Methods

Introduction

In this unit, we are going to learn to build reusable code segments.

Methods

A **method** is a group of code that can be called to perform a task. These tasks are usually simple, but some methods do complex operations. Methods are usually built to complete tasks that are performed many times while a program is running.

The main two benefits of using methods in a program are:

- **Readability** – The name of a method will tell you what is does without requiring you to analyze its code.
- **Maintainability** – The code for a task is kept in one place, rather than being rewritten multiple times throughout the program.

Method Parts

- **Return Type** – The type of data the method gives back.
  - void – the return type of a method that does not return anything.
- **Method Name** – A name that describes what the method does.
- **Parameters** – Data the method needs in order to complete its task.

The format for a method:

```
public static returnType methodName(parameters)
{
        // Code
}
```

Example:

```
// Pre: Sent two ints
// Post: returns the sum of the received ints
public static int add(int x, int y)
{
        int sum = x+y;
        return sum;
}
```

Calling a Method

To run a method you must use its name and send it values for each of its parameters. Methods that have a return value that is not void will send the result back to the place where the method was called.

The format for calling a method is as follows:

```
methodName (parameters)
```

Passing values

Values must be passed in the order they are listed in the method's parameter list.

Parameters by Value

Primitive values are sent **by value** to methods. This means copies of the values are received by the method. When the copy is changed it will have no effect on the original value.

Parameters by Reference

Objects and arrays are passed to methods **by reference**. By reference passing is when the memory address of a value is passed to the method. When you change the value in a method it changes the original.

Note: Although Strings and Wrappers are Objects they work similar to a primitive value when changed by a method.

Example:

```
int d = 60;
int a = 5;
int result = add(a,d)
System.out.println(a+“ + ”+ d + “ = ”+ result);
```

Output:

```
5 + 60 = 65
```

Method Overloading

Method Overloading is when a two or more methods have the same name, but different parameters. Overloaded methods can have different return types, but they are not required to. The computer figures out what method to run based on the combination of the methods name and its parameters, so the combination must be unique.

Example:

```
// Pre: Sent two ints
// Post returns the sum of the received ints
public static int add(int x, int y)
{
    return x+y;
}

// Pre: Sent two doubles
// Post returns the sum of the received doubles
public static double add(double x, double y)
{
    return x+y;
}
```

Note: When ints are sent to add the top method is run, and when doubles are sent the bottom method is run.

Recursion

Recursion is when a method calls itself. The basis of recursion is that a method does some work on a problem and then calls itself to continue the work. The method will stop calling itself when it gets to a **base case**, a simple task that does not need any further work.

Rules for Recursion:

- Must have a base case – The method must have a base case that covers the simplest form of the problem that all problems will eventually reach.
- Must recurse – The method must call itself
- Must solve the problem – The method will always need to solve the problem

Benefits / Disadvantages of recursion

- Recursive solutions tend to take less code to write and are simpler to read.
- Recursive methods take more memory than iterative solutions. This is due to creating variables for the parameters each time the method is called.

Example:

```
// The below recursive method finds the factorial of a number.
// Pre: Sent a number
// Post returns the factorial of the received number
public static int factorial(int a)
{
        if(a ==1)
                return 1;
        else
                return a * factorial(a-1);
}

public static void main(String[] args)
{
        int fact3 = factorial(3);
        int fact5 = factorial(5);
        System.out.println("The factorial of 3 is "+ fact3+".");
        System.out.println("The factorial of 5 is "+ fact5+".");
}
```

Example Output:

```
The factorial of 3 is 6.
The factorial of 5 is 120.
```

Blue Pelican Sections

Lesson 20

Terms

Base Case	The simplest form of a recursive problem.
By Reference	When the address of Objects are sent to methods, allowing the originals to be changed.
By Value	When copies of primitives are sent to methods.
Maintainability	How easy it is to fix and update the code of a program.
Method	A group of code that can be called to perform a task.
Method Name	The name of a method.
Method Overloading	When two methods have the same name, but different parameters.
Parameters	Data the method needs to perform its task.
Readability	How easy a programs code is to read and understand.
Recursion	When a method calls itself.
Return Type	The type of data a method gives back.