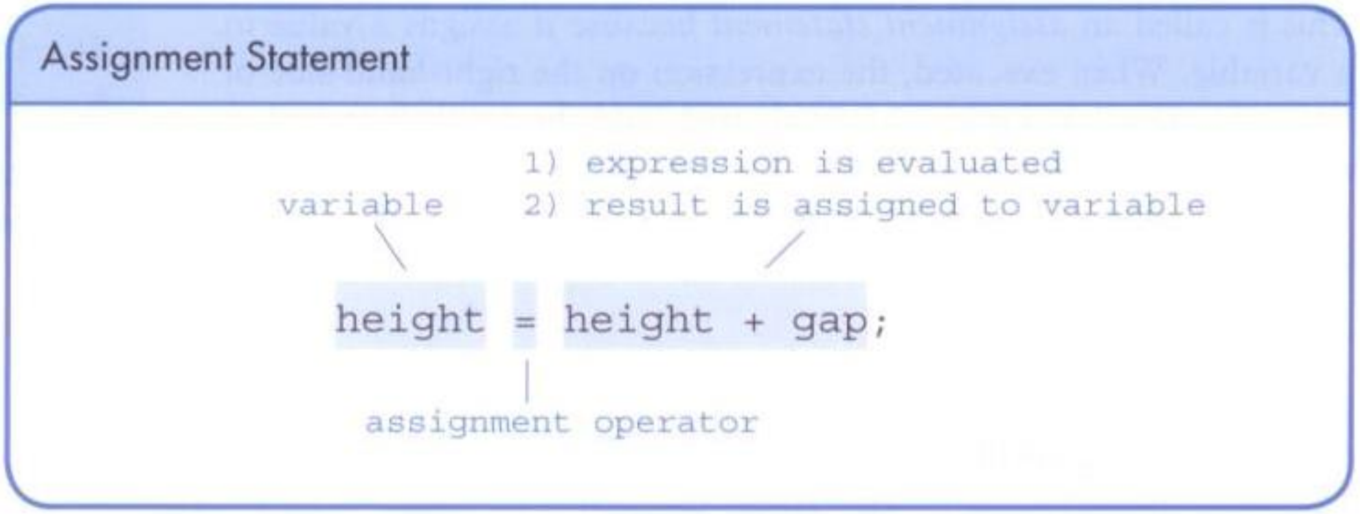


```
A heptagon has 7 sides.
A decagon has 10 sides.
A dodecagon has 12 sides.
```

Assignment

- The right-hand side could be an expression
- The expression is completely evaluated and the result is stored in the variable

```
 int height = 30;
int gap = 20;
```



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 used by an arithmetic operator are floating point, then the result is a floating point

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

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

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

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

Operator Precedence

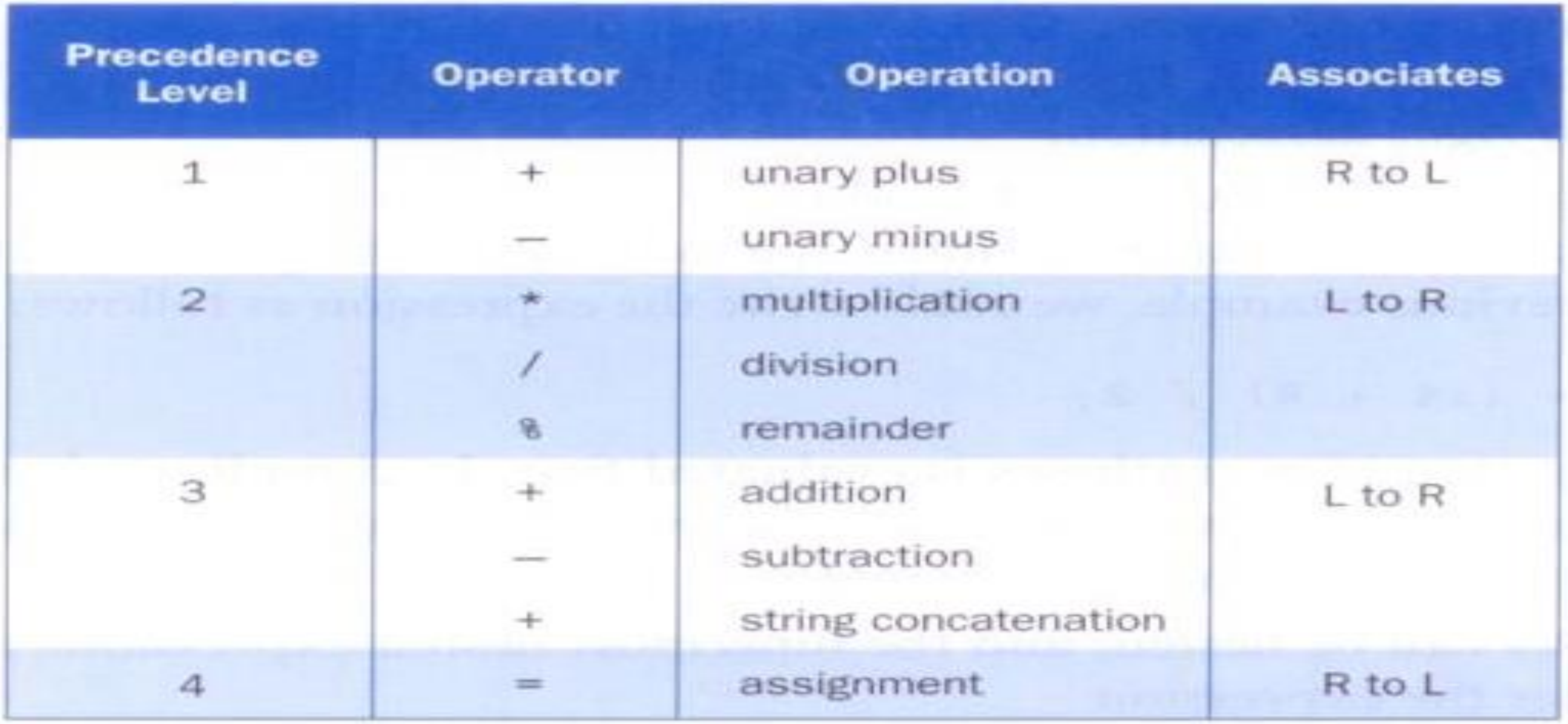
- Operators can be combined into complex expressions

```
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 prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

Operator Precedence

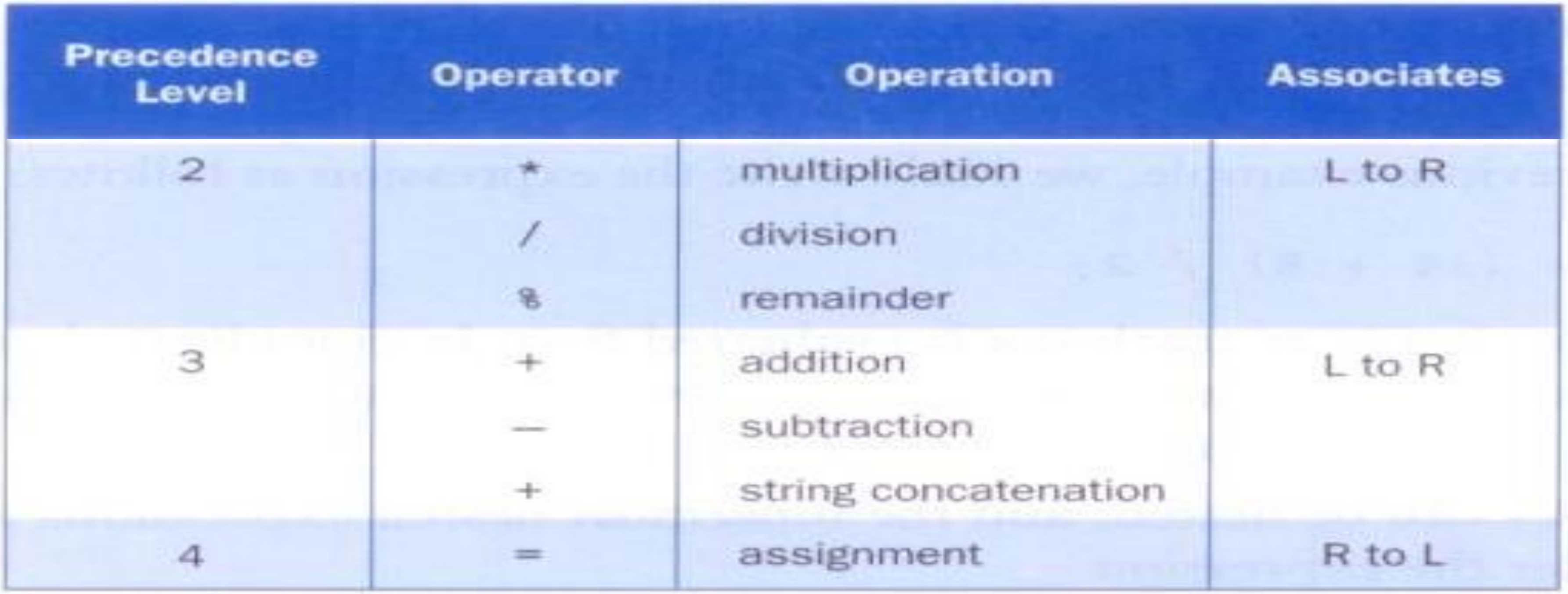
- Precedence among some Java operators:



Precedence Level	Operator	Operation	Associates
1	+	unary plus	R to L
	–	unary minus	
2	*	multiplication	L to R
	/	division	
	%	remainder	
3	+	addition	L to R
	–	subtraction	
	+	string concatenation	
4	=	assignment	R to L

Operator Precedence

- Precedence among some Java operators:



Precedence Level	Operator	Operation	Associates
2	*	multiplication	L to R
	/	division	
	%	remainder	
3	+	addition	L to R
	–	subtraction	
	+	string concatenation	
4	=	assignment	R to L

Operator Precedence

- What is the order of evaluation in the following expressions?

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

            a / (b + c) - d % e

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

Operator Precedence

- What is the order of evaluation in the following expressions?

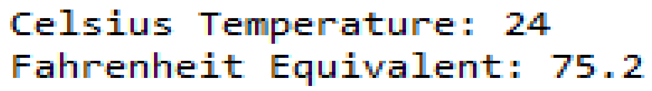
```
a + b + c + d + e
  1   2   3   4
```

```
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
```

```
//********************************************************************
//  TempConverter.java       Java Foundations
//
//  Demonstrates the use of primitive data types and arithmetic
//  expressions.
//********************************************************************

public class TempConverter
{
   //-----------------------------------------------------------------
   //  Computes the Fahrenheit equivalent of a specific Celsius
   //  value using the formula F = (9/5)C + 32.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int BASE = 32;
      final double CONVERSION_FACTOR = 9.0 / 5.0;

      double fahrenheitTemp;
      int celsiusTemp = 24;  // value to convert

      fahrenheitTemp = celsiusTemp * CONVERSION_FACTOR + BASE;

      System.out.println ("Celsius Temperature: " + celsiusTemp);
      System.out.println ("Fahrenheit Equivalent: " + fahrenheitTemp);
   }
}
```



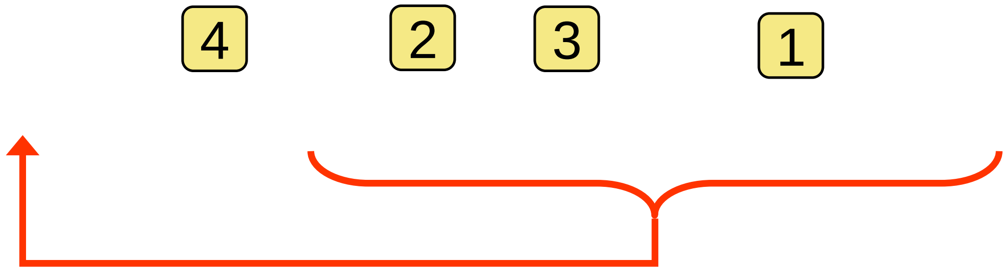
```
Celsius Temperature: 24
Fahrenheit Equivalent: 75.2
```

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

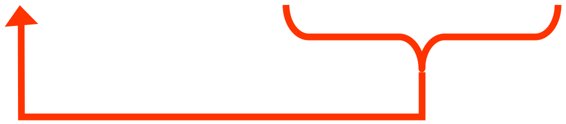
Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

```
int count;
int count = 99;
```

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 Operators

- The increment and decrement operators use only one operand
- The *increment operator* (++) adds one to its operand
- The *decrement operator* (−−) subtracts one from its operand
- The statement

  ```
  count++;
  ```

  is functionally equivalent to

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

Increment and Decrement Operators

- 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);
```

Data Conversions

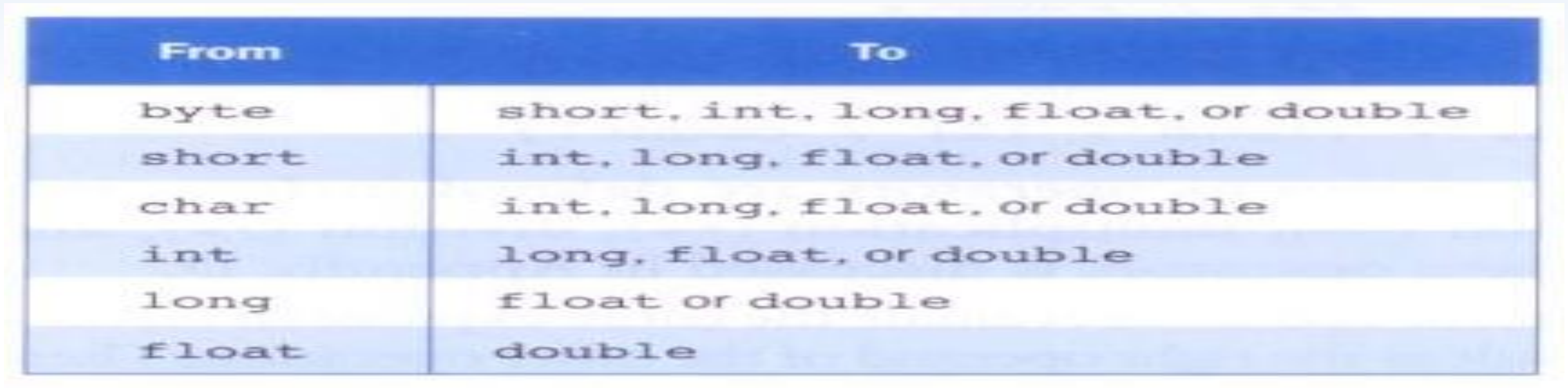
- Sometimes it is convenient to convert data from one type to another
- For example, in a particular situation we may want to treat an integer as a floating point value
- These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation

Data Conversions

- 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 a `short` to an `int`)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one.
- In Java, data conversions can occur in three ways
  - assignment conversion
  - promotion
  - casting

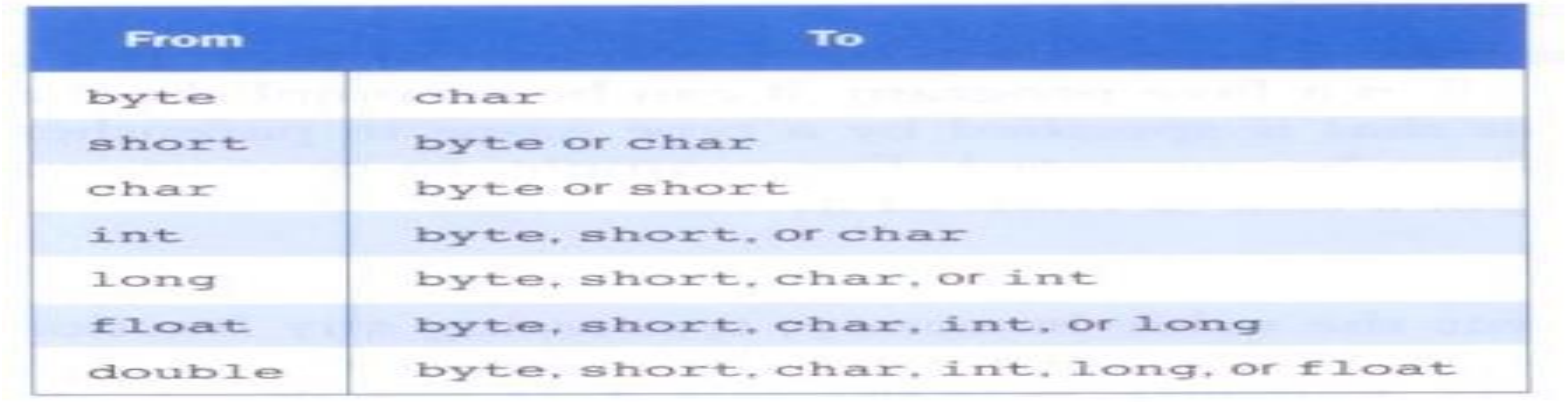
Data Conversions

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

```
int dollars = 100;
float money;
```

```
money = dollars;
```

- **Only widening conversions can happen via assignment**
- Note that the value or type of `dollars` did not change

Promotion

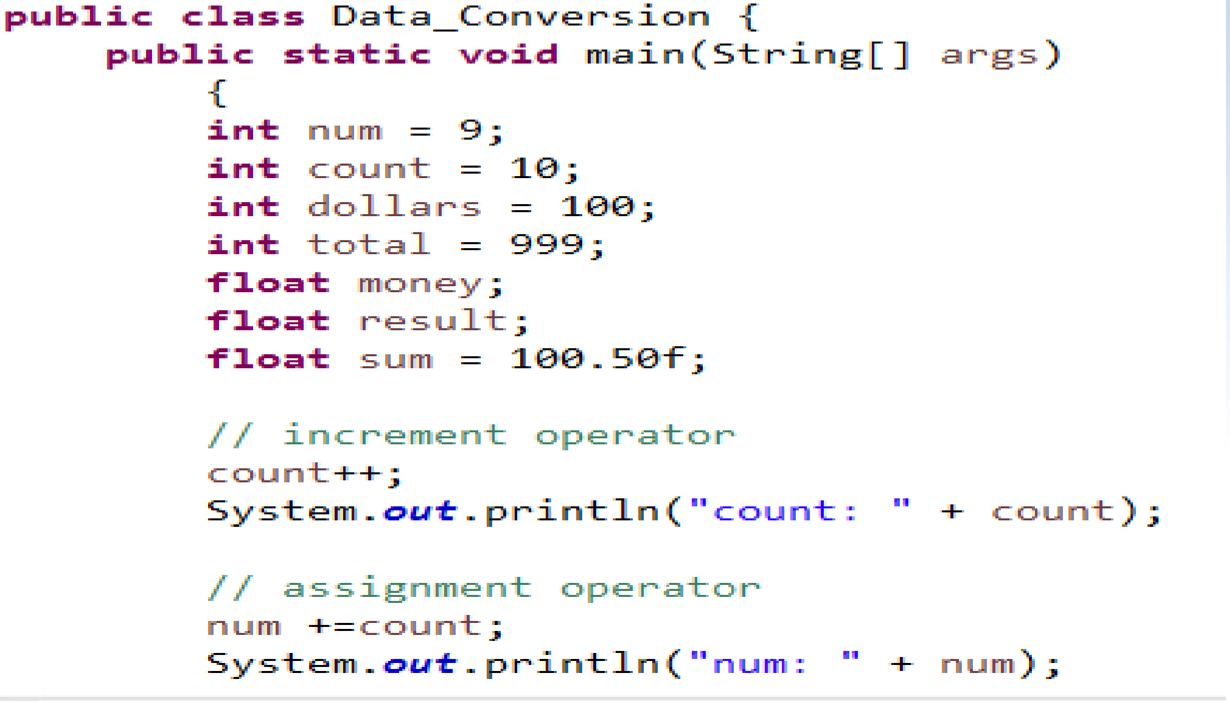
- *Promotion* happens automatically when operators in expressions convert their operands
- For example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a floating point value to perform the following calculation

```
result = sum / count;
```

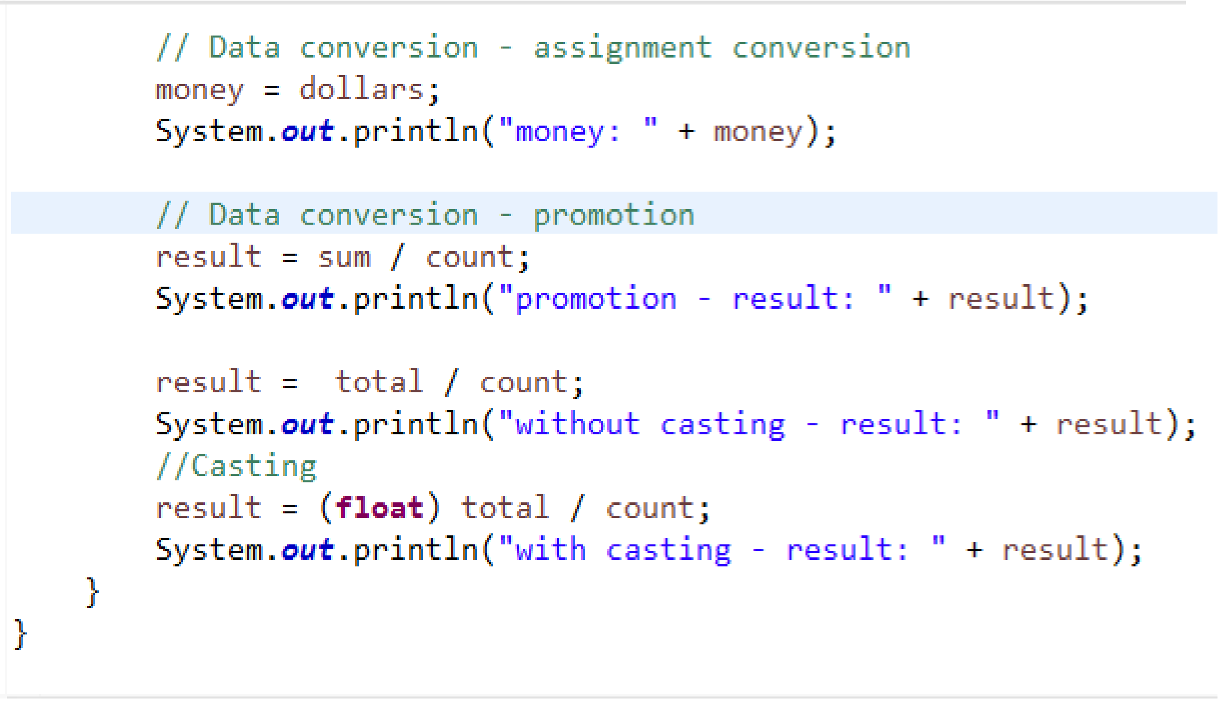
Casting

- *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
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`

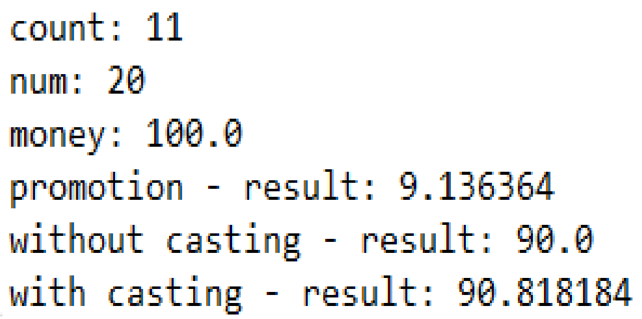
```
result = (float) total / count;
```



```java
public class Data_Conversion {
    public static void main(String[] args)
        {
        int num = 9;
        int count = 10;
        int dollars = 100;
        int total = 999;
        float money;
        float result;
        float sum = 100.50f;

        // increment operator
        count++;
        System.out.println("count:  " + count);

        // assignment operator
        num +=count;
        System.out.println("num:  " + num);
```



```java
        // Data conversion - assignment conversion
        money = dollars;
        System.out.println("money:  " + money);

        // Data conversion - promotion
        result = sum / count;
        System.out.println("promotion - result:  " + result);

        result =  total / count;
        System.out.println("without casting - result:  " + result);
        //Casting
        result = (float) total / count;
        System.out.println("with casting - result:  " + result);
    }
}
```



```
count: 11
num: 20
money: 100.0
promotion - result: 9.136364
without casting - result: 90.0
with casting - result: 90.818184
```

Data Conversion using Wrapper Classes Methods

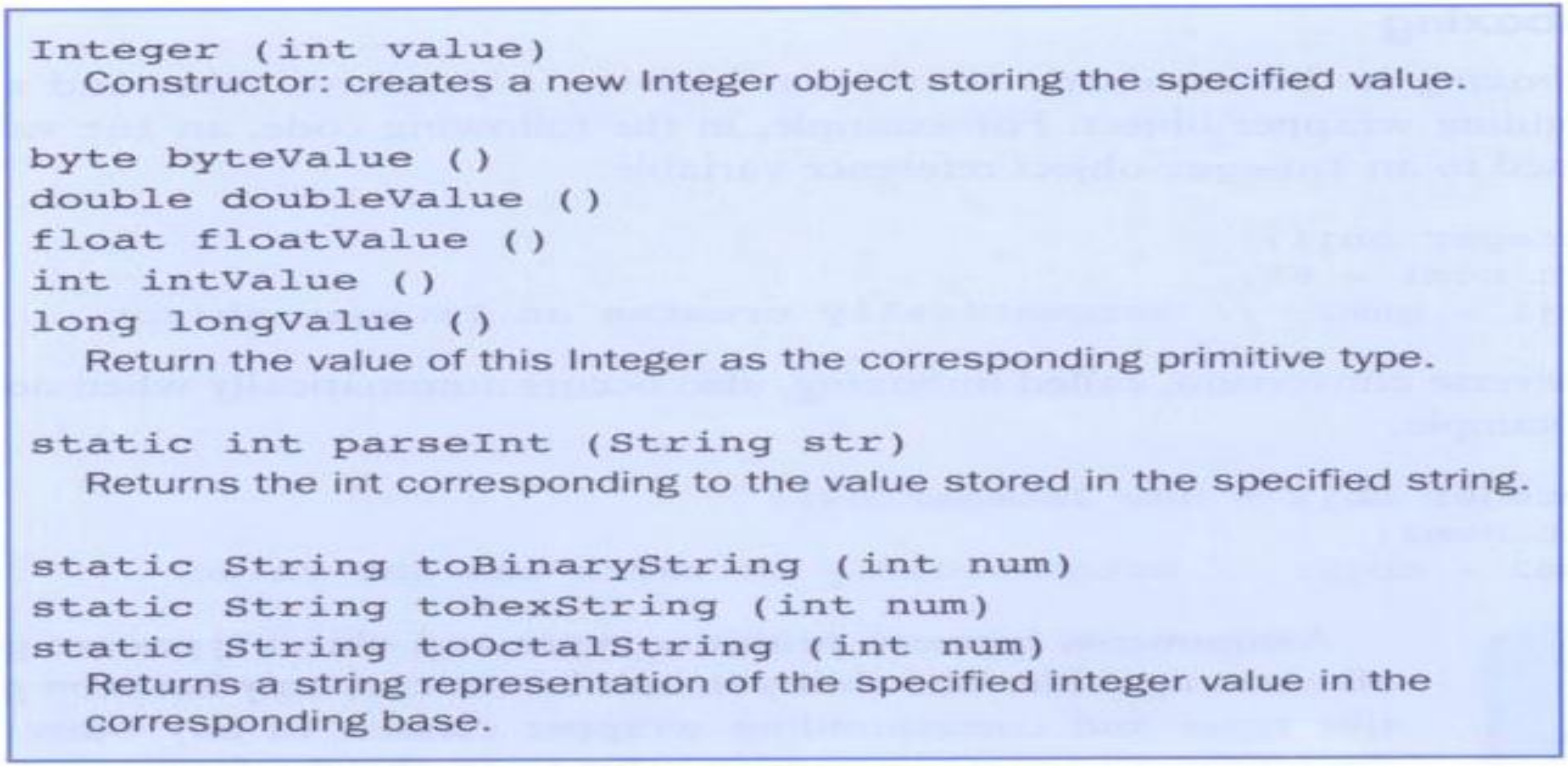
- Wrapper classes also contain static methods that help manage the associated type
- For example, the `Integer` class contains a method to convert an integer stored in a `String` to an `int` value:

```
num = Integer.parseInt(str);
```

- The wrapper classes often contain useful constants as well
- For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values

Wrapper Classes

- Some methods of the `Integer` class:



```
Integer (int value)
   Constructor: creates a new Integer object storing the specified value.

byte byteValue ()
double doubleValue ()
float floatValue ()
int intValue ()
long longValue ()
   Return the value of this Integer as the corresponding primitive type.

static int parseInt (String str)
   Returns the int corresponding to the value stored in the specified string.

static String toBinaryString (int num)
static String tohexString (int num)
static String toOctalString (int num)
   Returns a string representation of the specified integer value in the
   corresponding base.
```

Data Conversion Examples using Wrapper Classes Methods

Integer.parseInt(string)

```
String numberAsString = “2345";
int number = Integer.parseInt(numberAsString);
```

Long.parseLong(string)

```
String  numberAsString = “99999999876";
long  number = Long.parseLong(numberAsString);
```

Float.parseFloat(string)

```
String  numberAsString = “55.25";
float  number = Float.parseFloat(numberAsString);
```

Double.parseDouble(string)

```
String  numberAsString = "199.9995";
double number = Double.parseDouble(numberAsString);
```

The Scanner Class

- The `Scanner` class provides convenient methods for reading input values of various types
- The `Scanner` class is part of the **java.util** class library, and **must be imported into a program to be used**

```
 import java.util.Scanner;
```

- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

Reading Input

- The following line creates a `Scanner` object that reads from the keyboard

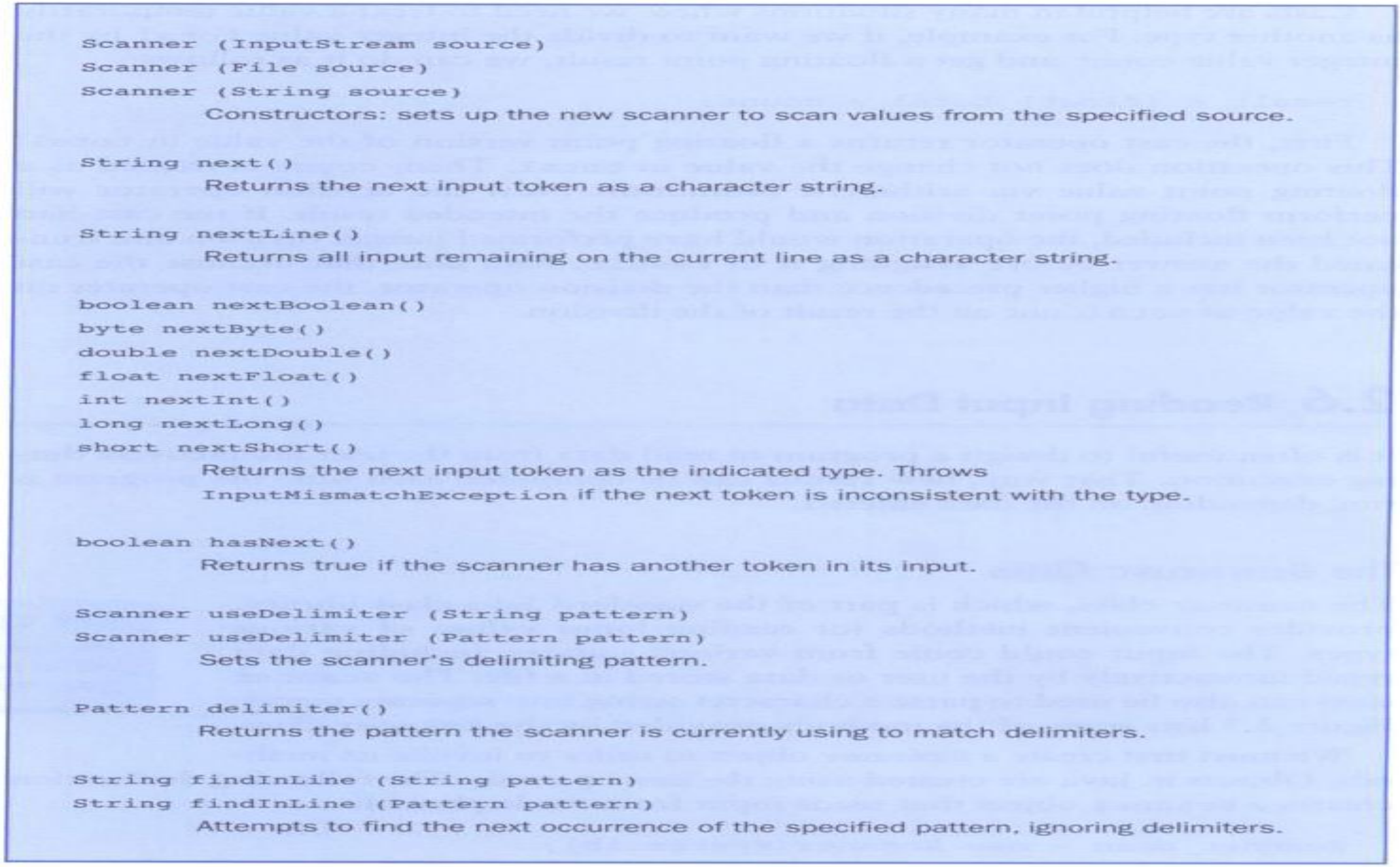
```
Scanner scan = new Scanner(System.in);
```

- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as

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

- The `nextLine` method reads all of the input until the end of the line is found

Some methods of the `Scanner` class



`Scanner (InputStream source)`
`Scanner (File source)`
`Scanner (String source)`
Constructors: sets up the new scanner to scan values from the specified source.

`String next()`
Returns the next input token as a character string.

`String nextLine()`
Returns all input remaining on the current line as a character string.

`boolean nextBoolean()`
`byte nextByte()`
`double nextDouble()`
`float nextFloat()`
`int nextInt()`
`long nextLong()`
`short nextShort()`
Returns the next input token as the indicated type. Throws `InputMismatchException` if the next token is inconsistent with the type.

`boolean hasNext()`
Returns true if the scanner has another token in its input.

`Scanner useDelimiter (String pattern)`
`Scanner useDelimiter (Pattern pattern)`
Sets the scanner's delimiting pattern.

`Pattern delimiter()`
Returns the pattern the scanner is currently using to match delimiters.

`String findInLine (String pattern)`
`String findInLine (Pattern pattern)`
Attempts to find the next occurrence of the specified pattern, ignoring delimiters.

```java
//********************************************************************
//  Echo.java       Java Foundations
//  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 + "\"");
   }
}
```

```
Enter a line of text:
Demonstrates the use of the Scanner
You entered: "Demonstrates the use of the Scanner"
```

Input Tokens

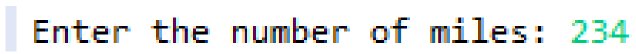
- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input

- White space includes space characters, tabs, new line characters

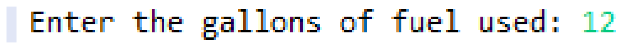
- The `next` method of the `Scanner` class reads the next input token and returns it as a string

- Methods such as `nextInt` and `nextDouble` read data of particular types

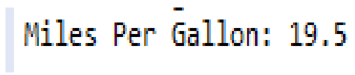
```java
//********************************************************************
//  GasMileage.java       Java Foundations
//  Demonstrates the use of the Scanner class to read numeric data.
//********************************************************************
import java.util.Scanner;

public class GasMileage
{
   //-----------------------------------------------------------------
   //  Calculates fuel efficiency based on values entered by the
   //  user.
   //-----------------------------------------------------------------
   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);
   }
}
```



```
Enter the number of miles: 234
```

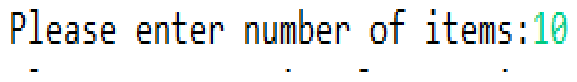


```
Enter the gallons of fuel used: 12
```

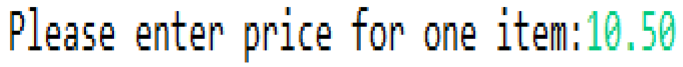


```
Miles Per Gallon: 19.5
```

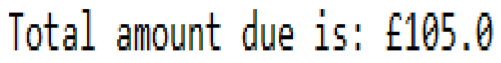
```java
// WrapperClassExample.java
// Demonstrates converting a String to an integer and a float using wrapper classes methods.
import java.util.Scanner;

public class WrapperClassExample {

public static void main(String[] args) {

     float price, totalprice;
     int Noitems;
     String input ="";
     Scanner keyboard = new Scanner (System.in);

     System.out.print("Please enter number of items:");
     input = keyboard.nextLine();
     Noitems = Integer.parseInt(input);

     System.out.print("Please enter price for one item:");
     input = keyboard.nextLine();
     price = Float.parseFloat(input);
     totalprice = price *Noitems;
     System.out.print("Total amount due is: £" + totalprice);
     }
}
```



```
Please enter number of items:10
```



```
Please enter price for one item:10.50
```



```
Total amount due is: £105.0
```