

# Java

Introducing Classes

# Class

- The class is at the core of Java
- It is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object
- As such, the class forms the basis for object-oriented programming in Java
- Any concept you wish to implement in a Java program must be encapsulated within a class

# class

- A class is a template that defines the form of an object. It specifies both the data and the code that will operate on that data
- Java uses a class specification to construct objects
- Objects are instances of a class. Thus, a class is essentially a set of plans that specify how to build an object

# class

- A class is created by using the keyword class. A simplified general form of a class definition is shown here

```
class classname{  
    type instance variable1;  
    type instance variable2;  
    type methodname(parameter-list){  
        //body of method  
    }  
}
```

# class

- The data, or variables, defined within a class are called instance variables.
- The code is contained within methods
- Collectively, the methods and variables defined within a class are called members of the class
- In most classes, the instance variables are acted upon and accessed by the methods defined for that class. Thus, as a general rule, it is the methods that determine how a class' data can be used

# A simple class

- Let's begin our study of the class with a simple example. Here is a class called Box that defines three instance variables: width, height, and depth. Currently, Box does not contain any methods

```
class Box{  
    double width;  
    double height;  
    double depth;  
}
```

# A simple class

- To actually create a Box object, you will use a statement like the following:

```
Box mybox = new Box(); // create a Box object called mybox
```

After this statement executes, mybox will refer to an instance of Box. Thus, it will have “physical” reality

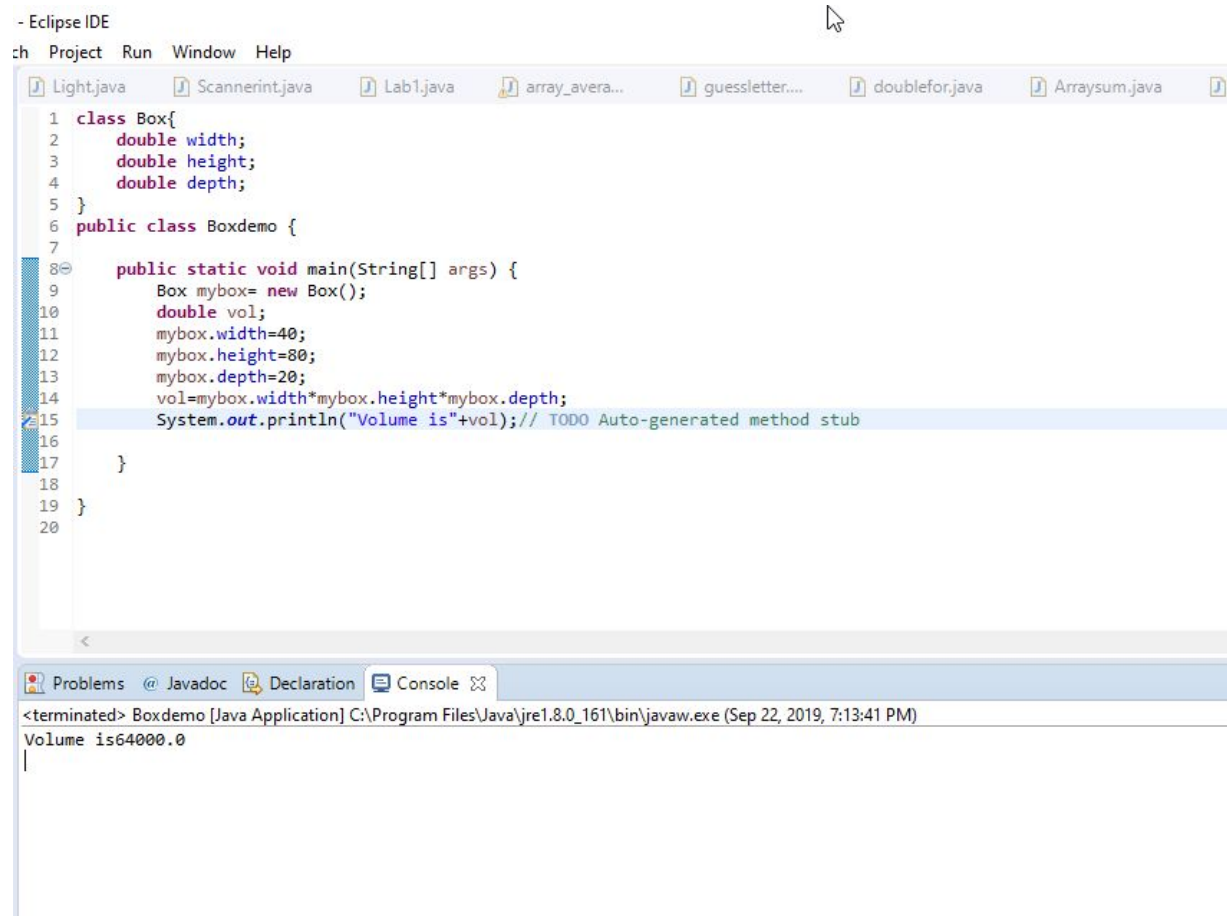
# A simple class

- each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class. Thus, every Box object will contain its own copies of the instance variables width, height, and depth
- To access these variables, you will use the dot (.) operator. The dot operator links the name of the object with the name of an instance variable. For example, to assign the width variable of mybox the value 100, you would use the following statement:

```
mybox.width = 100;
```



# Example

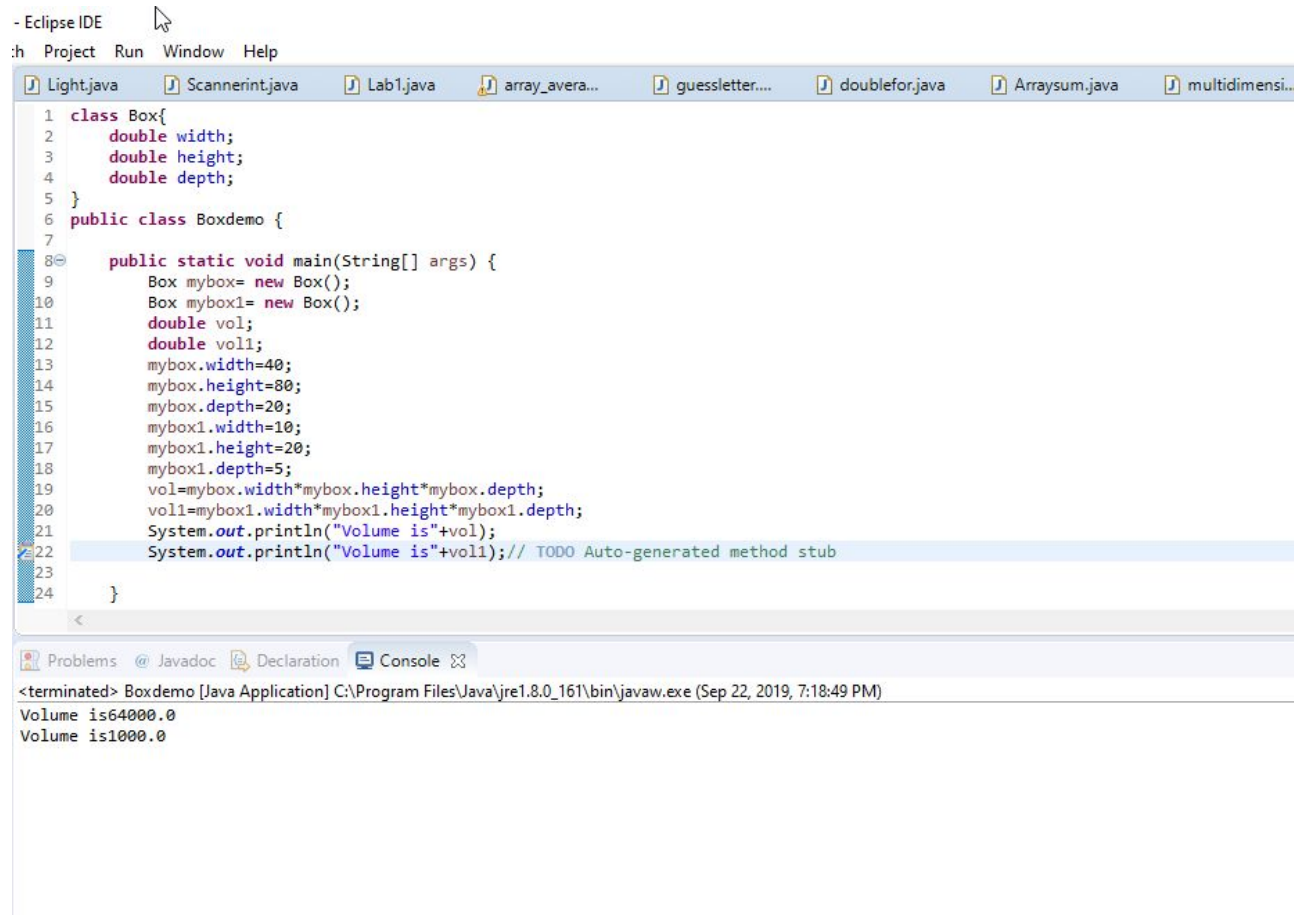


The screenshot shows the Eclipse IDE interface. The top menu bar includes 'File', 'Edit', 'Project', 'Run', 'Window', and 'Help'. The package explorer on the left shows a project named 'Light.java'. The editor window displays the following Java code:

```
1 class Box{
2     double width;
3     double height;
4     double depth;
5 }
6 public class Boxdemo {
7
8     public static void main(String[] args) {
9         Box mybox= new Box();
10        double vol;
11        mybox.width=40;
12        mybox.height=80;
13        mybox.depth=20;
14        vol=mybox.width*mybox.height*mybox.depth;
15        System.out.println("Volume is"+vol);// TODO Auto-generated method stub
16
17    }
18 }
19
20
```

The bottom of the IDE features a 'Console' tab. It displays the output of the program: '<terminated> Boxdemo [Java Application] C:\Program Files\Java\jre1.8.0\_161\bin\javaw.exe (Sep 22, 2019, 7:13:41 PM)' followed by 'Volume is64000.0'.

# Example



The screenshot shows the Eclipse IDE interface. The top menu bar includes 'File', 'Edit', 'Project', 'Run', 'Window', and 'Help'. The project explorer on the left lists several Java files: 'Light.java', 'Scannerint.java', 'Lab1.java', 'array\_avera...', 'guessletter....', 'doublefor.java', 'Arraysun.java', and 'multidimensi...'. The main editor window displays the source code for 'Boxdemo.java'. The code defines a 'Box' class with attributes 'width', 'height', and 'depth', and a 'Boxdemo' class with a 'main' method. The 'main' method creates two 'Box' objects, 'mybox' and 'mybox1', sets their dimensions, calculates their volumes, and prints the results. The console at the bottom shows the output of the program, indicating that it has terminated successfully and displaying the calculated volumes for both boxes.

```
1 class Box{
2     double width;
3     double height;
4     double depth;
5 }
6 public class Boxdemo {
7
8     public static void main(String[] args) {
9         Box mybox= new Box();
10        Box mybox1= new Box();
11        double vol;
12        double vol1;
13        mybox.width=40;
14        mybox.height=80;
15        mybox.depth=20;
16        mybox1.width=10;
17        mybox1.height=20;
18        mybox1.depth=5;
19        vol=mybox.width*mybox.height*mybox.depth;
20        vol1=mybox1.width*mybox1.height*mybox1.depth;
21        System.out.println("Volume is"+vol);
22        System.out.println("Volume is"+vol1);// TODO Auto-generated method stub
23
24    }
```

<terminated> Boxdemo [Java Application] C:\Program Files\Java\jre1.8.0\_161\bin\javaw.exe (Sep 22, 2019, 7:18:49 PM)  
Volume is64000.0  
Volume is1000.0

# Declaring objects

- As just explained, when you create a class, you are creating a new data type
- However, obtaining objects of a class is a two-step process. First, you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object
- Second, you must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the new operator. The new operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it

# Declaring Objects

- In the preceding sample programs, a line similar to the following is used to

declare an object of type Box:

```
Box mybox = new Box();
```

This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

```
Box mybox; // declare reference to object
```

```
mybox = new Box(); // allocate a Box object
```

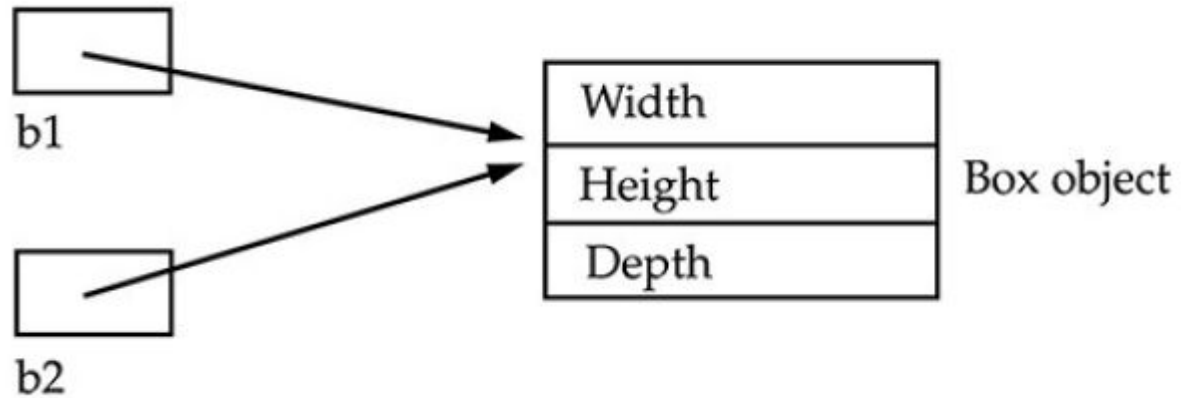
# Assigning object reference variables

```
Box b1 = new Box();
```

```
Box b2 = b1;
```

after this fragment executes, b1 and b2 will both refer to the same object. The assignment of b1 to b2 did not allocate any memory or copy any part of the original object. It simply makes b2 refer to the same object as does b1

# Assigning object reference variables



# Introducing methods

This is the general form of a method:

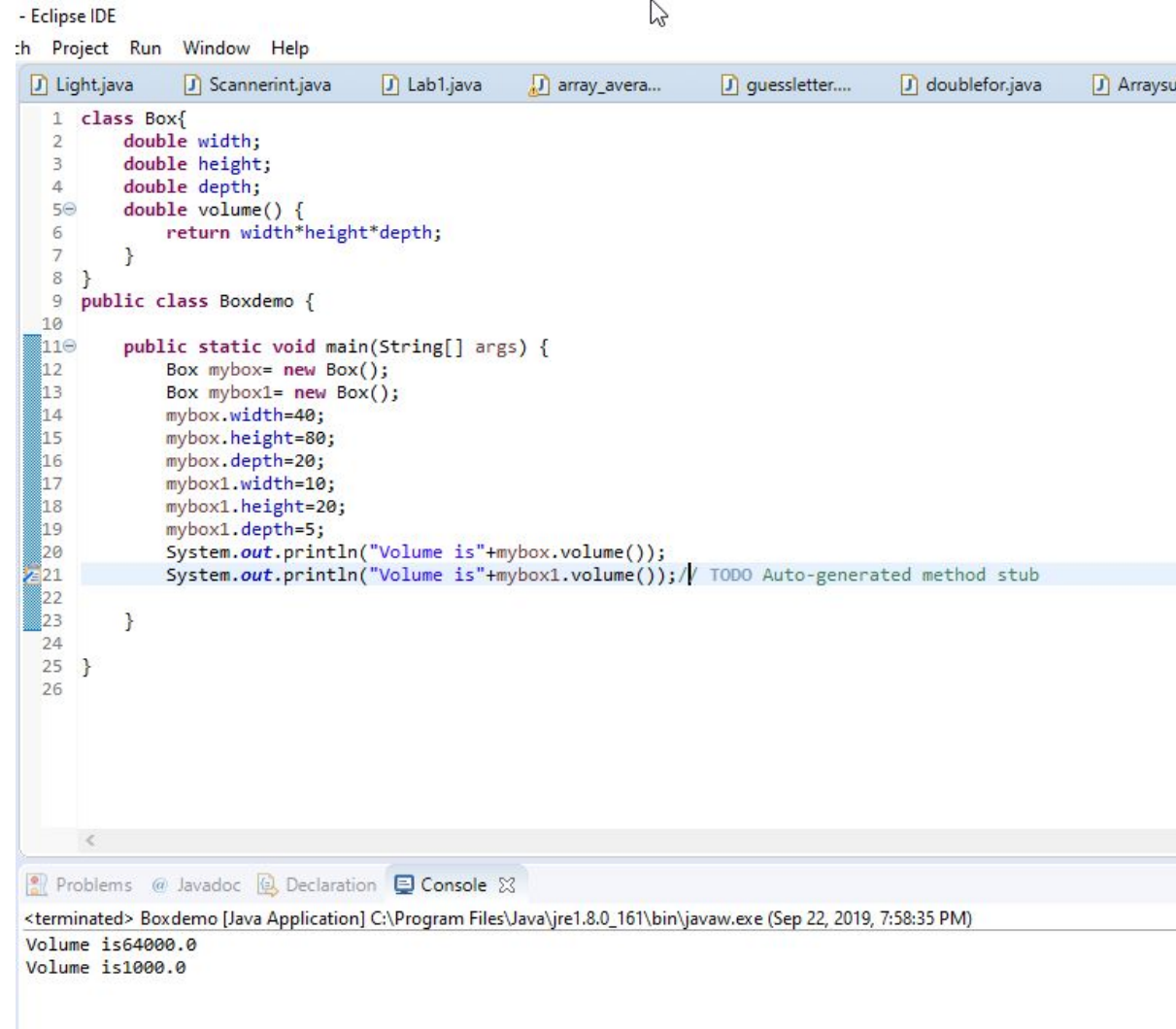
```
type name(parameter-list) {  
    // body of method  
}
```

Methods that have a return type other than void return a value to the calling routine using the following form of the return statement:

```
    return value;
```

Here, value is the value returned.

# Example



The screenshot shows the Eclipse IDE interface. The top menu bar includes 'File', 'Edit', 'Project', 'Run', 'Window', and 'Help'. The top toolbar shows icons for 'File', 'Edit', 'Project', 'Run', 'Window', and 'Help'. The top tab bar displays several open files: 'Light.java', 'Scannerint.java', 'Lab1.java', 'array\_avera...', 'guessletter....', 'doublefor.java', and 'Arraysu...'. The main editor window shows the following Java code:

```
1 class Box{
2     double width;
3     double height;
4     double depth;
5     double volume() {
6         return width*height*depth;
7     }
8 }
9 public class Boxdemo {
10
11     public static void main(String[] args) {
12         Box mybox= new Box();
13         Box mybox1= new Box();
14         mybox.width=40;
15         mybox.height=80;
16         mybox.depth=20;
17         mybox1.width=10;
18         mybox1.height=20;
19         mybox1.depth=5;
20         System.out.println("Volume is"+mybox.volume());
21         System.out.println("Volume is"+mybox1.volume());// TODO Auto-generated method stub
22
23     }
24
25 }
26
```

The bottom of the IDE shows the 'Console' tab with the following output:

```
<terminated> Boxdemo [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Sep 22, 2019, 7:58:35 PM)
Volume is64000.0
Volume is1000.0
```



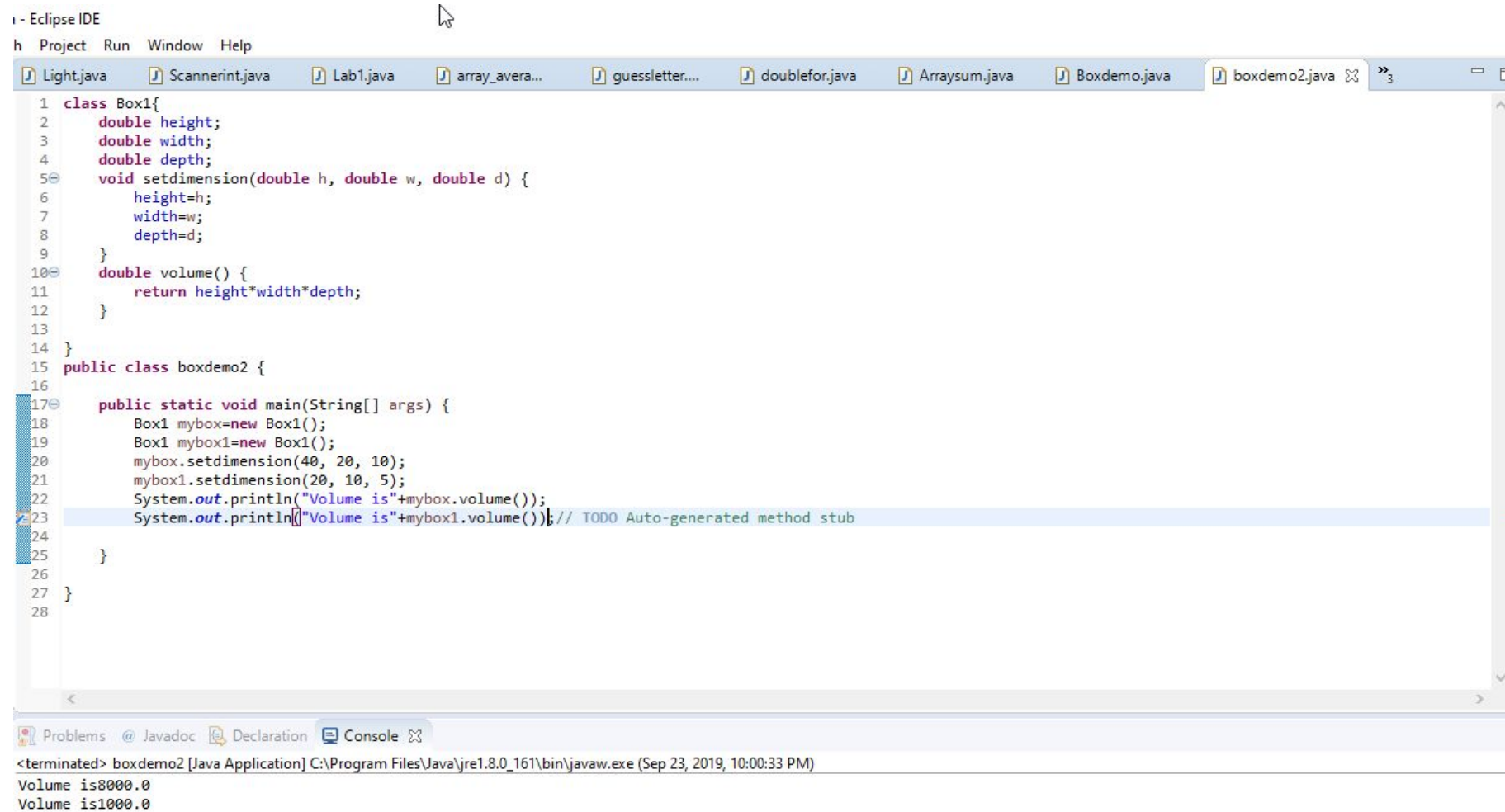
# Adding a method that takes parameter

- While some methods don't need parameters, most do
- Parameters allow a method to be generalized
- A parameterized method can operate on a variety of data and/or be used in a number of slightly different situations

Example:

```
int square(int x){  
    return x*x;  
}
```

# Example



```
1 class Box1{
2     double height;
3     double width;
4     double depth;
5     void setdimension(double h, double w, double d) {
6         height=h;
7         width=w;
8         depth=d;
9     }
10    double volume() {
11        return height*width*depth;
12    }
13 }
14
15 public class boxdemo2 {
16
17     public static void main(String[] args) {
18         Box1 mybox=new Box1();
19         Box1 mybox1=new Box1();
20         mybox.setdimension(40, 20, 10);
21         mybox1.setdimension(20, 10, 5);
22         System.out.println("Volume is"+mybox.volume());
23         System.out.println("Volume is"+mybox1.volume()); // TODO Auto-generated method stub
24     }
25 }
26
27 }
28
```

<terminated> boxdemo2 [Java Application] C:\Program Files\Java\jre1.8.0\_161\bin\javaw.exe (Sep 23, 2019, 10:00:33 PM)

Volume is8000.0  
Volume is1000.0

# Constructors

- It can be tedious to initialize all of the variables in a class each time an instance is created
- Even when you add convenience methods like `setDim( )`, it would be simpler and more concise to have all of the setup done at the time the object is first created
- This automatic initialization is performed through the use of a constructor.

# Constructors

- A constructor initializes an object immediately upon creation
- It has the same name as the class in which it resides and is syntactically similar to a method
- Once defined, the constructor is automatically called when the object is created, before the new operator completes
- Constructors have no return method

# Example

Example of a Java application in Eclipse IDE.

```
1 class Box3{
2     double height;
3     double width;
4     double depth;
5     Box3(){
6         System.out.println("A new object is created");
7     }
8     Box3(double h, double w, double d){
9         this.height=h;
10        this.width=w;
11        this.depth=d;
12    }
13    double volume() {
14        return height*width*depth;
15    }
16 }
17
18 public class Boxdemo3 {
19
20     public static void main(String[] args) {
21         Box3 mybox=new Box3();
22         mybox=new Box3(40,20,10);
23         Box3 mybox2=new Box3();
24         mybox2=new Box3(20,10,5);
25         System.out.println("Volume is"+mybox.volume());
26         System.out.println("Volume is"+mybox2.volume());// TODO Auto-generated method stub
27     }
28 }
29
30 }
31
```

Console Output:

```
<terminated> Boxdemo3 [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Sep 23, 2019, 10:33:28 PM)
A new object is created
A new object is created
Volume is8000.0
Volume is1000.0
```

# this keyword

- Sometimes a method will need to refer to the object that invoked it
- To allow this, Java defines the this keyword. this can be used inside any method to refer to the current object
- this is always a reference to the object on which the method was invoked. You can use this anywhere a reference to an object of the current class' type is permitted.

# Garbage Collection

- Since objects are dynamically allocated by using the new operator, you might be wondering how such objects are destroyed and their memory released for later reallocation
- Java handles deallocation automatically, it is called garbage collection
- It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no need to explicitly destroy objects

# A Stack class

- A stack stores data using first-in, last-out ordering
- A stack is like a stack of plates on a table—the first plate put down on the table is the last plate to be used
- Stacks are controlled through two operations traditionally called push and pop
- To put an item on top of the stack, you will use push. To take an item off the stack, you will use pop



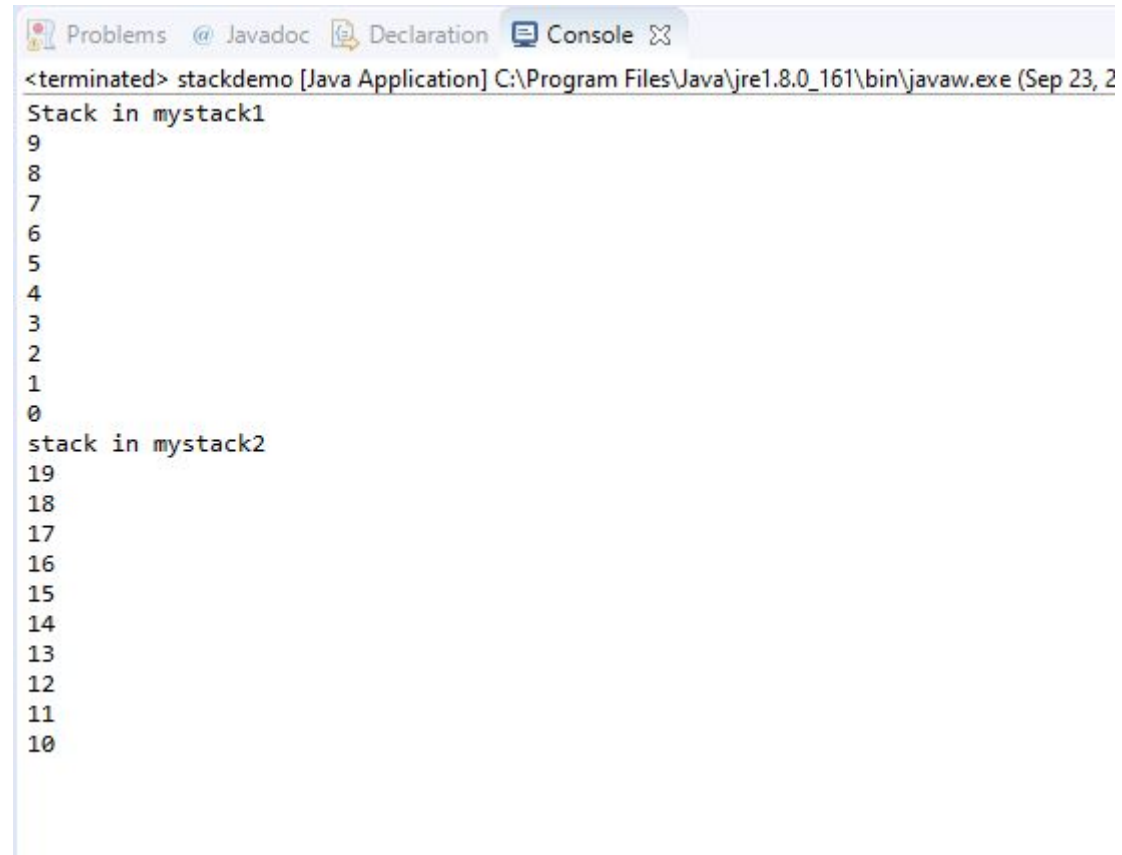
# A Stack Class

```
1
2 public class stack {
3     int stck[]=new int[10];
4     int tos;
5     stack() {
6         tos=-1;
7     }
8     void push(int item) {
9         if(tos==9)
10             System.out.println("Stack is full");
11         else
12             stck[++tos]=item;
13     }
14     int pop() {
15         if (tos<0) {
16             System.out.println("Stack is underflowed");
17             return 0;
18         }
19         else
20             return stck[tos--];
21     }
22 }
23
```

# A stack class

```
1
2 public class stackdemo {
3
4     public static void main(String[] args) {
5         stack mystack= new stack();
6         stack mystack1=new stack();
7         for(int i=0; i<10; i++)
8             mystack.push(i);
9         for(int i=10; i<20;i++)
10            mystack1.push(i);
11         System.out.println("Stack in mystack1");
12         for(int i=0; i<10; i++)
13             System.out.println(mystack.pop());// TODO Auto-generated method stub
14         System.out.println("stack in mystack2");
15         for(int i=10; i<20;i++)
16             System.out.println(mystack1.pop());
17     }
18 }
19
20
```

# A stack class output



```
Problems  @ Javadoc  Declaration  Console  ⌵
<terminated> stackdemo [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Sep 23, 2
Stack in mystack1
9
8
7
6
5
4
3
2
1
0
stack in mystack2
19
18
17
16
15
14
13
12
11
10
```

# Example

Java IDE

File Project Run Window Help

Light.java Scannerint.java Lab1.java array\_avera... guessletter.... doublefor.java birthda...

```
1 class powr{
2     int number;
3     int power;
4     int result;
5     powr(int n, int p){
6         this.number=n;
7         this.power=p;
8     }
9
10    int exponent() {
11        result=1;
12        int i;
13        for(i=1;i<=power; i++) {
14            result=result*number;
15        }
16        return result;
17    }
18 }
19
20 public class pwr {
21
22    public static void main(String[] args) {
23        powr p= new powr(5,3);
24        powr p1=new powr(2,5);
25        powr p2=new powr(7,2);
26        System.out.println(+p.exponent());
27        System.out.println(+p1.exponent());
28        System.out.println(+p2.exponent());// TODO Auto-generated method stub
29
30    }
31 }
32
33
```

Problems @ Javadoc Declaration Console

<terminated> pwr [Java Application] C:\Program Files\Java\jre1.8.0\_161\bin\javaw.exe (Sep 29, 2019, 8:09:18 PM)

125  
32  
49