PKG 09: METHODS

From PKG 07, Nested Loops:

```
public class LoopsVsMethods {

    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.print("Row " + i + ": ");
            for (int j = 0; j < 5; j++) {
                System.out.print(j + " ");
            }
            System.out.println("");
        }

        System.out.println("\n Doing other things...\n");

        for (int i = 0; i < 5; i++) {
            System.out.print("Row " + i + ": ");
            for (int j = 0; j < 5; j++) {
                System.out.print(j + " ");
            }
            System.out.println("");
        }
    }
}
```

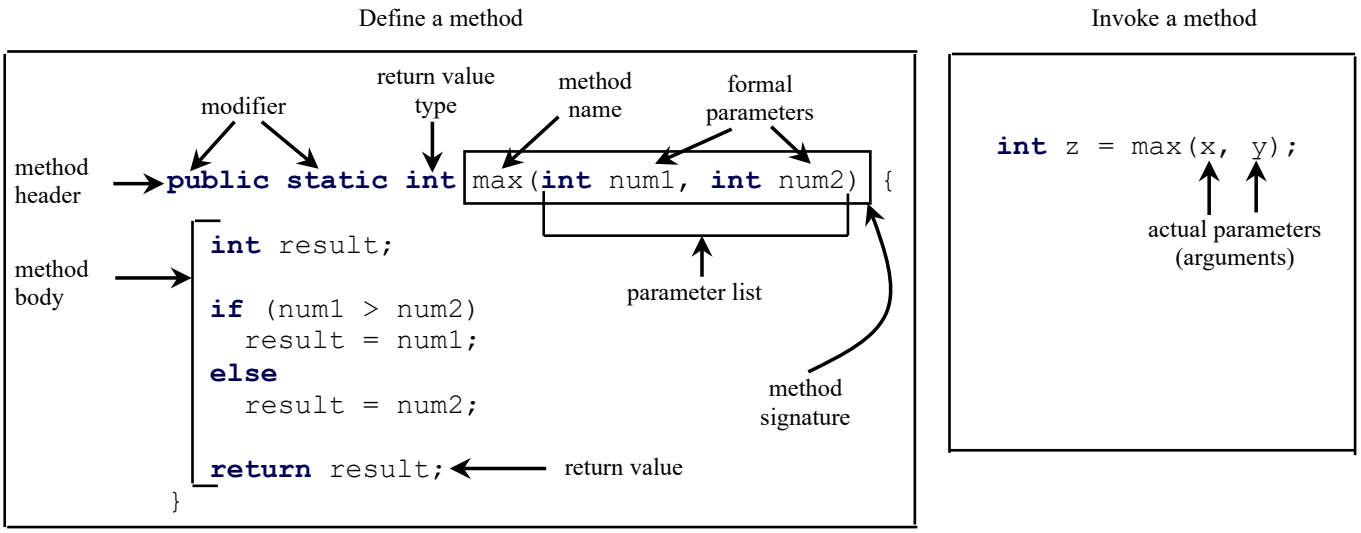
```
Row 0: 0 1 2 3 4
Row 1: 0 1 2 3 4
Row 2: 0 1 2 3 4
Row 3: 0 1 2 3 4
Row 4: 0 1 2 3 4

 Doing other things...

Row 0: 0 1 2 3 4
Row 1: 0 1 2 3 4
Row 2: 0 1 2 3 4
Row 3: 0 1 2 3 4
Row 4: 0 1 2 3 4
```

CODE REUSE, also called software reuse, is the use of existing software, or software knowledge, to build new software, following the reusability principles.

en.wikipedia.org/wiki/Code_reuse

```
public class LoopsVsMethods02 {

    public static void main(String[] args) {
        traverseBuilding(5, 5);
        traverseBuilding(2, 5);
        traverseBuilding(1, 8);
        traverseBuilding(5, 3);
    }

    public static void traverseBuilding(int floors, int offices) {
        System.out.println("Method STARTS");
        for (int i = 0; i < floors; i++) {
            System.out.print("Row " + i + ": ");
            for (int j = 0; j < offices; j++) {
                System.out.print(j + " ");
            }
            System.out.println("");
        }
        System.out.println("Method ENDS\n");
    }
}
```

```
Method STARTS
Row 0: 0 1 2 3 4
Row 1: 0 1 2 3 4
Row 2: 0 1 2 3 4
Row 3: 0 1 2 3 4
Row 4: 0 1 2 3 4
Method ENDS

Method STARTS
Row 0: 0 1 2 3 4
Row 1: 0 1 2 3 4
Method ENDS

Method STARTS
Row 0: 0 1 2 3 4 5 6 7
Method ENDS

Method STARTS
Row 0: 0 1 2
Row 1: 0 1 2
Row 2: 0 1 2
Row 3: 0 1 2
Row 4: 0 1 2
Method ENDS
```




```
class Method01 {

    public static void main(String[] args) {
        displayGreeting();
    }

    public static void displayGreeting() {
        System.out.println("Happy Thanksgiving!");
    }
}
```

```
Happy Thanksgiving!
```

```
class Method02 {

    public static void main(String[] args) {
        displayGreeting("Thanksgiving");
    }

    public static void displayGreeting(String s) {
        System.out.println("Happy " + s + "!");
    }
}
```

```
Happy Thanksgiving!
```

```
class Method03 {

    public static void main(String[] args) {
        String greeting = displayGreeting("Thanksgiving");
        System.out.println(greeting);
    }

    public static String displayGreeting(String s) {
        String resultS = "Happy " + s + "!";
        return resultS;
    }
}
```

```
Happy Thanksgiving!
```

METHODS, Daniel Liang

Let us practice and understand these 13 points by Daniel Liang thoroughly:

1. Making programs modular and reusable is one of the central goals in software engineering. Java provides many powerful constructs that help to achieve this goal. Methods are one such construct.

2. The method header specifies the modifiers, return value type, method name, and parameters of the method. The static modifier is used for all the methods in this chapter.

3. A method may return a value. The returnValueType is the data type of the value the method returns. If the method does not return a value, the returnValueType is the keyword void.

4. The parameter list refers to the type, order, and number of a method's parameters. The method name and the parameter list together constitute the method signature. Parameters are optional; that is, a method doesn't need to contain any parameters.

5. A return statement can also be used in a void method for terminating the method and returning to the method's caller. This is useful occasionally for circumventing the normal flow of control in a method.

6. The arguments that are passed to a method should have the same number, type, and order as the parameters in the method signature.

7. When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method-ending closing brace is reached.

8. A value-returning method can also be invoked as a statement in Java. In this case, the caller simply ignores the return value.

9. A method can be overloaded. This means that two methods can have the same name, as long as their method parameter lists differ.

10. A variable declared in a method is called a local variable. The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared and initialized before it is used.

11. Method abstraction is achieved by separating the use of a method from its implementation. The client can use a method without knowing how it is implemented. The details of the implementation are encapsulated in the method and hidden from the client who invokes the method. This is known as information hiding or encapsulation.

12. Method abstraction modularizes programs in a neat, hierarchical manner. Programs written as collections of concise methods are easier to write, debug, maintain, and modify than would otherwise be the case. This writing style also promotes method reusability.

13. When implementing a large program, use the top-down and/or bottom-up coding approach. Do not write the entire program at once. This approach may seem to take more time for coding (because you are repeatedly compiling and running the program), but it actually saves time and makes debugging easier.

METHODS, CHALLENGE

Challenge 1: Please rewrite the below program using methods.

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

        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user for input
        System.out.print("Your name: ");
        String name = input.nextLine();

        System.out.print("Your age: ");
        int age = input.nextInt();

        System.out.print("Your score: ");
        double testscore = input.nextDouble();

        char grade;
        String message;

        // Determine grade
        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }

        // Determine message
        switch (grade) {

            case 'A':
            case 'B':
            case 'C':
                message = "Keep up the good work.";
                break;
            default:
                message = "Please see your professor.";
        }

        // Output
        System.out.println(name + " | " + age + " | " + grade + "\n" + message);
    }
```

Challenge 2: A suggestion for the Extra Credit part of ASMT 2:

```
      if (answer == x) {

            runBmiStandardVersion();      // ASMT 01
      }

      if (answer == y) {

            runBmiProVersion();           // ASMT 02
      }
```

Let us all aim to complete this part. Let us discuss!