PART 3

You have been using static methods in your programs before and methods are the building blocks of procedural programming.

In this part of the tutorial we will introduce non-static methods, i.e. methods that are inside non-static classes.

The idea of having methods in a non-static class can be confusing at first since we introduced non-static classes as a way defining new, more elaborate data types called objects. The ability to put methods inside these classes means that objects are more than just serving as a new data type. They are much more powerful.

We will start by explaining why non-static methods are useful.

In principle, non-static methods do not give programmers any extra power because anything one can do using non-static methods can be done using static methods. Then why do we need non-static methods?

The heart of object-oriented programming is the ability to reuse someone else's code without even looking at the code itself. This is extremely important when developing big projects. But code reusability would not mean much if we were not able to reuse someone else's methods.

The ability to put methods in non-static classes means that we can use other people's methods without knowing the details of how it is implemented. With non-static methods, object oriented programming gets interesting.

All this will be better understood with an example.

Let's create a non-static class called Box which has three fields: int height, int width, int depth.

```
class Box
{
        int height;
        int width;
        int depth;

        Box(int h, int w, int d)
        {
                height = h;
                width = w;
                depth = d;
        }

}
```

Now that we have defined this Box class, we can create Box objects of any dimension.

```
class Application
{
        static void main(String args[])
        {
                Box b1 = new Box(3,4,5);
                Box b2 = new Box(2,1,4);
                Box b3 = new Box(2,2,2);
        }
}
```

Suppose we needed to calculate the surface area and the volume of a number of boxes. For this purpose we can define two methods which take a Box as a parameter and return the Box's surface area and volume respectively.

```
class Application
{
  static void main(String[] args)
  {
    Box b1 = new Box (2,3,4);
    Box b2 = new Box (1,3,2);
    System.out.println("The surface area of the first box is " + calculateSurfaceArea(b1) + ".");
    System.out.println("The volume of the second box is " + calculateVolume(b2) + ".");
  }

  static int calculateSurfaceArea(Box b)
  {
    return 2 * b.height * b.width + 2 * b.height * b.depth + 2 * b.width * b.depth;
  }

  static int calculateVolume(Box b)
  {
    return b.height * b.width * b.depth;

  }
}
```

The surface area and volume methods are very natural methods that any Box object should come equipped with. In other words, any person who has downloaded the Box class should not have to write the code for these methods. Somehow, these methods should reside in the Box object and any person should be able to use them without even knowing the details of the code inside them.

This is accomplished by putting these methods inside the Box class. When we do this, we will get rid of the "static" keyword and these methods will become non-static methods.

```
class Box
{
        int height;
        int width;
        int depth;

        Box(int h, int w, int d)
        {
                height = h;
                width = w;
                depth = d;
        }

        int calculateSurfaceArea()
        {
                return 2 * height * width + 2 * height * depth + 2 * width * depth;
        }

        int calculateVolume()
        {
                return height * width * depth;
        }
}
```

The careful reader will notice a difference in the calculateSurfaceArea and the calculateVolume non-static methods. For example let's look at the calculateSurfaceArea method.

```
static int calculateSurfaceArea(Box b)
{
        return 2 * b.height * b.width +
        2 * b.height * b.depth +
        2 * b.width * b.depth;
}
```

```
int calculateSurfaceArea()
{
        return 2 * height * width
        + 2 * height * depth
        + 2 * width * depth;
}
```

To understand why such a difference exists, it is a good idea to look at how we call the non-static methods in our static class.

```
class Application
{
        static void main(String[] args)
        {
            Box b1 = new Box (2,3,4);
            Box b2 = new Box (1,3,2);
            System.out.println("The surface area of the first box is " + b1.calculateSurfaceArea() + ".");
            System.out.println("The volume of the second box is " + b2.calculateVolume() + ".");
        }
}
```

In this code, we create two Box objects b1 and b2. Each of these objects has its own fields: int height, int width, int depth. On top of these, they come with two methods: calculateSurfaceArea and calculateVolume. Recall that we use the dot operator to access these objects' fields. Similarly, we use the dot operator to access the objects' methods.

Now let's go back to the non-static vs. static comparison.

```
static int calculateSurfaceArea(Box b)
{
        return 2 * b.height * b.width +
        2 * b.height * b.depth +
        2 * b.width * b.depth;
}
```

```
int calculateSurfaceArea()
{
        return 2 * height * width
        + 2 * height * depth
        + 2 * width * depth;
}
```

In the static version, we have to give the object as a parameter since a static method does not belong to a particular object. In the non-static version, we do not need to do this since the method is attached to an object already.

Also note that in the static version we have to access the fields of the object using the dot operator. In the non-static version, we directly use the names of the fields because the non-static method has direct access to the fields of the object that it belongs to.

```
class Box{

        int height;
        int width;
        int depth;

        Box(int h, int w, int d){
                height = h;
                width = w;
                depth = d;
        }
}
```

```
class Application
{
        static void main(String[] args)
        {
                Box b1 = new Box (2,3,4);
                Box b2 = new Box (1,3,2);
                System.out.println(
                calculateSurfaceArea(b1));
                System.out.println(
                calculateVolume(b2));
        }
        static int calculateSurfaceArea(Box b)
        {

                return 2 * b.height * b.width + 2 *
                b.height * b.depth + 2 * b.width *
                b.depth;
        }

        static int calculateVolume(Box b)
        {
                return b.height * b.width * b.depth;
        }
}
```

```
class Box{
        int height;
        int width;
        int depth;

        Box(int h, int w, int d)
        {
                height = h;
                width = w;
                depth = d;
        }

        int calculateSurfaceArea()
        {
                return 2 * height * width + 2 * height
                * depth + 2 * width * depth;
        }

        int calculateVolume()
        {
                return height * width * depth;
        }
}
```

```
class Application
{
        static void main(String[] args)
        {

                Box b1 = new Box (2,3,4);
                Box b2 = new Box (1,3,2);
                System.out.println(
                b1.calculateSurfaceArea());
                System.out.println(
                b2.calculateVolume());
        }
}
```

Now let's demonstrate how this transition may be useful to programmers all around the world.

Suppose you are the creator of the Box class and you share this class with everyone through the internet. Along with your program, you give a documentation. In the documentation, you provide the following information.

Constructor

Box(int h, int w, int d)

Creates a box with height h, width w and depth d.

Methods

1) int calculateSurfaceArea()

Returns the surface area of the box.

2) int calculateVolume()

Returns the volume of the box.

Any person who has downloaded your class can now use it without even looking at one line of the code. S/he can create Box objects of any dimension and can calculate the volume or surface area very easily.

```
class Application
{
        static void main(String args[])
        {
                Box b = new Box(3,4,2);
                System.out.println("Surface area of the box is " + b.calculateSurfaceArea() + ".");

        }
}
```

Conveniently, people using this object do not have to know the formula for calculating the surface area of a box.

On the internet, one can find many many classes along with their documentation. Some of these classes contain complicated code. But we do not need to know the code to use this class and its methods. This is how one creates big projects: by using previously created classes and not worrying about how it was implemented.

Non-static methods can be used for many purposes such as

- Altering the state of the object
- Provide some information about the object
- Make a calculation using the fields of the object

Certainly the uses of these methods are not limited to above. With what Java allows you with object oriented programming, your imagination is the limit.

Now let's see an example which demonstrates some of the different uses of non-static methods.

```
class Beer{

	int amount;

	Beer(int a){
		amount = a;
	}

	void drink(int d){
		amount = amount – d;
	}

	void chuck(){
		amount = 0;
	}

	void print_amount(){
		System.out.println(amount);
	}

	boolean is_empty(){
		if(amount == 0) return true;
		else return false;
	}

}
```

```
class Application
{

		static void main(String[] args)
		{

		  Beer b = new Beer(15);
		  b.print_amount();
		  b.drink(5);
		  if(b.is_empty())
				System.out.println(“No more beer.”);
		  b.print_amount();
		  b.chuck();
		  b.print_amount();
		  if(b.is_empty())
				System.out.println(“zzzzz”);

		}

}
```

Review of Concepts

- Apart from the fields and constructors, an object could also include methods. These methods could be used for a variety of reasons:
  - Make a sophisticated calculation using the values of the fields of the object
  - Print out some information about the object
  - Modify the values of the fields of the object

- Revisiting the reusability: With the addition of methods, the sharing of code concept should become more clear.
  - Let's say you implement the Box class. The class has fields, constructors and methods.
  - A friend of yours, who is in need of a Box object in his/her application, may use this Box class that you have written, without even knowing the formula to calculate the surface area.

Review of Syntax

- In order to add a method to the class declaration:

```
returnType methodName(parameters)
{
        // your code goes here...
}
```

Exercise

Write a simple class to represent elevators. An elevator has a current floor, number of floors, current number of passengers and maximum number of passengers.

An elevator:

Is constructed by specifying:

- the total number of floors in the building
- the maximum elevator capacity
- that the elevator initially doesn't have any passengers
- and that the elevator is initially located on the bottom floor.

Can move one floor up if it is not on the top floor.

Can move one floor down if it is not on the bottom floor.

Can accept a certain number of passengers (up to its maximum capacity).

Can drop off a certain number of passengers (no more than it actually has).

Can tell us which floor it is on.

Solution

```
class Elevator{

        int currentFloor;
        int numFloors;
        int currentPassengers;
        int maxPassengers;

        Elevator(int maxFloors, int capacity){

                numFloors = maxFloors;
                maxPassengers = capacity;
                currentFloor = 1;
                currentPassengers = 0;

        }

        void moveUp(){

                if(currentFloor < numFloors) currentFloor++;

        }

        void moveDown(){

                if(currentFloor > 0 ) currentFloor--;

        }
```

```
	boolean acceptPassengers(int num){

		boolean result = true;
		int difference = maxPassengers – currentPassengers;
		if(difference > 0 && num > 0){
			if(num < difference) currentPassengers += num;
			else currentPassengers += difference;
		}
		else result = false;
		return result;

	}

	boolean dropOffPassengers(int num){

		boolean result = true;
		if(num <= currentPassengers && num > 0) currentPassengers -= num;
		else{
			if(num > currentPassengers) currentPassengers = 0;
			else result = false;
		}
		return result;

	int getCurrentFloor(){
		return currentFloor;
	}

} // end of class Elevator
```