

KIT101 Programming Fundamentals

PP Task 3.4 Methods for Calculation

Overview




Purpose: Learn how to create methods to calculate and return values (functions).

Task: Use the following instructions to implement a method that calculates a value and apply it to user-given inputs.

Learning Outcomes: 1  2  3 

Time: Complete and submit for feedback before Week 4.

Resources:

- Introductory Programming Notes:
 - 05 Using Objects
 - 07 Methods in Self-contained Programs
- Online Lecture Materials & Echo360:
 - Solving Problems with Computers:  Some commonly used objects
 - Making Your Own Methods:
 -  ...in self-contained programs
 -  Flow of Control When Calling Methods

Submission Details

Upload the following to the MyLO submission folder for this task:

- Program **source code** demonstrating creation and use of a method that calculates and returns a value
- A **screenshot** showing the execution of your program

Assessment Criteria

A  Completed submission will:

- Use your method to calculate two fixed and one user input-based results
- Demonstrate the use of methods with parameters that return a value
- Have a suitable header comment for the method, describing its purpose

- Follow the Java coding conventions used in the unit (layout, use of case in variable names, including author name and program description at the top)
- Compile, with the screenshot showing that it works

Instructions

Methods can be action-oriented (they *do something*) or act as functions (they *calculate something*). Methods that just do something have a return type of `void`, while those that generate a result have a return type that describes that result. For instance, Section 7 of the notes introduced a `circleArea` method, reproduced here with a slight change to its parameter list (this version accepts a `double`):

```
/** Calculates the area of a circle from its radius. */
public static double circleArea(double radius) {
    double area; //calculated area

    area = Math.PI * radius * radius;

    return area;
}
```

which could alternatively be written more succinctly as:

```
/** Calculates the area of a circle from its radius. */
public static double circleArea(double radius) {
    return Math.PI * radius * radius;
}
```

or even as:

```
/** Calculates the area of a circle from its radius. */
public static double circleArea(double radius) {
    return Math.PI * Math.pow(radius, 2);
}
```

All of these implementations are equivalent: they all calculate the area of a circle. The first is more verbose, but helpful if you're new to programming as each step in the process is spelled out. The second and third are more succinct: since the method doesn't need to do

anything else with its calculated result it doesn't *have* to store it. The third also makes use of another function, `Math.pow(x, y)`, to evaluate the expression x^y , equivalent to $radius^2$ in this case.

Note: A single program can have only *one* of these declarations in it, since each version has the same name and parameter list. They're included here to illustrate alternative ways of achieving the same result.

The `circleArea` method could be included in a program and then called from `main()`, as in:

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    double circleR; //circle's radius
    double circleA; //circle's area

    System.out.print("Enter circle's radius: ");
    circleR = sc.nextDouble();
    circleA = circleArea(circleR);
    System.out.println("Circle's area is " + circleA);
}
```

A sample run of this program, with **user input highlighted**:

```
Enter circle's radius: 21.7
Circle's area is 1479.3445646488976
```

Tip: That output has too many significant digits (digits after the decimal point) for a proper, commercial application. However, formatting real-number output in Java is a little fiddly. Let your tutor know if it's something you're interested in and we can provide suitable resources.

In this task you can **choose** to implement **one of the three following programs**:

- [Distance Travelled](#): Calculate the distance travelled from an initial velocity, acceleration and time.
- [Airspeed Velocity](#): Calculate the airspeed velocity of an unladen swallow.
- [Investment Calculator](#): Calculate how much interest you will get on an investment.

Instructions for each of these programs follow. Once you have completed the task, submit the .java source file and a screenshot of one run of your program.

Note: If equations below look 'weird' (for example, $U = \frac{fA}{s}$) then please let us know, as it means something has gone wrong with the maths rendering JavaScript behind this page.

Option 1: Distance Calculator

It is possible to determine the distance travelled given an initial velocity, acceleration and time. For example, if you start with a velocity of 8 m/s and accelerate at 0.1 m/s² for 5 minutes (300 seconds) then you will have travelled 6900 meters.

The following formula can be used to calculate the distance from initial velocity (v) acceleration (a) and time (t).

$$D = vt + 0.5 \times at^2$$

For example, the above distance is calculated using:

- $D = vt + 0.5 \times at^2$
- $D = 8 \times 300 + 0.5 \times 0.1 \times 300^2$
- $D = 6900$

Tip: The `Math.pow(x, y)` function calculates the result of x^y .

1. Create a program named **DistanceTravelled**, which will use the Scanner to read an input value from the user (so add `import java.util.Scanner;` at the top).
2. Declare a method named **distance**. This method will have parameters for initial velocity, time and acceleration (all `double` values), and it will return a `double` value.
3. The main method will print '**Distance 1**', calculate the distance based on an initial velocity **0** m/s, **120** seconds, with an acceleration of **0.07** m/s², and display the result.
4. The main method will print '**Distance 2**', calculate the distance based on an initial velocity **8.33** m/s, **120** seconds, with an acceleration of **0** m/s², and display the result.
5. Main should then ask the user to enter initial velocity, time, and acceleration, and output the distance for these values.

Here is a sample run of the program, where the user's input has been **highlighted** for convenience:

Distance Calculator

Distance 1: 504.00000000000006 m

Distance 2: 999.6 m

Input initial velocity (m/s): **4.5**

Input time (seconds): **300**

Input acceleration (m/s^2): **0.01**

CALCULATING...

Distance: 1800.0 m

Compile and run the program and check that you are getting the correct values, then submit the source code (DistanceTravelled.java) and screenshot of the program running.

Option 2: Airspeed Velocity

For this option you will create a program to answer the age old question “What is the airspeed velocity of an unladen swallow?”. Watch [this video](#) if you are uncertain of the importance of this information.

The airspeed velocity of a bird [can be calculated](#) using an equation based on the [Strouhal Number](#). From this information we can determine that the airspeed velocity of a bird (U) can be calculated from the frequency (f) at which the bird beats its wings multiplied by the amplitude in metres (A) of each wing stroke, divided by the Strouhal Number (s). This is shown in the following equation:

$$U = \frac{fA}{s}$$

For example: A Zebra Finch beats its wings at a frequency of 27 Hz (f) (Hz [‘hertz’] means beats per second). The amplitude of this stroke is 11 cm (A), equivalent to 0.11 m. If we assume a Strouhal Number of 0.33 for this flight we get an airspeed (U) of 9 meters per second, as shown in the following steps:

- $U = fA/s$
- $U = 27 \times 0.11/0.33$
- $U = 9 \text{ m/s}$

1. Create a program named **Airspeed**, which will use the Scanner to read an input value from the user (so add **import java.util.Scanner;** at the top).

2. Declare a method named **airSpeedVelocity**. This method will have parameters for frequency and amplitude, both of type **double**, and it will return a **double** value.
3. Inside the method, declare a constant for the Strouhal Number with the value 0.33. For example, **final double STROUHAL_NUM = 0.33;**
4. The main method will print '**African Swallow**', and calculate the airspeed based on a frequency of **15** Hz and amplitude of **21** cm, and display the result.
5. The main method will also print '**European Swallow**', calculate the airspeed based on a frequency of **14** Hz and an amplitude of **22** cm, and display the result.
6. The main method should then ask the user to enter a frequency and an amplitude and print the airspeed based on the values entered.

Hint: The user will enter the frequency in Hz and the amplitude in cm. Convert the amplitude to meters by dividing the user's input by 100.

Here is a sample run of the program, where the user's input has been **highlighted** for convenience:

AirSpeed Velocity Calculator

African Swallow: 9.545454545454545 metres/second

European Swallow: 9.333333333333334 metres/second

Input frequency (Hz): **5**

Input amplitude (cm): **40**

CALCULATING...

ANSWER IS: 6.0606060606060606 metres/second

Compile and run the program and check that you are getting the correct values, then submit the source code (AirSpeed.java) and screenshot of the program running.

Option 3: Investment Calculator

With compound interest, interest is paid on the principal and any interest received during the period of the investment. For example, if you invest \$1000 at 15% interest then in the first year you will receive \$150 interest. This interest is then added to your investment, and in the

second year you will receive \$172.50 interest (15% of the \$1150).

The following formula can be used to calculate the amount of interest an investment will receive over a period of time.

$$I = P(1 + i)^n - P$$

For example, after 3 years our \$1000 invested at 15% will have accrued \$520.875 in interest:

- $I = P(1 + i)^n - P$
- $I = 1000 \times (1 + 0.15)^3 - 1000$
- $I = 520.875$

Tip: The `Math.pow(x, y)` function calculates the result of x^y .

1. Create a program named **InvestmentCalc**, which will use the Scanner to read an input value from the user (so add `import java.util.Scanner;` at the top).
2. Declare a method named **compoundInterest**. This method will have parameters for principal, years (both `ints`) and rate (a `double`) and it will return a `double` value.
3. The main method will print 'Bank A', calculate the interest based on a principal of **\$1000**, 3 years, at **3.5%** (0.035), and display the result.
4. The main method will print 'Bank B', calculate the interest based on a principal of **\$1000**, 3 years, at **4.5%** (0.045), and display the result.
5. The main method should then ask the user to enter a principal and rate, and output the interest for a 5 year investment.

Hint: The user will enter the percent (so 15 for 15%) so the program will need to be divided this by 100.

Here is a sample run of the program, where the user's input has been **highlighted** for convenience:

Investment Calculator

Bank A: \$108.71787499999982

Bank B: \$141.16612499999974

Input principle (\$): **1000**

Input interest rate (percent): **15**

CALCULATING...

5 year investment: \$1011.3571874999993

Compile and run the program and check that you are getting the correct values, then submit the source code (InvestmentCalc.java) and screenshot of the program running.
