Algorithm Design

CS110 Java

Handout 1

Problem: A Basic Wages Problem

Write a program that computes the wages for a given employee. Assume employees get paid by the hour and there is no overtime work.

The objective of this app is to compute and display an employee's earned wages, formatted as currency.

Introduction to Algorithm Design

Step 1: Understand the problem.

```
This program will ask the employee to input their rate of pay and the number of
hours they worked. The wages will be calculated and displayed.
```

Step 2: Define the input requirements. Specify primitive data types (or classes) and identifier names for this input data.

```
double rate;         // Rate of pay
double hours;        // Number of hours worked
```

Step 3: Define the output requirements. Specify primitive data types (or classes) and identifier names.

```
double wages;        // Computed wages
```

Step 4: Define the logic and computations to achieve the required output.

```
wages = hours * rate;
```

TIP:

Designing the solution/algorithm is always a first task when building an app.
Coding the app is the last step.
A correct algorithm should be easily translated into Java.

```
import java.util.Scanner;

public class WagesApp {

    public static void main(String[] args) {


        //STEP 1: DECLARE VARIABLES
        double rate;             // Rate of pay
        double hours;            // Number of hours worked
        double wages;            // Computed wages



        //STEP 2: CREATE A SCANNER OBJECT TO READ KEYBOARD INPUT
        Scanner in = new Scanner(System.in);



        //STEP 3: DISPLAY PROGRAM MESSAGE
        System.out.println("This program will compute"
                    + "\nwages based on a rate of pay for"
                    + "\nfor each hour worked.\n");



        //STEP 4: PERFORM INPUT
        System.out.print("Enter the rate of pay: $");
        rate = in.nextDouble();
        System.out.print("Enter the number of hours worked: ");
        hours = in.nextDouble();



        //STEP 5: COMPUTE WAGES
        wages = hours * rate;



        //STEP 6: DISPLAY WAGES WITH TWO DIGITS AFTER THE DECIMAL POINT
        System.out.print("Wages: $");
        System.out.printf("%.2f", wages);
    }

}
```

Code notes:

1. **\n** is a "sequence" character that produces a newline.
2. When reading input from the keyboard, a **Scanner** object must be created.
3. **println** will display a newline following an output segment.

Format Notes:

Output values can be formatted using format codes. To specify a format, begin with a % character and end with a letter that indicates the format code.

For example,

```
double total = 72.4795;
System.out.printf("price:%6.2f", total);
```

prints the string price:, followed by a floating-point number with a width of 6 and a precision of 2. The width is the total number of characters to be printed: in this case, a space, the digits 72, a period, and two digits. If you increase the width, more spaces are added. The precision is the number of digits after the decimal point. The output produced is: price: 72.48

The following table shows common format types.

code/flag	Type/meaning	Example
d	Decimal integer	452
f	Fixed floating point	4.500
g	General floating point. Scientific notation is used for very large or small values.	4.5
e	Exponential floating point	3.23e+3
s	String	Price:
-	Left alignment	45 followed by spaces
0	Show leading zeros	002.53
+	Show a plus sign for positive numbers	+2.34
(	Enclose negative numbers in parentheses	(2.34)
'	Show decimal separators	23,456,456
s	Converts letters to uppercase	2.34E+1
n	Platform independent newline	

Practice: Explore the formats shown below.

```
//VARIABLE DECLARATIONS
int x = 23;
int y = 98;
double z = 1.2345678;
```

1. `System.out.printf("%d%d%f%n", x, y, z);`
2. `System.out.printf("%d %d %f%n", x, y, z);`
3. `System.out.printf("%10d %10d %10f%n", x, y, z);`
4. `System.out.printf("%-5d %-5d %10f%n", x, y, z);`
5. `System.out.printf("%5.1f%n", z);`
6. `System.out.printf("%25.5f%n", z);`