**Programming Principles 02 Tutorials 05**

**Task 1 – Enhancing the “convert”**

Create a new jave file in the **IDE** and copy-paste the following code into it:

|  |  |  |
| --- | --- | --- |
| **public class Convert{**    **public static final double MM\_IN\_INCH = 25.4;** | **Java** |  |
| **public static final double MM\_IN\_FOOT = 304.8;**  **public static final double MM\_IN\_YARD = 914.4;**  **public static final double MM\_IN\_MILE = 1609344;** |  |  |
|  |  |

**public static double cm2inches(double cm){**

**return cm \* 0.393;**

**}**

**public static double m2feet(double m){**

**return m \* 3.281;**

**}**

**public static double m2feet(double m){**

**return m \* 1.094;**

**}**

**public static double km2miles(double km){**

**return km \* 0.621**

**}**

Save the file as “**Convert.java**”, then create a **new java file** and copy-paste the following code into the main method of it:

|  |  |  |
| --- | --- | --- |
|  | **Java** |  |
|  |  |  |

**System.out.println(“20 metres is ”+ Convert.m2feet(20)+ “ feet.”);**

**System.out.println(“4 miles is ,”+ Convert.MM\_IN\_MILE \* 4+”, millimetres.”);**

Save this file (class name not important) in the **same project** as Convert.java, and then run the code to test it.

Once you’ve made sure that it is working, **enhance the Convert.java** to make it more useful in these ways:

1. Add a second parameter named “rounding” to each of the methods. The parameter should have a default value of 2, and should be used to round the result of the method.

Test your work by calling the methods in your second file:

* + Convert.cm2inches(7); should return a value of 2.75
  + Convert.m2yards(10, 1); should return a value of 10.9

Java doesn’t support default value to the parameter and see how it could be achieved - <https://stackoverflow.com/questions/997482/does-java-support-default-parameter-values>

1. Add a third parameter named “addUnit” to each of the methods. The parameter should have a default of False. If True is specified for the parameter when the method is called, the method should return a string of the result with the unit of measurement at the end, e.g. '2.75 inches' instead of 2.75. Test your work by calling the methods:

 convert.m2feet(5, 0, True); should return 16.0 feet

* convert.km2miles(2, True); should return 1.24 miles

Optional Parameters – common strategies and approaches in Java

<https://stackify.com/optional-parameters-java/>

**Task 2 – Input Type Validating methods**

Create a new java file in the **IDE** and write the code to implement the following methods.

So far in the unit, we've assumed that the user will input the correct type of data when prompted.

Hence, we’ve done things like converting input to an integer without making sure it actually is one:

|  |  |
| --- | --- |
| **Scanner input = new Scanner(System.in);**  **System.out.println('Enter an integer: ');**  **int intVal = input.nextInt();** | **Java** |
|  |  |

This works if the user types an integer, but if they don’t the program will end with an Exception:

**InputMismatchException**

Write a method of ensuring that the user enters a number when prompted, and re-prompts them if they enter something other than a number.

Create a method named “inputInt()” which works similar to the built-in “scanner.nextInt()”method in java, except that it will repeatedly re-prompt the user for input until they enter an integer. The pseudocode for the method is below.

|  |  |  |
| --- | --- | --- |
| **Define inputInt(prompt) method:** | **Pseudocode** |  |
| **Endless Loop:** |  |  |
|  |  |
| **Prompt for input with prompt parameter** |  |  |

**Try:**

**Convert response to int and store in numResponse variable**

**On InputMismatchException exception:**

**Print 'invalid input' error message**

**Continue to next iteration of loop**

**Return numResponse variable** (ends method / breaks loop)

As you can see, if there is an error trying to convert the input to an integer, the error message will be printed and the continue statement will go back to the start of the loop – re-prompting for a value without returning anything. Only when the conversion is successful will the method return/end.

Once you’ve written the code for the method, *test it*. If you enter anything other than an integer, you should receive an error message and be re-prompted. Once you enter an integer, the program should continue. Use the following code to test your method:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Enter an** | | **int: no** |
| **int value = inputInt('Enter an int: ');** | **Java** | **Invalid input: int required.** | | |
| **System.out.println('Value is', value);** |  | **Enter** | **an int: 7** | |
|  |
|  |  | **Value** | **is** | **7** |
|  |  |

Once your method is working correctly, copy and paste the code to create a second method in the same class. Name the second method “inputFloat()”, and make it work the same as “inputInt()”, but require the user to enter a floating point number.

**Task 3 – Enhancing the Input Type Validating Methods**

Once both “inputInt()” and “inputFloat()” are working, enhance both methods in the following ways to make them more useful:

1. Add second parameter named “errorMessage” to the methods. The parameter should have a default value of the “invalid input” error message currently used by the methods.

This parameter allows a custom error message to be shown instead of the default one.

* + inputInt('Enter an int: '); would use the default error message
  + inputInt('Enter your age: ', 'Enter age as an integer.'); would use “Enter age as an integer.” as the error message

1. Add two more parameters named “minValue” and “maxValue” to the methods. These parameters allow a minimum and maximum allowable value to be specified for the input. If the input is less than the minimum or greater than the maximum (assuming a value has been specified for that parameter), show an appropriate error message.

With these two enhancements, the code of the methods should resemble this pseudocode:

|  |  |
| --- | --- |
| **Define “inputInt” method:** | **Pseudocode** |

**Parameters: prompt (required),**

**errorMessage (default of default error message), minValue, maxValue**

**Endless Loop:**

**Prompt for input with prompt parameter**

**Try:**

**Convert response to int and store in numResponse variable**

**On ValueError exception:**

**Print errorMessage parameter**

**Continue to next iteration of loop**

**If numResponse < minValue:**

**Print “value below minimum” error message Continue to next iteration of loop**

**If numResponse > maxValue:**

**Print “value above maximum” error message Continue to next iteration of loop**

**Return numResponse variable** (ends method / breaks loop)

**Task 4 – Improving Prior Code**

Once you have completed Task 3, use the “inputInt()” and “inputFloat()” methods to improve code you wrote in prior workshops. A number of previous workshop tasks required integers or floats to be entered, but simply assumed that the value received was appropriate and tried to convert it without any error checking.

Treat your file containing the “inputInt()” and “inputFloat()” methods as a module (remember to remove any testing code from it): Import it into your previous workshop files and replace “input()” statements that are converted to int or float with your new methods.

Remember, programming is a cyclical process – it is very rare that you will write a piece of code perfectly the first time. You will often find yourself returning to previously written code once you find a way to refine or enhance it.

*That’s all for this workshop. If you haven’t completed the workshop or readings,*

*find time to do so before next week’s class.*