Programming in Java

Classes and Methods

Basics of class

- A class is that it defines a new data type. Once defined, this new type can be used to create objects of that type.
- Thus, a class is a template for an object, and an object is an instance of a class. Because an object is an instance of a class.

General form of class

```
class classname
    {
            type instance-variable1;
            type instance-variable2;
            // ...
            type instance-variableN;
            type methodname1(parameter-list)
                  {
            // body of method
                }
            // ...
            type methodnameN(parameter-list) {
            // body of method
            }
}
```

General form of class

- Instance variables
- Methods contain code
- Members of class
- Class declaration and the implementation of the methods are stored in the same place and not defined separately.
- Having specification, declaration, and implementation all in one place makes for code that is easier to maintain.

A Simple Class

- ```
  class Box
   {
        double width;
        double height;
        double depth;
   }
  ```
- How it is defining new data type box.
- To actually create a Box object, you will use a statement like the following:

  ```
  Box mybox = new Box();
  ```
- To access these variables the dot (.) operator is used.
- All classes of same program can be written separately and compiled.
  ```

```
class Box {
        double width;
        double height;
        double depth;
        }
class BoxDemo
 {
        public static void main(String args[])
         {
                Box mybox = new Box();
                double vol;
                mybox.width = 10;
                mybox.height = 20;
                mybox.depth = 15;
                vol = mybox.width * mybox.height * mybox.depth;
                System.out.println("Volume is " + vol);
        }
}
```

Declaring Objects

- Box mybox = new Box();
- This statement combines the two steps just described.

  Box mybox; // declare reference to object

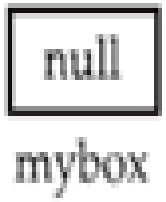
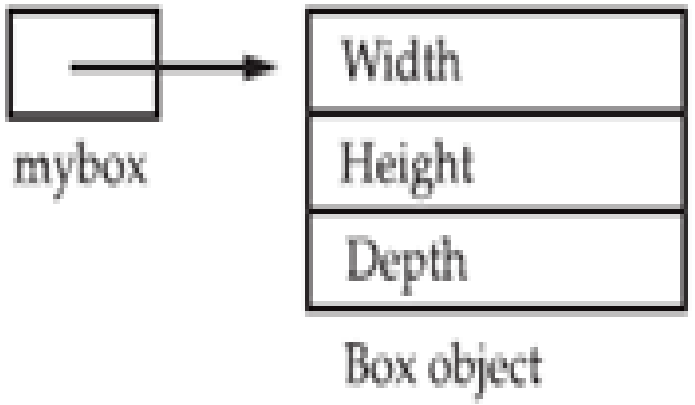
  mybox = new Box(); // allocate a Box object

- The first line declares mybox as a reference to an object of type Box. After this line executes, mybox contains the value null, which indicates that it does not yet point to an actual object. Any attempt to use mybox at this point will result in a compile-time error.

- The second line allocates an actual object and assigns a reference to it to mybox.

- Object reference is analogous to pointer but it cannot be manipulated as actual pointers.

Creating objects

Statement	Effect
Box mybox;	
mybox = new Box();	

Creating objects

- New allocates memory for an object during run time.
- The advantage of this approach is that your program can create as many or as few objects as it needs during the execution of your program.
- It is possible that new will not be able to allocate memory for an object because insufficient memory exists. If this happens, a run-time exception will occur.
- **Assigning Object Reference Variables**

Methods

- This is the general form of a method:

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

- difference between parameters and arguments
- 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;
```

Class with methods

```
// This program includes a method inside the box class.
class Box {
double width;
double height;
double depth;
void volume()
    {
        System.out.print("Volume is ");
        System.out.println(width * height * depth);
     }
}
class BoxDemo3
{
            public static void main(String args[])
               {
            Box mybox1 = new Box();
            mybox1.width = 10;
            mybox1.height = 20;
            mybox1.depth = 15;
            mybox1.volume();
               }
}
```

Methods

- When a method uses an instance variable that is defined by its class, it does so directly, without explicit reference to an object and without use of the dot operator.
- Returning a Value
- The type of data returned by a method must be compatible with the return type specified by the method.
- The variable receiving the value returned by a method must also be compatible with the return type specified for the method.

Passing values to method

```
class Box {
        double width;
        double height;
        double depth;
        double volume()
                {
                return width * height * depth;
                 }
        void setDim(double w, double h, double d)
              {
                width = w;
                 height = h;
                 depth = d;
               }
      }
```

Passing values to method

```
class BoxDemo5
        {
        public static void main(String args[])
                {
                Box mybox1 = new Box();
                double vol;
                mybox1.setDim(10, 20, 15);
                vol = mybox1.volume();
                System.out.println("Volume is " + vol);
                }
        }
```

Using constructor for initialization

```
class Box {
        double width;
        double height;
        double depth;
        Box()
            {
                System.out.println("Constructing Box");
                width = 10;
                height = 10;
                depth = 10;
            }
        double volume()
            {
                return width * height * depth;
                }
        }
```

Using constructor for initialization

```
class BoxDemo6
        {
        public static void main(String args[])
            {
                Box mybox1 = new Box();
                double vol;
                vol = mybox1.volume();
                System.out.println("Volume is " + vol);
              }
        }
```

Parameterized constructor

- Used when we want to set values of dataitems at runtime.

```
class Box
    {
        double width;
        double height;
        double depth;
      Box(double w, double h, double d)
        {
            width = w;
            height = h;
            depth = d;
        }
      double volume()
        {
            return width * height * depth;
        }
    }
```

Parameterized constructor

```
class BoxDemo7
        {
        public static void main(String args[])
                {
                Box mybox1 = new Box(10, 20, 15);
                Box mybox2 = new Box(3, 6, 9);
                double vol;
                vol = mybox1.volume();
                System.out.println("Volume is " + vol);
                vol = mybox2.volume();
                System.out.println("Volume is " + vol);
                }
        }
```

Method Overloading

```
class OverloadDemo {
void test() {
System.out.println("No parameters");
}
void test(int a) {
System.out.println("a: " + a);
}
void test(int a, int b) {
System.out.println("a and b: " + a + " " + b);
}

double test(double a) {
System.out.println("double a: " + a);
return a*a;
}
}
```

Method Overloading

```
class Overload {
public static void main(String args[]) {
OverloadDemo ob = new OverloadDemo();
double result;
// call all versions of test()
ob.test();
ob.test(10);
ob.test(10, 20);
result = ob.test(123.25);
System.out.println("Result of ob.test(123.25): " + result);
}
}
```

Automatic type conversion in method overloading

```
// Automatic type conversions apply to overloading.
class OverloadDemo {
void test() {
System.out.println("No parameters");
}
// Overload test for two integer parameters.
void test(int a, int b) {
System.out.println("a and b: " + a + " " + b);
}
// overload test for a double parameter
void test(double a) {
System.out.println("Inside test(double) a: " + a);
}
}
```

Automatic type conversion in method overloading

```
class Overload {
public static void main(String args[])
{
OverloadDemo ob = new OverloadDemo();
int i = 88;
ob.test();
ob.test(10, 20);
ob.test(i);              // this will invoke test(double)
ob.test(123.2);      // this will invoke test(double)
}
}
```

Overloaded Constructors

```
class Box {
double width;
double height;
double depth;
Box(double w, double h, double d)
{
width = w;
height = h;
depth = d;
}
double volume() {
return width * height * depth;
}
}
```

```
/* Here, Box defines three constructors to initialize the dimensions of a box various ways. */
class Box {
double width;
double height;
double depth;
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
Box() {
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box
}
Box(double len) {
width = height = depth = len;
}
double volume() {
return width * height * depth;
}
}
```

```
class OverloadCons {
public static void main(String args[]) {
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box();
Box mycube = new Box(7);
double vol;
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
vol = mybox2.volume();
System.out.println("Volume of mybox2 is " + vol);
vol = mycube.volume();
System.out.println("Volume of mycube is " + vol);
}
}
```

Using Objects as Parameters

```
// Objects may be passed to methods.
class Test {
int a, b;
Test(int i, int j) {
a = i;
b = j;
}
// return true if o is equal to the invoking object
boolean equals(Test o) {
if(o.a == a && o.b == b) return true;
else return false;
}
}
class PassOb {
public static void main(String args[]) {
Test ob1 = new Test(100, 22);
Test ob2 = new Test(100, 22);
Test ob3 = new Test(-1, -1);
System.out.println("ob1 == ob2: " + ob1.equals(ob2));
System.out.println("ob1 == ob3: " + ob1.equals(ob3));
} }
```