Building Java Programs

Chapter 3: Introduction to Parameters and Objects

Chapter outline

- parameters
  - passing parameters to static methods
  - writing methods that accept parameters
- methods that return values
  - calling methods that return values (e.g., the Math class)
  - writing methods that return values
- using objects
  - String objects
  - Point objects
  - console input with Scanner objects

Reminder: class constants

In the last chapter, we used class constants to fix "magic number" redundancy problems:

```java
public static final int FLOOR_WIDTH = 3;
public static void main(String args[]) {
  drawFigure();
  drawFloor();
}
```

A redundant solution

```java
public class Draw {
  public static void main(String args[]) {  
    drawFloor();
    drawSquare();
    drawSquare();
    drawSquare();
    system.out.println("Draw");
  }
  public static void drawFloor() {
    for (int x = 0; x < FLOOR_WIDTH; x++) {
      system.out.print("");
    }
  }
  public static void drawSquare() {
    for (int x = 0; x < FLOOR_WIDTH; x++) {
      for (int y = 0; y < FLOOR_WIDTH; y++) {
        system.out.print("");
      }
      system.out.println("");
    }
  }
}
```

Another repetitive figure

- Now consider the task of drawing the following figures:

```
********
******
********
```

- The lines and figures are similar, but not exactly the same.
**Parameterization**

- **parameterized method**: One that is given extra information (e.g., number of stars to draw) when it is called.

- **parameter**: A value passed to a method by its caller.

- Writing parameterized methods requires 2 steps:
  - Define the method to accept the parameter
  - Call the method and pass the parameter value(s) desired

```
public static void draw(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print('*');
    }
}
```

**Writing parameterized methods**

- **parameterized method declaration syntax**:

```java
public static void <name> <expression> {
    statement(s);
}
```

- **Example**:

```java
public static void printSpace(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(' ');
    }
}
```

- Whenever printSpace is called, the caller must specify how many spaces to print.

**Calling parameterized methods**

- **passing a parameter**: Calling a parameterized method and specifying a value for its parameter(s).

- Parameterized method call syntax:

```java
<name> <expression> ;
```

- **Example**:

```java
System.out.println("* ");
printSpace(7);   // unordered
```

**Value semantics**

- **value semantics**: When primitive variables (int, double) are passed as parameters, their values are copied.
- Modifying the parameter inside the method will not affect the variable passed in.

```java
public static void main(String[] args) {
    int x = 3;
    printSpace(x + 1);
    System.out.println("\". x = \" + x); // unchanged
}
```

**How parameters are passed**

- When the parameterized method call executes:
  - the value written is copied into the parameter variable
  - the method's code executes using that value

```java
Public static void main(String[] args) {
    int x = 3;
    printSpace(x + 1);
    System.out.println("\". x = \" + x); // unchanged
}
```

**Common errors**

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.

```java
printSpace();  // ERROR: parameter value required
```

- The value passed to a method must be of the correct type, matching the type of its parameter variable.

```java
printSpace(3.7);  // ERROR: must be of type int
```

- Exercise: Change the Stars program to use a parameterized static method for drawing lines of stars.
Stars solution

```java
// Prints several lines of stars.
// Uses a parameterized method to reuse redundancy.
public class Stars {
    public static void main(String[] args) {
        drawLine(5);
        drawLine(10);
    }
}
```

**Multiple parameters**

- Methods can accept multiple parameters.
  - The parameters are separated by commas.
  - When the method is called, it must be passed values for each of its parameters.

**Multiple parameters declaration syntax:**

```java
public static void <name> (<type> <name>, <type> <name>, ..., <type> <name>) {
    // statements...
}
```

**Multiple parameters call syntax:**

```java
<name> (<expression>, <expression>, ..., <expression>);
```

Multiple parameters example

```java
public static void main(String[] args) {
    printLine(10, 0);
    printLine(10, 8);
    printLine(8, 0);
    printLine(8, 8);
}
```

Output:

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

Exercise: Write an improved Stars program that draws boxes of stars using parameterized static methods.

Stars solution, cont'd.

```java
... // Prints a box of stars of the given size.
public static void drawBox(int width, int height) {
    drawLine(width);
    for (int i = 1; i < height - 1; i++) {
        printSpaces(width - 2);
        drawLine(width - 2);
    }
    drawLine(width);
    // Prints the given number of squares.
    public static void drawSquare(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
    }
    drawLine(width);
    // Prints a box of stars of the given size.
    public static void drawBox(int width, int height) {
        drawLine(width);
        for (int i = 1; i < height - 1; i++) {
            printSpaces(width - 2);
            drawLine(width - 2);
        }
        drawLine(width);
    }
}
```

Parameter "mystery" problem

**What is the output of the following program?**

```java
public class Mystery {
    public static void main(String[] args) {
        int x = 2, y = 3, z = 2;
        System.out.println(x * * + y * + z * *);
    }
}
```

Output:

```
** * * + * * + **
```
Parameter question

- Rewrite the following program to use parameterized methods:

```java
public class Triangle {
    public static void main(String[] args) {
        System.out.println("Height: ");
        System.out.println("Base: ");
        System.out.println("Area: ");
    }
}
```

Parameter answer

```java
public class Triangle {
    public static void main(String[] args) {
        System.out.println("Height: ");
        System.out.println("Base: ");
        System.out.println("Area: ");
    }
}
```

Parameter questions

- Write a method named `printDiamond` that accepts a number as a parameter and prints a diamond figure:

```
  *
 /\/
/  \
```

- Write a method named `multiplicationTable` that accepts a maximum integer as a parameter and prints a table of multiplication from 1 x 1 up to that integer times itself.

- Write a method named `bottlesOfBeer` that accepts an integer as a parameter and prints the "99 Bottles of Beer" song with that many verses.

Java's Math class

- Java has a class named `Math` with useful static methods and constants for performing calculations.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceil</td>
<td>Ceiling value</td>
<td><code>Math.ceil()</code></td>
<td><code>1.0</code></td>
</tr>
<tr>
<td>floor</td>
<td>Floor value</td>
<td><code>Math.floor()</code></td>
<td><code>0.0</code></td>
</tr>
<tr>
<td>abs</td>
<td>Absolute value</td>
<td><code>Math.abs()</code></td>
<td><code>5.0</code></td>
</tr>
<tr>
<td>round</td>
<td>Round value</td>
<td><code>Math.round()</code></td>
<td><code>42</code></td>
</tr>
<tr>
<td>max</td>
<td>Maximum value</td>
<td><code>Math.max()</code></td>
<td><code>100</code></td>
</tr>
<tr>
<td>min</td>
<td>Minimum value</td>
<td><code>Math.min()</code></td>
<td><code>2</code></td>
</tr>
<tr>
<td>random</td>
<td>Random double between 0 and 1</td>
<td><code>Math.random()</code></td>
<td><code>0.5</code></td>
</tr>
<tr>
<td>exp</td>
<td>Exponential</td>
<td><code>Math.exp()</code></td>
<td><code>2.71828</code></td>
</tr>
</tbody>
</table>

Methods that return values

- `return`: To send a value out as the result of a method, which can be used in an expression.

  - A `return` is like the opposite of a parameter.
  - Parameters pass information in from the caller to the method.
  - Return values pass information out from a method to its caller.

```
public class MathDemo {
    public static void main(String[] args) {
        double height = 10.0;
        double base = 20.0;
        double area = height * base / 2;
        System.out.println("Area: ");
        System.out.println(area);
    }
}
```
Math method examples

- Math method call syntax:
  Math. <method name> | <parameter(s)>

- Examples:
  double squareRoot = Math.sqrt(21.0); // 4.58
  System.out.println(squareRoot);
  int absoluteValue = Math.abs(-50); // 50
  System.out.println(absoluteValue);

- Notice that the preceding calls are used in expressions; they can be printed, stored into a variable, etc.

Math method questions

- Evaluate the following expressions:
  Math.abs(-1.23)
  Math.pow(2, 2)
  Math.pow(-2, 3)
  Math.abs(Math.abs(21.0) - Math.abs(236.0))
  Math.ceil(4.2) + Math.floor(5.9999)
  Math.abs(Math.min(-3, -51))

- Math.max and Math.min can be used to bound numbers. Consider an int variable named age.
  - What statement would replace negative ages with 0?
  - What statement would cap the maximum age to 40?

Methods that return values

- Syntax for methods that return a value:
  public static <type> <name> | <parameter(s)> | | <statement(s)>

- Returning a value from a method:
  public static double slope(int x1, int y1, int x2, int y2) { 
    double dx = x2 - x1;
    double dy = y2 - y1;
    return dy / dx;
  }

- Example:
  // returns the slope of the line between the given points.
  public static double slope(int x1, int y1, int x2, int y2) {
    double dx = x2 - x1;
    double dy = y2 - y1;
    return dy / dx;
  }

Return examples

// Computes length of right triangle hypotenuse given side lengths.
public static double hypotenuse(int a, int b) {
  return Math.sqrt(a * a + b * b);
}

// Rounds the given number to two decimal places.
// Example: round(10.345, 2) returns 10.35
public double round(double value) {
  return result = value + Math.pow(10, 2) / 100;
  // round to nearest integer
  return Math.round(result) / 100;
}

// Computes length of right triangle hypotenuse given side lengths.
public static double hypotenuse(int a, int b) {
  return Math.sqrt(a * a + b * b);
}

// Rounding the given number to two decimal places.
// Example: round(10.345, 2) returns 10.35
public double round(double value) {
  return result = value + Math.pow(10, 2) / 100;
  // round to nearest integer
  return Math.round(result) / 100;
}

Return questions

- Write a method named area that accepts a circle’s radius as a parameter and returns its area.
  * You may wish to use the constant Math.PI in your solution.

- Write a method named distanceFromOrigin that accepts x and y coordinates as parameters and returns the distance between that (x, y) point and the origin.

- Write a method named attendance that accepts a number of lectures attended by a student, and returns how many points a student receives for attendance. The student receives 2 points for each of the first 5 lectures and 1 point for each subsequent lecture.
How to comment: params

- If your method accepts parameters and/or returns a value, write a brief description of what the parameters are and what kind of value will be returned.
- In your comments, you can also write your assumptions about the values of the parameters.
- You may wish to give examples of what values your method returns for various input parameter values.

Examples:

```java
// This method returns the factorial of the given integer.
// The factorial is the product of all integers up to that number.
// Assume that the parameter value is non-negative.
// If n = 0, then the value is 1.
// In this case, the method is called without a parameter.
return n == 0 ? 1 : n * factorial(n - 1);
```

```
// Using objects
```

```
Reading: 3.3
```

```
Objects
```

- So far, we have seen:
  - methods, which represent behavior
  - variables, which represent data (categorized by types)

It is possible to create new types that are combinations of the existing types.
- Such types are called object types or reference types.
- Languages such as Java in which you can do this are called object-oriented programming languages.
- We will learn how to use some of Java’s objects.
- In Chapter 8 we will learn to create our own types of objects.

```
Objects and classes
```

- object: An entity that contains data and behavior.
  - There are variables inside the object, representing its data.
  - There are methods inside the object, representing its behavior.
- class: A program, or a template for a type of objects.

Examples:

```java
// The class String represents objects that store text characters.
// The class Point represents objects that store (x, y) data.
// The class Circle represents objects that read information from the keyboard, files, and other sources.
```

```
Constructing objects
```

- construct: To create a new object.
  - Objects are constructed with the new keyword.
  - Most objects must be constructed before they can be used.

Constructing objects, general syntax:

```java
<type> <name> = new <type> (<parameters>);
```

- Examples:
  ```java
  Point p = new Point(7, -4);
  JScrollPane window = new JScrollPane(300, 200);
  Color orange = new Color(255, 128, 0);
  ```
  ```java
  Classes' names are usually uppercase (e.g., Point, Color).
  ```
  ```java
  Strings are also objects, but are constructed without new:
  ```
  String name = "Amanda Ann Camp";
  ```

```
Calling methods of objects
```

- Objects contain methods that can be called by your program.
  - For example, a string's methods manipulate or process the text of that string.
  - When we call an object's method, we are sending a message to it.
  - We must specify which object we are sending the message to, and then write the method's name.

```java
// Calling a method of an object, general syntax:
<variable> .<method name> (<parameters>);
```

- The results will be different from one object to another.
- Examples:
  ```java
  String message = "Hello, World!";
  System.out.println(message.length()); // 13
  Point p1 = new Point(3, 8);
  p1.x = 100;
  System.out.println(p1.x); // 100
  ```
  ```java
  ```
  ```
Point objects

- Java has a class of objects named Point.
- To use Point, you must write: import java.awt.*;

Constructing a Point object, general syntax:
```
Point <name> = new Point(<x>, <y>);  // the origin, (0, 0)
```

- Examples:
  Point p1 = new Point(0, -2);
  Point p2 = new Point();

- Point objects are useful for several reasons:
  - They store two values, an (x, y) pair, in a single variable.
  - They have useful methods we can call in our programs.

Point data and methods

- Data stored in each Point object:
  ```
  Field name Description
  x Component of the point
  y Component of the point
  ```

- Useful methods of each Point object:
  ```
  Method name Description
  distanceTo(Point p) How far away the point is from point p
  translate(int x, int y) Add the point x and y to the given values
  ```

- Point objects can also be printed using println statements:
  ```
  Point p = new Point(5, -2);
  System.out.println(p);  // java.awt.Point{x=5,y=-2}
  ```

Using Point objects

- An example program that uses Point objects:
  ```
  import java.awt.*;
  public class PointExample {
    public static void main(String[] args) {
      // construct two Point objects
      Point p1 = new Point(0, 1);
      Point p2 = new Point(1, 0);
      // print each point and the distance apart
      System.out.println(p1);  // (0, 1)
      System.out.println(p2);  // (1, 0)
      double distance = p1.distanceTo(p2);
      // translate the point p2 to a new location
      p2.translate(5, 10);
      System.out.println(p2);  // (6, 10)
      System.out.println(distance);  // 5.0
    }
  }
  ```

Point objects question

- Write a program that computes a right triangle’s perimeter.
  - The perimeter is the sum of the triangle’s side lengths a + b + c.
  - Read values a and b and compute side length c as the distance between the points (0, 0) and (a, b).

```
side a = 12
side b = 5
perimeter c = 33.0
```

Point objects answer

```java
import java.awt.*;  // For Point
import java.util.*;  // For Scanner

public class TriangleExample {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Side a: ");
    int a = in.nextInt();
    System.out.print("Side b: ");
    int b = in.nextInt();
    Point p1 = new Point(0, 0);
    Point p2 = new Point(a, b);
    double c = p1.distanceTo(p2);
    System.out.println("Perimeter is: "+(a + b + c));
  }
}
```
**Swapping primitive values**

- Consider the following code to swap two int variables:
  ```java
  public static void main(String[] args) {
      int a = 3;
      int b = 5;
      System.out.println(a + " + b = ");
      // swap a with b
      int temp = a;
      a = b;
      b = temp;
      System.out.println(a + " + b = ");
  }
  ```

  - What is wrong with this code? What is its output?

**Swapping, corrected**

- When swapping, you should set aside one variable's value into a temporary variable, so it won’t be lost.
  ```java
  public static void main(String[] args) {
      int a = 3;
      int b = 5;
      System.out.println(a + " + b = ");
      // swap a with b
      int temp = a;
      a = b;
      b = temp;
      System.out.println(a + " + b = ");
  }
  ```

**A swap method?**

- Swapping is a common operation, so we might want to
  make it into a method.
- Does the following `swap` method work? Why or why not?
  ```java
  public static void main(String[] args) {
      int a = 3;
      int b = 5;
      System.out.println(a + " + b = ");
      // swap a with b
      swap(a, b);
      System.out.println(a + " + b = ");
  }
  ```

  ```java
  public static void swap(int a, int b) {
      int temp = a;
      a = b;
      b = temp;
  }
  ```

**Value semantics**

- **value semantics**: Behavior where variables are copied when assigned to each other or passed as parameters.
- Primitive types in Java use value semantics.
- When one variable is assigned to another, the value is copied.
- Modifying the value of one variable does not affect others.

**Modifying primitive parameters**

- When we call a method and pass primitive variable’s values as parameters, we can assign new values to the parameters inside the method.
  - But this does not affect the value of the variable that was passed; it's value was copied, and the two variables are otherwise distinct.

**Reference semantics**

- **reference semantics**: Behavior where variables refer to a common value when assigned to each other or passed as parameters.
  - Objects in Java use reference semantics.
  - Objects variables do not actually store an object; they store the address of an object's location in the computer memory.
  - Variables for objects are called reference variables.
  - We often draw reference variables as small boxes that point an arrow toward the object they refer to.

**Examples**

- **Example:**
  ```java
  Point pt = new Point();
  pt = new Point();
  ```
Multiple references

- If two reference variables are assigned to refer to the same object, the object is not copied.
  - Both variables literally share the same object.
  - Calling a method on either variable will modify the same object.

```java
public void p1Example()
{
    p1 = new Point(0, 0);
    p2 = new Point(1, 1);
    p3 = new Point(1, 1);
}
```

Output:

```
p1 [1, 1] p2 [1, 1] p3 [1, 1]
```

- Here 3 variables refer to 2 objects. If we change p3, will it also change? If we change p2, will p3 change?

Why references?

- The fact that objects are passed by reference was done for several reasons:
  - Efficiency. Objects can be large, bulky things. Having to copy them every time they are passed as parameters would slow down the program.
  - Sharing. Since objects hold important state and have behavior that modifies that state, it is often more desirable for them to be shared by parts of the program when they're passed as parameters. Often we want the changes to occur to the same object.

Objects as parameters

- When an object is passed as a parameter, the object is not copied. The same object is shared by the original variable and parameter.
  - If a method is called on the parameter, it will affect the original object that was passed to the method.
  - Since the variables p1 and the parameter refer to the same object, modifying one will also make a change in the other.

```java
public static void example(Point arg)
{
    arg = new Point(1, 1);
}
```

Output:

```
p1 [1, 1] p2 [1, 1] p3 [1, 1]
```

String objects

- string: A sequence of text characters.
  - One of the most common types of objects.
  - In Java, strings are represented as objects of class String.

- String variables can be declared and assigned, just like primitive values:
  ```java
  String name = "Larry Singer";
  int x = 5, y = 10;
  String point = "(" + x + ", " + y + ")";
  ```

- Unlike most other objects, a String is not created with new.
  - Examples:
    ```java
    String name = "Larry Singer";
    int x = 3, y = 2;
    String point = "(" + x + ", " + y + ")";
    ```

Indexes

- The characters in a String are each internally numbered with an index, starting with 0.
  - Example:
    ```java
    String name = "P. Diddy";
    ```
    ```java
    index 0 1 2 3 4 5 6 7
    character P . D i d d y
    ```
    - Individual characters are represented inside the String by values of a primitive type called char.
    - Literal char values are surrounded with apostrophe (single-quote) marks, such as ‘a’ or ‘4’.
    - An escape sequence can be represented as a char, such as \" (new-line character) or \’ (apostrophe).
String methods

- **Useful methods of each String object:**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>charAt(index)</code></td>
<td>Character at a specific index</td>
</tr>
<tr>
<td><code>length()</code></td>
<td>Returns the length of the string</td>
</tr>
<tr>
<td><code>substring(beginIndex, endIndex)</code></td>
<td>Returns a substring of the string</td>
</tr>
<tr>
<td><code>toUpperCase()</code></td>
<td>Converts all characters to uppercase</td>
</tr>
</tbody>
</table>

- **These methods are called using the dot notation:**
  ```java
  String example = "Speak friend and enter:",
  System.out.println(example.toUpperCase());
  ```

Modifying Strings

- **The methods that appear to modify a string (`substring`, `toUpperCase`, `toLowerCase`, etc.) actually create and return a new string.**
  ```java
  String s = "Lil bow wow",
  s.toUpperCase();
  System.out.println(s); // output: LIL BOW WOW
  ```

- **If you want to modify the variable, you must reassign it to store the result of the method call:**
  ```java
  String s = "Lil bow wow",
  s = s.toUpperCase();
  System.out.println(s); // output: LIL BOW WOW
  ```

String method examples

```java
// index 012345678901
String s1 = "Stuart Raves";
String s2 = "Marty Stargo"
System.out.println(s1.indexOf("s")); // 12
System.out.println(s1.substring(1, 4)); // tu

String s3 = s2.toUpperCase();
System.out.println(s3.substring(6, 10)); // STEP

String s4 = s1.substring(0, 6);
System.out.println(s4.toUpperCase()); // stuart
```

Interactive programs using Scanner objects

- **Reading:**
  ```java
  String reading = "3.4"
  System.out.println(reading);
  ```

Interactive programs

- **We have written programs that print console output.**
- **It is also possible to read input from the console:**
  ```java
  Scanner input = new Scanner(System.in);
  double value = input.nextDouble();
  System.out.println("Value: ", value);
  ```
  - The user types the input into the console.
  - We can capture the input and use it in our program.
  - Such a program is called an Interactive program.
- **Interactive programs can be challenging:**
  - Computers and users think in very different ways.
  - Users tend to misbehave.
Input and System.in

- We print output using an object named System.out.
  - This object has methods named println and print.
- We read input using an object named System.in.
  - System.in is not intended to be used directly.
  - We will use a second object, from a class called Scanner, to help us read input from System.in.

- Constructing a Scanner object to read console input:
  - Scanner console = new Scanner(System.in);
  - Examples:
    - System.out.println("Hello, world");
- Once we have constructed the scanner, we call various methods on it to read the input from the user.

Scanner methods

- Methods of Scanner that we will use in this chapter:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next()</td>
<td>Reads and returns user input as an int.</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>Reads and returns user input as a double.</td>
</tr>
<tr>
<td>next()</td>
<td>Reads and returns user input as a string.</td>
</tr>
</tbody>
</table>

- Each of these methods pauses your program until the user types input and presses Enter.
- Then the value typed is returned to your program.

- Example:
  - System.out.println("You are now ");
  - prompt.nextInt();

Java class libraries, import

- Java class libraries: A large set of Java classes available for you to use (part of the JDK).
- These objects are organized into groups named packages.
- To use the objects from a package, you must include an import declaration at the top of your program.

- Import declaration, general syntax:
  - import <package name> *;

- Scanner is in a package named java.util.
  - To use Scanner, put this at the start of your program:
  - import java.util.*;

Input tokens

- token: A unit of user input, as read by the Scanner.
  - Tokens are separated by whitespace (spaces, tabs, newlines).

- How many tokens appear on the following line of input?
  - 23 John Smith 42.3 "Hello World"

- When the token doesn't match the type the Scanner tries to read, the program crashes.

  - Example:
    - System.out.println("What is your name ");
    - String name = console.nextLine();
    - Output: User's input is unhandled:
      - What is your name: 

Example Scanner usage

- import java.util.*; // so that I can use Scanner

  public class ReadNameApp
  {
      public static void main(String[] args)
      {
          Scanner console = new Scanner(System.in);
          String name = console.nextLine();
          System.out.println("What is your first name ");
          String first = console.nextLine();
          System.out.println("What is your last name ");
          String last = console.nextLine();
          System.out.println("That's quite a name!");
      }
  }

- Output: (user input underlined)
  - What is your first name: John
  - What is your last name: Smith

Another Scanner example

- import java.util.*; // so that I can use Scanner

  public class Average
  {
      public static void main(String[] args)
      {
          Scanner console = new Scanner(System.in);
          System.out.println("Please type three numbers ");
          int num1 = console.nextInt();
          int num2 = console.nextInt();
          int num3 = console.nextInt();
          System.out.println("The average is "+ (num1 + num2 + num3) / 3);
      }
  }

- Output: (user input underlined)
  - Please type three numbers: 1.2 3.4 5.6

- Note that the Scanner can read multiple values from one line.
  - You can use the next() method for each separated value.
  - Or use the nextDouble() method if the values are all numbers.

- Next Example: K-4
Scanners as parameters

* If multiple methods read user input, declare a Scanner in main and pass it to each of them as a parameter.

    public static void main(String[] args)
    { Scanner console = new Scanner(System.in);
      int age = console.nextInt();
      System.out.println("The age is "+ age);
    }

public static float readDouble(Scanner console)
    { System.out.print("Type 3 numbers: ");
      int num = console.nextInt();
      float result = num + num;
      return result;
    }

Scanner BMI question

A person’s body mass index (BMI) is computed by the following formula:

\[ \text{BMI} = \frac{\text{weight}}{\text{height}^2} \]

* Write a program that produces the following output:

This program wants to know for two people and computes their body mass index (BMI) and weight status.

<table>
<thead>
<tr>
<th>Person</th>
<th>Body mass index</th>
<th>Weight status</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>25.45</td>
<td>Normal</td>
</tr>
<tr>
<td>P2</td>
<td>30.90</td>
<td>Obesity</td>
</tr>
</tbody>
</table>

    public class BMI {
      public static void main(String[] args)
      { System.out.print("Type 3 numbers: ");
        int num = console.nextInt();
        float result = num + num;
        return result;
      }
    }

Scanner BMI solution

// This program computes the person’s body mass index (BMI) // and requests their body weight and height. // (Written by J. B. & D. B.)
public class BMI {
  public static void main(String[] args)
  { Scanner console = new Scanner(System.in);
    System.out.print("Enter your weight: ");
    double weight = console.nextDouble();
    System.out.print("Enter your height: ");
    double height = console.nextDouble();
    double bmi = weight / (height * height);
    System.out.println("Your BMI is: "+ bmi);
  }
}

Scanner BMI solution, cont.

// This program computes a person’s body mass index (BMI) // and requests their body weight and height. // (Written by J. B. & D. B.)
public class BMI {
  public static void main(String[] args)
  { Scanner console = new Scanner(System.in);
    System.out.print("Enter your weight: ");
    double weight = console.nextDouble();
    System.out.print("Enter your height: ");
    double height = console.nextDouble();
    double bmi = weight / (height * height);
    System.out.println("Your BMI is: "+ bmi);
  }
}