How to Convert Pseudocode to Java Code

By Dr. Ruimin Hu

Part 1

Part 1

General Syntax of Java Language

(Let us use the pay roll program used in our pseudocode as an example. Code segments are in textboxes)

1. Java Class

All code is written in a block called **`class`**, enclosed in a pair of curly braces, { }, which is like a container to hold source code. For example,

```
class PayRoll
{

}
```

- **`class`** is a keyword to start a Java program.
- **`PayRoll`** is a name (class name), named by a programmer, for a program. It starts with a uppercase letter. The java file should be saved as the name given to the class (for this example, `PayRoll.java`).

2. The `main()` Method

Inside a class block, programmers convert their design using pseudocode into real Java code.

Any executable class, called an application, has to contain the `main()` method. The `main()` method header has been pre-written and built in Java language. The header is:

```
class PayRoll
{
        public static void main(String args[])
        {

        }
}
```

So every time when you start a new application, just copy this set of code. The only thing needed to be changed is the class name. When you save your code, make sure your class name matches your source code name; otherwise, you cannot compile your code.

3. Steps of Computer Operations in a Non-Modular Program

For a non-modular program, you implement the following steps designed in pseudocode in the {} block under the `main()` method:

Step 1: Declare variables and constants (if any) to store input and result.

Step 2: Get data input.

Step 3: Process data and store the result in a variable.

Step 4: Output data and result.

Before writing Java code, please notice that:

- In pseudocode, these keywords, **Declare**, **Input**, **Set**, and **Display**, indicate the four steps; but in Java, there are NOT these keywords.
- Names of primitive data types are different: **Real** is **`double`**, **Integer** is **`int`**, **Character** is **`char`**, and so on. **String** is a pre-defined class in Java, so **String** is still `String`.
- Each statement, which is placed in a {} block, ends with a semicolon(;).

Now, let us learn how Java language implements the four steps in the `main()` method:

Step 1: Declare variables and constants (if any) to store input and result.

- Variable declaration syntax:
  ```
  dataType variableName;
  ```
  For example:
  ```
  double hours;
  ```

- Constant declaration syntax:
  The keyword, **`final`**, (not constant) is used to declare a constant. Remember a constant value has to be initialized.
  ```
  final dataType contantName = inializedValue;
  ```
  For example:
  ```
  final double PAY_RATE = 8.5;
  ```

```
class PayRoll
{
       public static void main(String args[])
       {
            final double PAY_RATE = 8.5;
            double hours;
            double grossPay;
       }
}
```

Step 2: Get data input

Inputting data from the Keyboard cannot be simply implemented using a keyword as we use for pseudocode.

One of ways to get user inputs from the keyboard is to use the **`Scanner`** class (in **`java.util`** package) in Java library. There are following steps to use Scanner class:

1) Before writing the class in which the **`Scanner`** is used, you need to import the package. The syntax is:

```
import java.util.Scanner;
```

2) In the `main()` method, create an object to use `Scanner` class (Now you just copy the following line; I will explain this when we learn object-oriented programming):

```
Scanner keyboard = new Scanner(System.in);
```

3) Then use the object variable, `keyboard`, with the dot (.) to call methods (functions) defined in the `Scanner` class to get inputs for different types of data:
   - **`nextInt():`** returns an integer value.
   - **`nextDouble()`**: returns a double value.
   - **`next():`** returns a String (single word).
   - **`nextLine():`** returns a String (multiple words).

   Make sure you use variables in right types to store user inputs. For example, a **`double`** variable, hours, is used to store a returned **`double`** value:

```
hours = keyboard.nextDouble();
```

```
import java.util.Scanner;
class PayRoll
{
       public static void main(String args[])
       {

            final double PAY_RATE = 8.5;
            double hours;
            double grossPay;

            Scanner keyboard = new Scanner(System.in);

            System.out.print("Please enter hours worked: ");
            hours = keyboard.nextDouble();   //Input hours
       }
}
```

Step 3: Process data and store the result into a variable

The most common data processing in programming is to perform calculation on data using mathematical operators and/or math functions in Java library(we will learn this later), and then assign the calculated value (result) to a variable.

The mathematical operators in programming are:

Math Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modular – the result is a reminder of an integer division

In the example, the data processing is to calculate the gross pay, and then store the result in the variable, `grossPay`:

```
import java.util.Scanner;
class PayRoll
{
       public static void main(String args[])
       {

            final double PAY_RATE = 8.5;
            double hours;
            double grossPay;

            Scanner keyboard = new Scanner(System.in);

            System.out.print("Please enter hours worked: ");
            hours = keyboard.nextDouble();

            grossPay = PAY_RATE * hours;
       }
}
```

Step 4: Output data and result

Finally, you need to display information on the screen.

Output of information is implemented by the class `System`, its object `out`, and its method `println()` or `print()` in this format:

The parameter of the method is a `String`, which is the content you want to display.

```
System.out.print();

System.out.println();
```

The difference between the two method is that the `println()` method creates a new line after displaying contents, while the `print()` method does not.

Both methods can accept one String, so you have to use the concatenate operator, +, to connect multiple strings into one. For example:

```
System.out.println("Hours worked: " + hours);
```

If there is no parameter, the `println()` method just makes an empty line.

With output statements, the program is complete!

```
import java.util.Scanner;
class PayRoll
{
       public static void main(String args[])
       {

            final double PAY_RATE = 8.5;
            double hours;
            double grossPay;

            Scanner keyboard = new Scanner(System.in);

            System.out.print("Please enter hours worked: ");
            hours = keyboard.nextDouble();

            grossPay = PAY_RATE * hours;
            System.out.println();
            System.out.println("Hours worked: " + hours);
            System.out.println("Hourly pay rate: " + PAY_RATE);
            System.out.println("Weekly pay: " + grossPay);
      }
}
```

4. Comments in a Program

Programmers can make comments (notes) in java files. Comments are ignored by the compiler. There are two types of comments in Java:

// makes a single line comments
/* … */ makes multiple line comments

The following is the complete program with comments. In the `main()` method, I wrote original pseudocode in comments for comparison purposes.

```
import java.util.Scanner;  //for keyboard input
public class PayRoll  //public keyword is optional in this line
{
     public static void main(String args[])
     {

          //Declare variables and constant to store data
          double hours;              //Declare Real hours
          final double PAY_RATE = 8.5;//Constant Real PAY_RATE = 8.5
          double grossPay;           //Declare Real grossPay

          //Input data
          Scanner keyboard = new Scanner(System.in);
          System.out.print("Please enter hours worked: ");
          hours = keyboard.nextDouble();   //Input hours

          //Process data (calculate weeklyPay)
          grossPay = PAY_RATE * hours; //Set grossyPay=PAY_RATE*hours

          //output
          System.out.println();  //Display an empty line
          System.out.println("Hours worked: " + hours);
                              //Display  "Hours worked:", hours
          System.out.println("Hourly pay rate: " + PAY_RATE);
                         //Display  "Hourly pay rate:" PAY_RATE
          System.out.println("Weekly pay: " + grossPay);
                         //Display  "Weekly pay:", grossyPay
     }  //end main() method
  }  //end class
```

Syntax for Modular Programs

The previous `PayRoll` program only contains one module, `main()`, which is essential for an executable program (application).

In a modular program, beside the `main()` module, a programmer can create modules and functions to perform different tasks, and then call (use) them in the `main()`. In Java, all modules and functions are called **methods**.

- A module is a void method, which DOES NOT return a value.
- A function is a method, which RETURNS a value

In Java, there are NOT these words, **Module**, **Function**, and **End**. The definition of a method is written in a block enclosed in pairs of curly braces, {}.

Like a class header, a method header is written before the open brace {, and **DOES NOT** end with a semicolon!

Now let us convert the previous example into a modular program.

1. Declare Global Constant

A global constant can be declared outside of all methods, and shared by all methods. To declare a global constant, start with a keyword, **static**.

```
import java.util.Scanner;
public class PayRoll02
{
   static final double PAY_RATE = 8.5;
   public static void main(String[] args)
   {

   }
}
```

2. Define Methods

A method's definition starts with two keywords: **public** and **static**.

- Function – the method returns a value. General syntax:

```
public static returnType methodName(dataType paraName, …)
{
      //statements
      //return statement
}
```

A function has to have return type in the header, and return statement at the end of the body. If there is any parameter, list type and name in the ().

For example, add a function to perform the input task:

```
import java.util.Scanner;
public class PayRoll02
{
      static final double PAY_RATE = 8.5;
      public static void main(String[] args)
      {

      }

      public static double getInput()
      {
            Scanner keyboard = new Scanner(System.in);
            double hours;
            System.out.print("Please enter hours: ");
            hours = keyboard.nextDouble();
            return hours;
      }
 }
```

Add another function to process data – calculate the gross pay. To calculate the pay the function needs to have an input, hours worked. So there is a parameter for the function.

```
import java.util.Scanner;
public class PayRoll02
{
      static final double PAY_RATE = 8.5;
      public static void main(String[] args)
      {

      }

      public static double getInput()
      {
            Scanner keyboard = new Scanner(System.in);
            double hours;
            System.out.print("Please enter hours: ");
            hours = keyboard.nextDouble();
            return hours;
      }

      public static double calcuPay(double hours)
      {
            double grossPay;
            grossPay = hours * PAY_RATE;
           return grossPay;
      }
 }
```

- Module - the method DOES NOT return a value. General syntax:

```
public static void methodName(dataType paraName, …)
{
      //statements
      //no return statement here!
}
```

A module has no return type. We cannot leave that spot empty. The word **void** there indicates this is a method without returning a value.

For example, create a module to output the information:

```
import java.util.Scanner;
public class PayRoll02
{
	static final double PAY_RATE = 8.5;
	public static void main(String[] args)
	{

	}

	public static double getInput()
	{
		Scanner keyboard = new Scanner(System.in);
		double hours;
		System.out.print("Please enter hours: ");
		hours = keyboard.nextDouble();
		return hours;
	}

	public static double calcuPay(double hours)
	{
		double grossPay;
		grossPay = hours * PAY_RATE;
		return grossPay;
	}

	public static void output(double h, double gp)
	{
		System.out.println("Hours worked: " + h);
		System.out.println("Pay Rate: " + PAY_RATE);
		System.out.println("Total Pay: " + gp);
	}
 }
```

3. Call (Use) Methods

If methods have been defined, you need to call them in another method, e.g., in the `main()` method; otherwise they are not in use.

- To call a method (function) which returns a value, you need to declare a variable to store the returned value. For example:

```
double hours = 0.0;
hours = getInput();
```

- To call a void method (module) which does not return a value, you call the method name directly, and provide argument as needed. For example:

```
output(hours, grossPay);
```

Let us take a look at a whole executable program. Please pay attention to each method call and the corresponding method definition (in the same color)!

```
import java.util.Scanner;
public class PayRoll02
{
      static final double PAY_RATE = 8.5;

      public static void main(String[] args)
      {
            double hours = 0.0;
            double grossPay = 0.0;

            hours = getInput();
            grossPay = calcuPay(hours);
            output(hours, grossPay);
      }

      public static double getInput()
      {
            Scanner keyboard = new Scanner(System.in);
            double hours;
            System.out.print("Please enter hours: ");
            hours = keyboard.nextDouble();
            return hours;
      }

      public static double calcuPay(double hours)
      {
            double grossPay;
            grossPay = hours * PAY_RATE;
            return grossPay;
      }

      public static void output(double h, double gp)
      {
            System.out.println("Hours worked: " + h);
            System.out.println("Pay Rate: " + PAY_RATE);
            System.out.println("Total Pay: " + gp);
      }
 }
```

The following is the complete program comments. Original pseudocode is in comments for comparison purposes.

```
import java.util.Scanner;
public class PayRoll
{
   //Declare a global constant
   static final double PAY_RATE = 8.5;  //Constant Real PAY_RATE = 8.5

   public static void main(String[] args) //Module main()
   {
      //Declare variables
      double hours = 0.0;    //Declare Real hours = 0.0
      double grossPay = 0.0; //Declare Real grossPay = 0.0

      hours = getInput();      //Set hours = getInput()
      grossPay = calcuPay(hours);//Set grossPay = calPay(hours)
      output(hours, grossPay); //Call output(hours, grossPay)
   } //End Main()

   //Define method(function), getInput()
   public static double getInput()    //Function Real getInput()
   {
      Scanner keyboard = new Scanner(System.in);
      double hours;                              //Declare Real hours
      System.out.print("Please enter hours: ");
                                    //Display "Please enter hours: "
      hours = keyboard.nextDouble();             //Input hours
      return hours;                              //Return hours
   }//End Function

   //Define method(function), calcuPay()
   public static double calcuPay(double hours)
                                 //Function Real calcuPay (Real hours)
   {
      double grossPay;            //Declare Real grosspay
      grossPay = hours * PAY_RATE;//Set grossPay = hours* PAY_RATE
      return grossPay;                  // Return grossPay
   } //End Function

   //Define method(module), output()
   public static void output(double h, double gp)
                                    //Module output(Real h, Real gp)
   {
      System.out.println("Hours worked: " + h);
                                     //Display "Hours worked:  ",  h
      System.out.println("Pay Rate: " + PAY_RATE);
                                   //Display "Pay Rate : ", PAY_RATE
      System.out.println("Total Pay: " + gp);
                                      //Display "Total Pay:  ",  gp
   } //End Module
} //End Class
```

Common Syntax Errors in Java Programming

- The most common error is "symbol is not found" which is caused by mistyping identifier (name of variable, constant, object, class, and module/function) or using undeclared/undefined variable, constant, object, class, module/function and so on.
- Mismatch the curly braces, {}.
- Miss a semicolon for a statement.
- Place a semicolon at the end of a class's or method's header.

Some Programming Tips for Beginners

- It is a good practice to write a pair of curly braces, {}, when you start **defining a class or method** in case mismatching the {}.
- Compile and run your code **block by block**, even **statement by statement**. This makes debug easier! DO NOT wait to compile code after you finish the whole program!
- To make your code more readable and make debug easier,
    - Indent code
    - Leave some empty lines and space
    - Make notes (comments)