METHOD OVERLOADING

Passing Information to a Method

- **Parameters** are the variables listed in a method declaration.
  Ex. `public double computeSalePrice(double origPrice, double discountRate) { }`
- **Arguments** are the actual values that are passed in when the method is invoked.
  Ex. `computeSalePrice(37.50, 0.15)`
- Java is a **pass-by-value** language. This means that a copy of the variable is made, and the method receives that copy.
  Ex. `main()` method calls `newNumber()` method

```
public static void main(String[] args) {
    int num = 1;
    newNumber(2);
    System.out.println(num);
}
public static void newNumber(int num) {
    num = 3;
}
```

  Output: 1
  Explanation: The variable `num` in `main()` does not change even after calling it because no assignments are made to it.

Overloading Methods

- A **method signature** consists of the method's name and the parameter types.
  Ex. The method signature of `computeSalePrice(double origPrice, double discountRate)` is
  `computeSalePrice(double, double)`
- **Method overloading** occurs when there are different method signatures with the same name but different type parameters.
  Ex.

```
public void draw(String s) { }
public void draw(int i) { }
public void draw(int i, double f) { }
```

  Thus, you can create statements like the following:

```
draw("Circle");
draw(10);
draw(10, 5.0);
```

- Overloaded methods are differentiated by the number and the type of arguments passed into the method.
- You cannot declare more than one (1) method with the same name and the same number and type of arguments.
- You cannot declare two (2) methods with the same signature even if they have different return types.
  Ex.

```
public void draw(int i) { }
public int draw(int i) { } //does not compile
```

Autoboxing

- **Autoboxing** is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes.
  Ex. `int` to `Integer`, `double` to `Double`
- When the primitive type version is not present, Java performs autoboxing.

```
public void draw(Integer i) {
    System.out.print("Integer");
}
```

  Calling statement: `draw(10);`
  Output: `Integer`
- When the primitive type version is present, Java does not need to perform autoboxing.
  Ex.

```
public void draw(Integer i) {
    System.out.print("Integer");
}
public void draw(int i) {
    System.out.print("int");
}
```

  Calling statement: `draw(10);`
  Output: `int`

References:
Baesens, B., Backiel, A., & Broucke, S. (2015). *Beginning Java programming: The object-oriented approach*. Indiana: John Wiley & Sons, Inc.
Oracle Docs (n.d.). *Citing sources.* Retrieved from https://docs.oracle.com/javase/tutorial/java/javaOO/index.html