Set 19: Overloading

Skill 19.01: Explain the purpose of overloading
Skill 19.02: Apply overloading to a constructor
Skill 19.03: Apply overloading to methods

Skill 19.01: Explain the purpose of overloading

Skill 19.01 Concepts

Overloading in Java is the ability to create different objects from the same class depending on the types of parameters that are provided.

Overloading can also be applied to methods. Method overloading in Java is the ability to create multiple methods of the same name, but with different parameters. The main advantage of overloading is cleanliness of code.

To understand the purpose of overloading in Java let's consider the following implementation of a class *Box* with only one constructor taking three arguments.

Simple Box class

```java
class Box{

    double width, height,depth;

    // constructor used when all dimensions specified
    Box(double w, double h, double d){

        width = w;
        height = h;
        depth = d;
    }

    // compute and return volume
    double volume(){

        return width * height * depth;
    }
}
```

As we can see, the *Box* constructor requires three parameters. This means that all declarations of *Box* objects must pass three arguments to the *Box* constructor. For example, the following statement is currently invalid,

```java
Box ob = new Box();
```

Since *Box* requires three arguments, it's an error to call it without them. Suppose we simply wanted a box object without initial dimensions or want to initialize a cube by specifying only one value that would be used for all three dimensions. From the above implementation of Box class these options are not available to us.
These types of problems of different ways of initializing an object can be solved by constructor overloading. Below is the improved version of class Box with constructor overloading.

Box class with constructor overloading

```
class Box {

    double width, height, depth;

    // constructor used when all dimensions specified
    Box(double w, double h, double d) {

        width = w;
        height = h;
        depth = d;

    }

    // constructor used for a cube
    Box(double len) {

        width = height = depth = len;
    }

    // constructor used when no dimensions are specified
    Box() {

        width = height = depth = 0;
    }

    // compute and return volume
    double getVolume() {

        return width * height * depth;

    }

}
```

Now, let's consider some calls to the Box class from a driver class,

BoxMaker

```
public class BoxMaker {

    public static void main(String args[]) {

        // create boxes using the various constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;

        // get volume of first box
        vol = mybox1.getVolume();
        System.out.println(" Volume of mybox1 is " + vol);

        // get volume of second box
        vol = mybox2.getVolume();
        System.out.println(" Volume of mybox2 is " + vol);

        // get volume of cube
        vol = mycube.getVolume();
        System.out.println(" Volume of mycube is " + vol);

    }
}
```

Output

```
Volume of mybox1 is 3000.0
Volume of mybox2 is 0.0
Volume of mycube is 343.0
```

Skill 19.01: Exercise 1

Skill 19.02: Apply overloading to a constructor

Skill 19.02 Concepts

SumNums with constructor overloading

```
public class SumNums {

    public SumNums() {

        System.out.println("Nothing to sum!");

    }

    public SumNums(int a, int b) {

        System.out.println(a + b);

    }

    public SumNums(int a, int b, int c) {

        System.out.println(a + b + c);

    }

    public SumNums(double a, double b) {

        System.out.println(a + b);

    }

}
```

When you create an object from this class with overloaded constructors, java will match the parameters with the appropriate constructor.

SumNumsDriver	Output

`public class SumNumsDriver {`
 	
`    public static void main(String args[]) {`
 	
`        SumNums s1 = new SumNums();`
`        SumNums s2 = new SumNums(1, 2);`
`        SumNums s3 = new SumNums(1, 2, 3);`
`        SumNums s4 = new SumNums(1.0, 2.0);`
`    }`
`}`
	`Nothing to sum`
	`3`
	`6`
	`3.0`

Skill 19.02: Exercise 1

Skill 19.03: Apply overloading to methods

Skill 19.03 Concepts

Overloaded methods are differentiated by the number and the type of arguments passed into the method. For example, Suppose that you have a class that can use calligraphy to draw various types of data (strings, integers, and so on). It would be cumbersome to use a new name for each method—for example, drawString, drawInteger, drawFloat, and so on.

In the Java programming language, you can use the same name for all the drawing methods but pass a different argument list to each method. Thus, the data drawing class might declare four methods named draw, each of which has a different parameter list.

DataArtist class with method overloading

```
public class DataArtist {

    public void draw(String s) {

        // implementation not shown

    }

    public void draw(int i) {

        // implementation not shown

    }

    public void draw(double d) {

        // implementation not shown

    }

    public void draw(int i, double d) {

        // implementation not shown

    }

}
```

In the code sample above, *draw(String s)* and *draw(int i)* are distinct and unique methods because they require different argument types.

You cannot declare more than one method with the same name and the same number and type of arguments, because the compiler cannot tell them apart. For example, the code below would throw an error,

SumNums class with overloading error

```
public class SumNums {

    public int getSum(int a, int b) {

        return a + b;

    }

    public double getSum(int a, int b) {

        return (double) a + b;

    }
}
```

Fixing the error requires that we change the data types of the parameters. When we call the overloaded method, java will look for the method that matches the parameters.

SumNums

```
public class SumNums{

    // accepts int data types
    public int getSum(int a, int b) {

        int sum = a + b;
        return sum;
    }

    // accepts double data types
    public int getSum(double a, double b) {

        double sum = a + b;
        return (int) sum;
    }
}
```

SumNumsDiver

```
public class SumNumsDriver {

    public static void main(String args[]) {

        SumNums s = new SumNums();
        System.out.println(s.getSum(1, 2));
        System.out.println(s.getSum(2.0, 2.0));
    }
}
```

Output

```
3
4
```

Skill 19.03: Exercise 1