The programs for this lab should be stored in your Eclipse repository. Create a new Java project called **lab3** in your workspace. Remember to use a workspace location on your P: drive or a USB drive (not the C: drive).

The programs are beginning to get a bit more complicated. Rather than just jumping in and starting to write code, take a bit of time to think about how you will solve the problem. Develop an "***algorithm***" for the program before coding it.

- Identify the data (i.e., variables, etc.) needed and the names and types you will use for each piece of data. Use meaningful names instead of just using things like x, y,z1 ,z2, etc.
- Identify the ***built-in*** Java methods or classes you will use (e.g., Scanner).
- Create an outline of the steps that need to be performed to solve the problem. You don't have to describe the steps in Java – write out the steps in English and/or draw diagrams.

When your programs run correctly, turn in a printout of the Java code for each program. Make sure you include comments for your program (especially your name and the lab number).

Problem 1

Write a new class called CompletePay that extends the functionality of SimplePay from lab 2. Like that program, the new code should compute the total pay for a specified number of hours worked. The key difference is that you now need to take into account the *possibility* that the worker worked overtime. Maybe they worked overtime… maybe they didn't. Workers should get paid the standard pay rate for the first 40 hours and *time and a half* for overtime. In other words, hours *up to and including* 40 are paid at the regular rate. And any hours worked *over* 40 hours should be paid at 1.5 times the regular rate. Create an algorithm for the program before coding it. Also note that the resulting total pay is displayed using exactly two digits after the decimal point.

Example 1:

```
Enter the hours worked: 45.0
Enter the pay rate per hour: 9.25
Total pay is $439.38
```

Problem 2

Write a program named MinOfThree that asks the user to enter three integers and displays the minimum value of the three.

Example 1:

```
Enter number 1: 4
Enter number 2: 3
Enter number 3: 9
The minimum number is 3
```

Problem 3

Write a program named DogYears that computes the human equivalent of a dog's age. There are several different formulas for calculating human years and dog years, but we'll use a simple one to get started (see http://www.onlineconversion.com/dogyears.htm). Assume that a dog's age can be calculated as follows:

- 10.5 human years per year for the first 2 years
- then 4 human years per year for each year after that

Your program should ask the user how old the dog is, compute the dog's age in human years, and display the result. Create an algorithm for the program before coding it.

Example 1:

```
How old is your dog? 2
Your dog is 21 in human years
```

Example 2:

```
How old is your dog? 3.7
Your dog is 27.8 in human years
```

Problem 4

Write a program called MakeChange that determines the change to be dispensed from a vending machine. Items can cost between 25 cents and 95 cents, in 5-cent increments (25, 30, 35…90, or 95), and the machine accepts only a single dollar bill to pay for the item. Think about how you are going to make change and write an algorithm for the program first. *Change should be given using coins of the greatest value possible*. An obvious approach to this problem uses a large if-else statement. But it is possible to do this in different ways! Any approach is okay.

Example 1:

```
Enter the price of the item (from 25 cents to 95 cents): 45
You bought an item for 45 cents and deposited a dollar,
so your change is:
2 quarter(s)
1 nickel(s)
```

Example 2:

```
Enter the price of the item (from 25 cents to 95 cents): 80
You bought an item for 80 cents and deposited a dollar,
so your change is:
2 dime(s)
```