An Introduction to Polymorphism in Java:

Method Overloading
In this section, you will learn about one of Java’s most exciting features: method overloading. In Java, two or more methods within the same class can share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be overloaded, and the process is referred to as method overloading. Method overloading is one of the ways that Java implements polymorphism.

In general, to overload a method, simply declare different versions of it. The compiler takes care of the rest. You must observe one important restriction: the type and/or number of the parameters of each overloaded method must differ. It is not sufficient for two methods to differ only in their return types. (Return types do not provide sufficient information in all cases for Java to decide which method to use.) Of course, overloaded methods may differ in their return types, too. When an overloaded method is called, the version of the method whose parameters match the arguments is executed.

Here is a simple example that illustrates method overloading:

```java
// Demonstrate method overloading.
class Overload {
    void ovlDemo() {
        System.out.println("No parameters");
    }
    // Overload ovlDemo for one integer parameter.
    void ovlDemo(int a) {
        System.out.println("One parameter: "+ a);
    }
    // Overload ovlDemo for two integer parameters.
    int ovlDemo(int a, int b) {
        System.out.println("Two parameters: "+ a + " "+ b);
        return a + b;
    }
    // Overload ovlDemo for two double parameters.
    double ovlDemo(double a, double b) {
        System.out.println("Two double parameters: "+ a + " "+ b);
        return a + b;
    }
}
class OverloadDemo {
    public static void main(String args[]) {
        Overload ob = new Overload();
        int resI;
        double resD;
        // call all versions of ovlDemo()
        ob.ovlDemo();
        System.out.println();
    }
}
```
ob.ovlDemo(2);
System.out.println();
resI = ob.ovlDemo(4, 6);
System.out.println("Result of ob.ovlDemo(4, 6): " + resI);
System.out.println();
resD = ob.ovlDemo(1.1, 2.32);
System.out.println("Result of ob.ovlDemo(1.1, 2.32): " + resD);
}
}

This program generates the following output:
No parameters
One parameter: 2
Two parameters: 4 6
Result of ob.ovlDemo(4, 6): 10
Two double parameters: 1.1 2.32
Result of ob.ovlDemo(1.1, 2.32): 3.42

As you can see, ovlDemo( ) is overloaded four times. The first version takes no parameters, the second takes one integer parameter, the third takes two integer parameters, and the fourth takes two double parameters. Notice that the first two versions of ovlDemo( ) return void, and the second two return a value.

Polymorphism
- Polymorphism means many (poly) shapes (morph)
- In Java, polymorphism refers to the fact that you can have multiple methods with the same name in the same class
- There are two kinds of polymorphism:
  - Overloading
    - Two or more methods with different signatures
  - Overriding
    - Replacing an inherited method with another having the same signature

Overloading

class Test {
    public static void main(String args[]) {
        myPrint(5);
        myPrint(5.0);
    }

    static void myPrint(int i) {
        System.out.println("int i = " + i);
    }
}
static void myPrint(double d) { // same name, different parameters
    System.out.println("double d = " + d);
}

int i = 5
double d = 5.0

Why overload a method?
- So you can use the same names for methods that do essentially the same thing
  - Example: println(int), println(double), println(boolean), println(String), etc.
- So you can supply defaults for the parameters:
  - int increment(int amount) {
      count = count + amount;
      return count;
  }
  - int increment() {
      return increment(1);
  }
- Notice that one method can call another of the same name
- So you can supply additional information:
  - void printResults() {
      System.out.println("total = " + total + ", average = " + average);
  }
  - void printResult(String message) {
      System.out.println(message + ": ");
      printResults();
  }

Legal assignments

class Test {
    public static void main(String args[]) {
        double d;
        int i;
        d = 5;              // legal
        i = 3.5;             // illegal
        i = (int) 3.5;       // legal
    }
}
- Widening is legal
- Narrowing is illegal
Legal method calls:

class Test {
    public static void main(String args[]) {
        myPrint(5);
    }

    static void myPrint(double d) {
        System.out.println(d);
    }
}

- Legal because parameter transmission is equivalent to assignment
- myPrint(5) is like double d = 5; System.out.println(d);

Illegal method calls:

class Test {
    public static void main(String args[]) {
        myPrint(5.0);
    }

    static void myPrint(int i) {
        System.out.println(i);
    }
}

- Illegal because parameter transmission is equivalent to assignment
- myPrint(5.0) is like int i = 5.0; System.out.println(i);