

Classes, Objects and their Members

Dheeru Mundluru



```
ClientA.java ClientB.java JarHell.java my.ini new_7 new_6 Student.java BasicsDemo.java new_8
7     scores[3] = 100;
8
9     /*int[] scores = new int[] {90, 70, 80, 100};*/
10
11    //int[] scores = {90, 70, 80, 100};
12
13    System.out.println("Mid-Term 1: " + scores[0]);
14    System.out.println("Mid-Term 2: " + scores[1]);
15    System.out.println("Final: " + scores[2]);
16    System.out.println("Project: " + scores[3]);
17    System.out.println("# exams: " + scores.length);
18
19    Student[] students = new Student[3];//{new Student(), new Student(), new Student()};
20    students[0] = new Student();
21    students[1] = new Student();
22    // students[2] = new Student();
23    /*students[0].name = "John";
24    students[1].name = "Raj";
25    students[2].name = "Anita";*/
26    System.out.println("Student 1: " + students[0]);
27    System.out.println("Student 2: " + students[1]);
28    System.out.println("Student 3: " + students[2]);
29 }
```

Example 3 :

```
Static void array()
{
    Student[] students = new Student[3];
    students[0] = new Student();
    students[1] = new Student();

    System.out.println("Student 1 : " + students[0]);
    System.out.println("Student 2 : " + students[1]);
    System.out.println("Student 3 : " + students[2]);
}
```

above code will run fine and return random the value for student[0], student[1] and null for student[2].

Edit Search View Encoding Language Settings Macro Run

39. Arrays + Demo

```
7     scores[3] = 100;  
8  
9     /*int[] scores = n  
10    //int[] scores = {  
11        System.out.println("Inside arrays ...");  
12        System.out.println("Mid-Term 1: " + scores[0]);  
13        System.out.println("Mid-Term 2: " + scores[1]);  
14        System.out.println("Final: " + scores[2]);  
15        System.out.println("Project: " + scores[3]);  
16        System.out.println("# exams: " + scores.length);  
17        System.out.println();  
18  
19        Student[] students;  
20        students[0] = new Student();  
21        students[1] = new Student();  
22        // students[2] = new Student();  
23        /*students[0].name = "John";  
24        students[1].name = "Jane";  
25        students[2].name = "Mike";  
26        System.out.println("Student 1: " + students[0].name);  
27        System.out.println("Student 2: " + students[1].name);  
28        System.out.println("Student 3: " + students[2].name);  
29    }  
30
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
```

```
Inside arrays ...
```

```
Mid-Term 1: 90
```

```
Mid-Term 2: 70
```

```
Final: 80
```

```
Project: 100
```

```
# exams: 4
```

```
Student 1: Student@681a9515
```

```
Student 2: Student@3af49f1c
```

```
Student 3: null
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```


2D Arrays

Dheeru Mundluru

2D Arrays

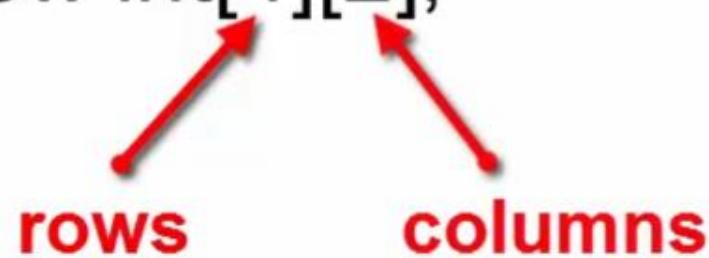
| | | |
|---|---|----|
| | 0 | 1 |
| 0 | 9 | 11 |
| 1 | 2 | 5 |
| 2 | 4 | 4 |
| 3 | 6 | 13 |

myArray[0][1]

Two dimensional Array is : an array of 1D or normal array.

Creating a 2D Array

- ▶ `int[][] myArray = new int[4][2];`



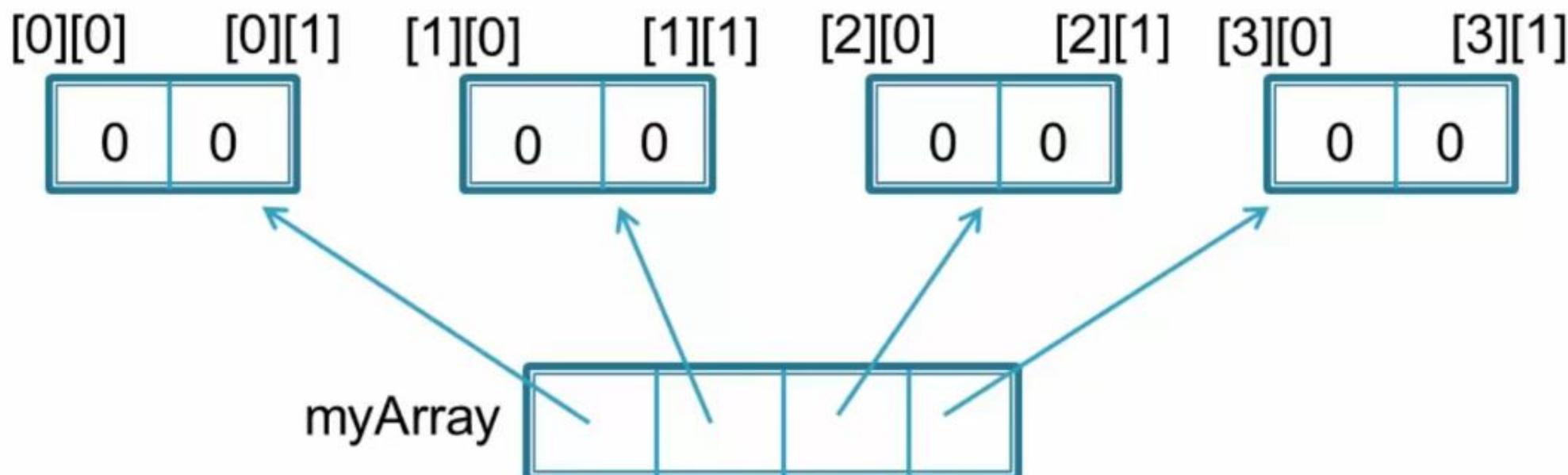
The first bracket points to Row (number of 1 D array) and Second point to columns.

Array can be created :

```
Int[][] myArray = new int[4][2];
```

Creating a 2D Array

- ▶ `int[][] myArray = new int[4][2];`



Declare an 2D Array,

Syntax :

```
int[][] myArray = new int[row/number of 1D array][column size of each 1D array];
```

Example 1:

```
Int[][] myArray = new int[4][2]
```

The above code will create 4 , 1D array ..each with size 2.

Each element in the array will be initialized with default value (0 for int).

Array Initialization

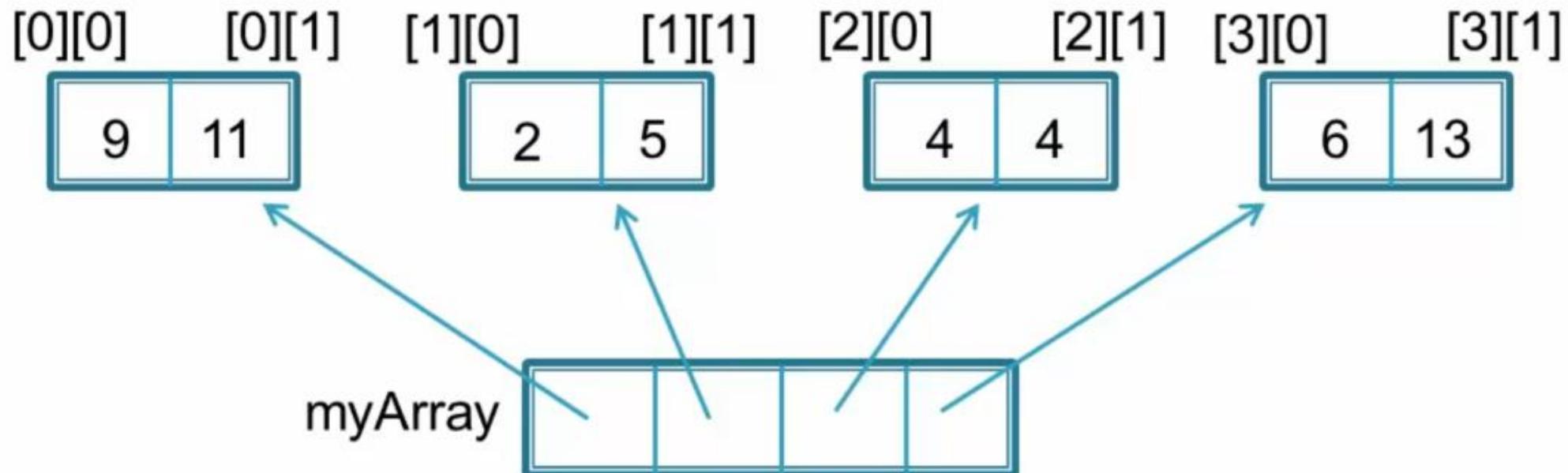
- ▶ myArray[0][0] = 9;
- ▶ myArray[0][1] = 11;
- ▶ myArray[1][0] = 2;
- ◀ ▶ myArray[1][1] = 5;
- ▶ ...

Array Initialization Approach 1

Example 1 :

```
myArray[0][0] = 9;  
myArray[0][1] = 11;  
myArray[1][0] = 2;  
myArray[1][1] = 5;
```

Illustration



2D Array Creation & Initialization

```
int[][] myArray = new int[][]{  
    {9,11},  
    row → {2, 5},  
    {4, 4},  
    {6, 13}  
};
```

Array Initialization Approach 2

Example 1 :

```
Int[][] myArray = new int[][]{  
    {9,11},  
    {2,5},  
    {4,4},  
    {6,13},  
}
```

2D Array Creation & Initialization

```
int[][] myArray = {  
    {9,11},  
    {2, 5},  
    {4, 4},  
    {6, 13}  
};
```

Array Initialization Approach 3

Example 1 :

```
Int[][] myArray = {  
    {9,11},  
    {2,5},  
    {4,4},  
    {6,13},  
}
```

Array with Irregular Rows

- ▶ `int[][] myArray = new int[2][];`
`myArray[0] = new int[5];`
`myArray[1] = new int[2];`

Symmetric Matrix

| | | | | |
|--------|----|----|----|----|
| int[1] | 7 | -2 | 6 | 2 |
| int[2] | -2 | 3 | 20 | 11 |
| int[3] | 6 | 20 | 9 | 5 |
| int[4] | 2 | 11 | 5 | -4 |

Array Initialization Approach 4 // Array with irregular Rows

Example 1 :

```
Int[][] myArray = new int[2][];
myArray[0] = new int[5];
myArray[1] = new int[2];
```

The above code will create 2, 1D array ,
First 1D array with size 5 and
Second 1D array with size 2.

Array Operations

► **length**

- `myArray.length` → 4
- `myArray[0].length` → 2

► `int[] row = myArray[2];`

Get Length of the array

Example :

```
Int[][] myArray = new int[2][];
```

```
myArray[0] = new int[5];  
myArray[1] = new int[2];
```

To get the number of 1D array :

myArray.length , will give the result.

To get the size of first 1D array :

myArray[0].length , will give the result.

3D Arrays

Dheeru Mundluru

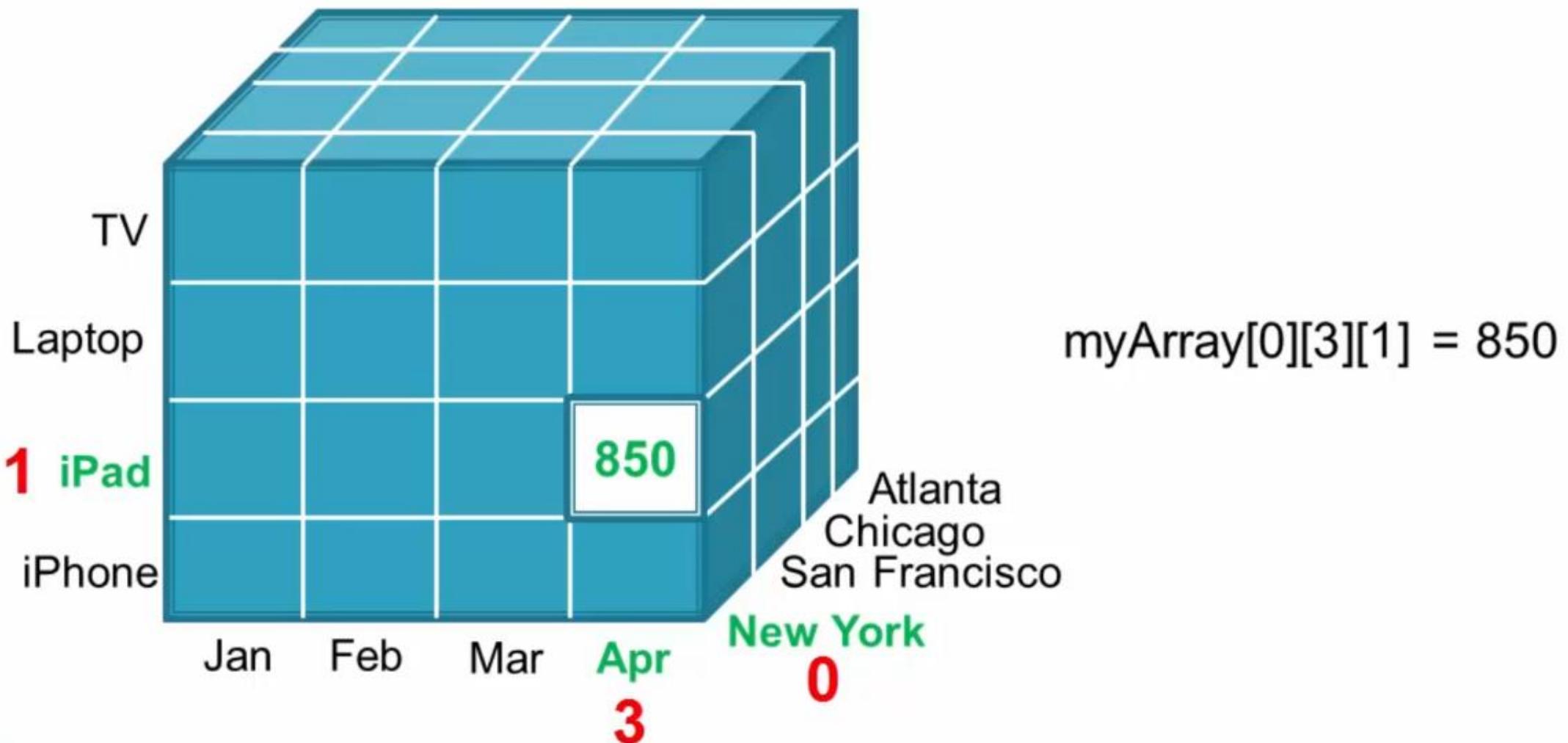
Electronic Store ~ Sales Data



Electronic Store ~ Sales Data



Electronic Store ~ Sales Data



Let's take a quick look at a 3D array.

Example : Data warehousing software store data of store sells
850 iPads were sold in the month of **April** in **New York** store.

We will structure :

Cities are present first dimension

Month second dimension.

Products third dimension.

Means,

```
Int[][][] myArray = new Int[Cities][Month][Product];
```

Then the array can be :

```
Int[][][] myArray = new Int[4][4][4];
```

And 850 will be at **myArray[0][3][1]**

3D Array Creation & Initialization

- ▶ `int[][][] myArray = new int[4][4][4];`
- ▶ `myArray[0][3][1] = 850;`
- ...



Array creation Approach 1 :

```
int[][][] myArray = new Int[4][4][4];
```

```
myArray[0][3][1] = 850;
```

41. 3D Arrays + Demo

```
static void threeDimensionalArrays() {
    System.out.println("\nInside threeDimensionalArrays ...");
    int[][][] unitsSold = new int[][][] {
        { // New York
            {0,0,0,0}, // Jan
            {0,0,0,0}, // Feb
            {0,0,0,0}, // Mar
            {0,850,0,0} // Apr
        },
        { // San Francisco
            {0,0,0,0}, // Jan
            {0,0,0,0}, // Feb
            {0,0,0,0}, // Mar
            {0,0,0,0} // Apr
        },
        {
            {0,0,0,0},
            {0,0,0,0},
            {0,0,0,0},
            {0,0,0,0}
        },
        {
            {0,0,0,0},
            {0,0,0,0},
            {0,0,0,0},
            {0,0,0,0}
        }
    };
}
```

Array creation Approach 2 :

```
int[][][] myArray = new Int[][][]  
{  
    As above  
}
```



ava

```
        {0,0,0,0}, // Jan
        {0,0,0,0}, // Feb
        {0,0,0,0}, // Mar
        {0,0,0,0} // Apr
    },
    {
        {0,0,0,0},
        {0,0,0,0},
        {0,0,0,0},
        {0,0,0,0}
    },
    {
        {0,0,0,0},
        {0,0,0,0},
        {0,0,0,0},
        {0,0,0,0}
    }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

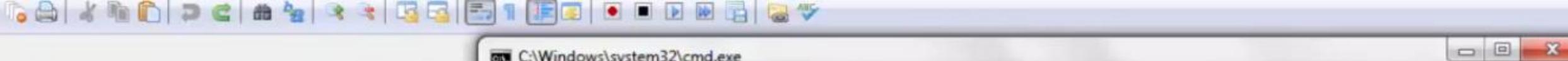
public static void main(String[] args) {
// Language Basics 1
//print();
//primitives();
//
```



Array creation Approach 2 :

Get the value from myArray[0][3][1]

```
System.out.println("unitsSold : " + myArray[0][3][1]);
```



C:\Windows\system32\cmd.exe

C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo

Inside threeDimensionalArrays ...
unitsSold[0][3][1]: 850

C:\javaindepth\src\com\semanticsquare\basics>

```
System.out.println("uni  
}  
  
public static void main(Str  
// Language Basics 1  
//print();  
//primitives();  
//
```




ava

```
}
```

```
static void threeDimensionalArrays() {
    System.out.println("\nInside threeDimensionalArrays ...");
    int[][][] unitsSold = {
        { // New York
            {0,0,0,0}, // Jan
            {0,0,0,0}, // Feb
            {0,0,0,0}, // Mar
            {0,850,0,0} // Apr
        },
        { // San Francisco
            {0,0,0,0}, // Jan
            {0,0,0,0}, // Feb
            {0,0,0,0}, // Mar
            {0,0,0,0} // Apr
        },
        {
            {0,0,0,0},
            {0,0,0,0},
            {0,0,0,0},
            {0,0,0,0}
        },
        {
            {0,0,0,0},
            {0,0,0,0},
            {0,0,0,0}
        }
    };
}
```



Array creation Approach 3 :

```
int[][][] myArray = {  
    ...  
    As above  
}
```

41. 3D Arrays + Demo

```
}
```

```
static void threeDimensionalArrays() {
    System.out.println("\nInside threeDimensionalArrays ...");
    int[][][] unitsSold = {
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
Inside threeDimensionalArrays ...
unitsSold[0][3][1]: 850

C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java

C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
Inside threeDimensionalArrays ...
unitsSold[0][3][1]: 850

C:\javaindepth\src\com\semanticsquare\basics>
```



Methods: Introduction

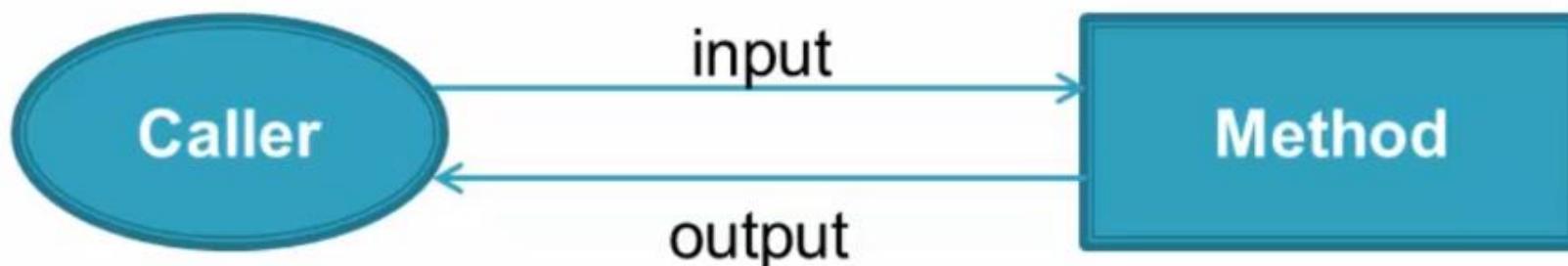
Dheeru Mundluru



Methods

define behavior

- ▶ **Self-contained logic** that can be used *many times*
- ▶ Can receive *input* & generate *output*



We know that an object has, behavior(defined by methods) and state(defined by variables).
Methods represent logics which can be used any number of times.

Methods typically received its input, then they do some processing and finally can generate
some output.

Syntax

The diagram illustrates the components of a method definition and its usage:

- signature**: Points to the part of the method definition that includes the return type and the method name.
- parameters or formal parameters**: Points to the part of the method definition that includes the parameters listed in parentheses after the method name.
- arguments or actual parameters**: Points to the part of the usage statement that includes the arguments passed to the method.

```
returnType methodName(type param1, type param2, ... ) {  
    ...  
    return someValue;  
}  
  
type var = methodName(arg1, arg2, ...)
```

Let's look at a simple depiction of method.

We will have a method which needs to be invoked.

We will also have a caller who invokes the method call.

If Method needs some input data, then the caller has to pass the input data.

The method then performs its actions.

The method then optionally return some data back to the caller

The caller then further uses the returned data.

Here is the syntax for a method :

- It has a **method name**, which is followed by an optional list called **parameters or formal parameters** (to accept input data).

- The parameter can be primitive or object reference
- methods may return a value, the returned value can also be primitive or object reference.
- methods **return type** (mandatory) should always be **right before the method name**
- In Java, the method name + parameter list together are referred as **method signature**.
Methods return type is not part of the signature.
- The caller syntax involves, method name, followed **arguments or actual parameters**.

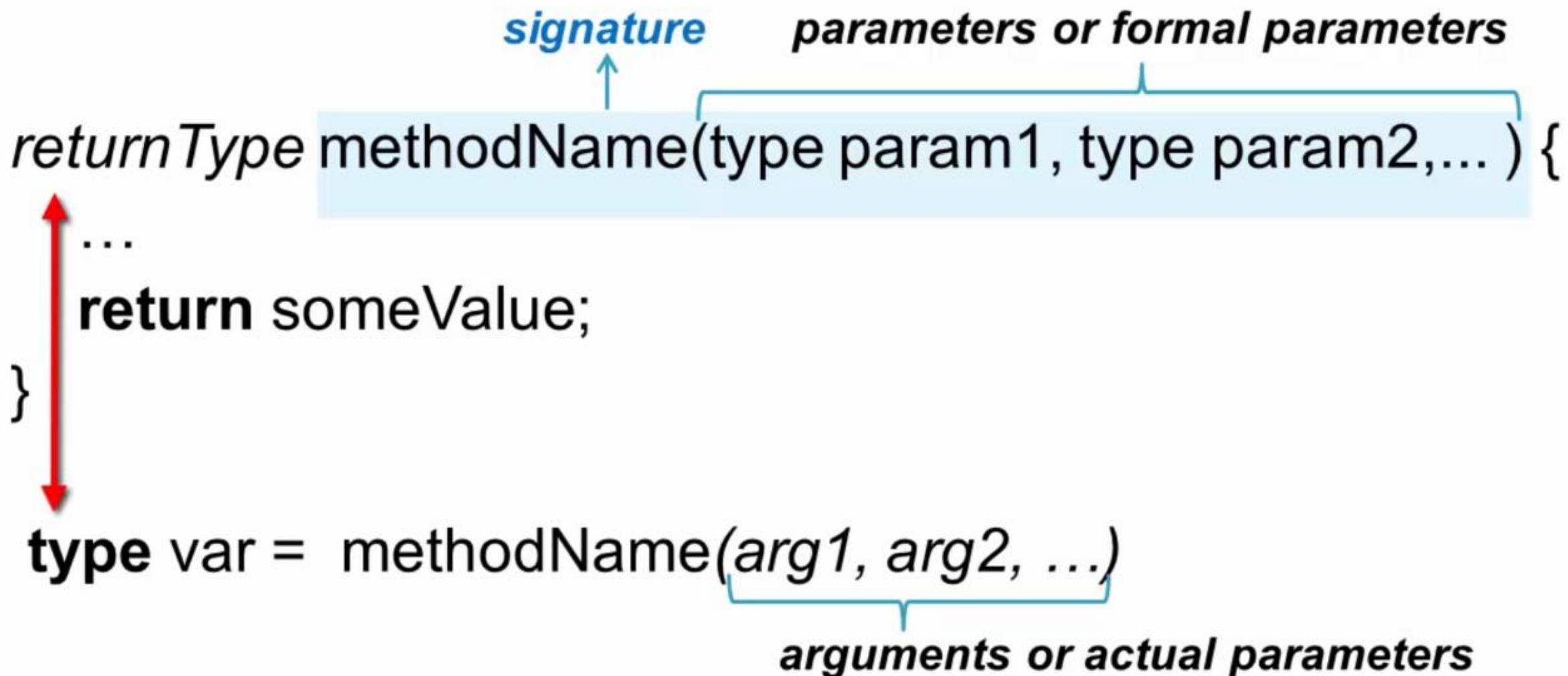
Syntax

```
signature      parameters or formal parameters
↑
returnType methodName(type param1, type param2,...) {
...
return someValue;
}
```

```
type var = methodName(arg1, arg2, ...)
                                         arguments or actual parameters
                                         ↓
```

- The arguments and parameters must be compatible with each other.
- The number of arguments must match with the number of parameters.

Syntax



- If the method is returning a value, the value is typically assigned to some variable, however, nothing stops the caller from ignoring the return value
- The return type in the method declaration and variable data type of the callers must also match.

Syntax

```
signature      parameters or formal parameters  
returnType methodName(type param1, type param2,... ) {  
    ...  
    return someValue;  
}
```

type var = methodName(arg1, arg2, ...)

arguments or actual parameters

- Similarly, the value returned by the method and the return type in the matter declaration should also be compatible.

Return Type

► void

- Nothing to return
- Optional **return;** as last statement
- ```
void print() {
 System.out.println("Hello World!!");
}
```

- Must be *primitive, array, class, interface, or void*
- Other than *void* ➔ *must return value*



If the method has white as **void** type, then it implies that the method does not return anything

java

## 42. Methods: Introduction + Demo

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static double sum(double x, double y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 double d = sum(3.0, 2.0);
 System.out.println(d);
}
```

1x 9:44 / 20:51 CC ANSI

Example 1 :

```
static double sum(double x, double y)
{
 return (x+y);
}
```

This code will work fine

java

## 42. Methods: Introduction + Demo

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 },
 {
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

double static sum(double x, double y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();
}
```

Example 2 :

```
double static sum(double x, double y)
{
 return (x+y);
}
```

The above code will give compilation error because and return type should be just before the method name, which is not in this case (static keyword is in between).

so we have the static keyword in between double and some, then we would get a compilation error.

You see here, it says environmental declaration written type required.

So it says that we need a written type.

So it's got to be before the method name.

This is how it should be now, so that's the method definition.

Now let's go ahead and invoke it.

So let's define the variable as double.

Let's call it D. And that's in the method some that's pass 3.0 2.0 and let's print the value.

So the value that some returns will be assigned to this variability and that gets printed.

So let me run it.

You can see it is printing the value fine.

OK, so we said that.

So these are the arguments and we said that the arguments should have one to one match with this.

So we have two arguments and two tables here.

So the type is also matching the order should match on the number of arguments, should also match here with the parameters.

OK, there should be a one to one correspondence.

Now, we also said the type should be compatible, which means that we can either pass three point or a double value, a double, or we can also pass a smaller data type so we can pass a float also.

As for the smaller than double, so let's go ahead and compile, recompile and rerun it so we see the value file.

So we can also do that next.

Let's change this to float here.

So both the types, we are making it as flawed, so let's recompile this.

You clear the screen.

So in this case, the second argument is giving an issue because there is some incompatibility here, because the second argument that we are passing is double, but the second parameter is flawed.

OK, so double know is a larger data type.

So we have to pass either a float here, OK, float, return, or we can also apply a cost so we can say float.

So we can also do this so we can put a we can insert cost here.

So now both of them are floats and we have the match matching parameters.

So that's good.

Now, here explains why would be a float, but the return type is double, so we said that the return value must be compatible with this.

And so we can see that here we are returning a float, but then here we have double.

So it's compatible.

And so there are no issues with that.

Now, let's see if this is flawed.

So let's change this to float and let's make this.

Double.

This double, so we are coming up to doubles, but the return type is flawed, and so that should give a compilation error.

Because there is incompatibility here, it's got to be compatible, so we have to apply a cost now here when we apply the cost.

So if we just do this, the cost would be applied to only X, but not white.

OK, so what we have to do is we can put a parenthesis here.

Now the is apply to do explicitly so explicitly would be double the Nicosia's applied and then we are good.

OK, so it compiled fine and it still prints the value for.

OK, so that's what we have on here.

The type is flawed, and on this end the type is double.

That is totally fine because fraud is a smaller type and can be assigned to a double.

So that is going to be an implicit cost.

OK, so that's that.

But now let's change it back to the original form that says X plus.

Why?

This is expressed by.

This is.

Now, let's just make this flawed, OK?

So the return type is double and on the colors and the variable is flawed.

OK, so a double is being assigned to float and obviously it should give us a completion error.

So incompatible type.

So what should we do?

So we should just insert a cast here.

And densify.

OK, so that's not so we can apply a cast in this way so we can apply a cast here, we can apply cast here, or we can even apply a cast in the written statement itself.

So next, let's define a method called average.

Let's say a, b, g.

OK, so average, we have to compute for average of these two numbers, so what we can do is let's define a variable called sum here.

So that's a double.

OK, I can see some calls to let's make use of this method sum and pass X, OK?

So if average is involved with some input, so X and Y will have some values, so we will just pass those values to some.

OK, so from the average method, we are invoking the sum.

So we are reusing the functionality of this and here we need to compute the sum.

So let's just say some divided by two.

And here.

So this is Nita.

So let me just recompile.

OK, as you can see, initially, we are printing five because we were in Woking some and then when we invoke average at Prince 2.5, so that's what it is.

So from the main method, we are invoking the average method and from average we are invoking the sun.

OK, so one method is accessing another method and another method is accessing another method.

So what we are seeing here is the benefit of divide and conquer.

So instead of having X plus why here we have a separate method just for doing that and we are reusing that.

OK, so that's software reusability in action.

So this some method can now be used not just by average, but some other code.

Also, for instance, here we are using some and there can be some other program which can reuse that particular sum and that can be in an entirely different project.

OK, so you get the benefit of reusability.

Now, this particular method is very small, very small matter, but let's say if the method is really huge, if the logic here is huge and if we are not making use of methods, then we would probably end up use writing that entire logic here in this method.

And wherever we need that logic, we would be writing that logic.

So the benefit is by having all of that logic and one method, the benefit that we get is we are avoiding duplication.

Had that logic been duplicated in this method and also in all other methods.

And if we have to make a small change, then we have to make that change in all of those places.

But by having all of that logic in one single method, then if the change has to be made, then it would be made only in that particular method.

And we can invoke that method from all of the different places.

So that's the benefit of having methods.

So you have your avoiding duplication and then you also get the benefit of reusability on the court also looks very clean.

So that's the benefit that you get of defining logic and methods.

Now for the white key, what we already know this, so we have written some letters in our demos, so all of these are wired, so they do not write anything.

So we don't have any written statements like we have in this case.

Now, the final thing I want to demonstrate is about passing I.D. So let's say we have a method called search, so let's just say it returns boolean.

And let's call it a search and let's define an entry and let's call it list.

And let's say we want to search for an element in the list on the element of this key.

OK, so that's just a matter of definition.

And let's just return through here.

It doesn't matter.

So what I want to demonstrate is how can we invoke such now here?

So search.

Last to.

And let me define list here.

And.

So this is the list and we want to pass it on, this compiles perfectly fine.

OK, now let's say if we want to pass it here, if you want to do this, then it's not going to work. You see, it gives us some compulsion headers that because this kind of simplistic notation which we have for creating a arrest, is valid only in declaration statements, but we cannot use it when invoking a particular method.

OK, so when we are invoking a particular method, either we have to pass a variable or we have to use

a second way of declaring variables.

If you recall, we had this so you can also do this on purpose.

And so it is created here and this pass to the method.

So the only thing is we cannot do this notation here.

This way of declaring it is valid only in the declaration statements.

OK, so it will not work if we want to pass here.

So that's it.

And let me just compile this.

Here it is, so it compiled fine, so that's it.

And so this notation works only in declaration statements and it will not work in Matorin locations or it will also not work here.

You can also you cannot also do this even in reassignments.

It's not going to work.

OK, as you can see, it gives us some competition.

OK, so it can be possible only within a declaration, statements, statements such as this one.

So that's about it.

So that's an introduction to Method's.

Thank you.

And see you in the next picture.

```
C:\Windows\system32\cmd.exe
java
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:106: error: <identifier> expected
 double static sum(double x, double y) {
 ^
BasicsDemo.java:106: error: invalid method declaration; return type required
 double static sum(double x, double y) {
 ^
2 errors
C:\javaindepth\src\com\semanticsquare\basics>_

```



java

## 42. Methods: Introduction + Demo

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static double sum(double x, double y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 double d = sum(3.0f, 2.0);
 System.out.println(d);
}
```

1x 10:30 / 20:51 © semanticsquare.com 21 In: 118 Col: 26 Sel: 0 Dos Windows ANSI

Example 3 :

```
static double sum(double x, double y)
{
 return (x+y) ;
}

double d = sum(3.0f , 2.0)
```

The above code will work fine because argument 3.0f (float literal) is compatible with double literal (implicit casting will happen).

## 42. Methods: Introduction + Demo

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:106: error: <identifier> expected
 double static sum(double x, double y) {
 ^
BasicsDemo.java:106: error: invalid method declaration; return type required
 double static sum(double x, double y) {
 ^
2 errors
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
```

```
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
S}
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
```

```
5.0
```

```
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```

```
P}
```

```
}
```





java

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static double sum(float x, float y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 double d = sum(3.0f, 2.0);
 System.out.println(d);
}
```



Example 4 :

```
static double sum(float x, float y)
{
 return (x+y);
}

double d = sum(3.0f, 2.0)
```

The above code will give compilation error , because argument 2.0 is double data type (default data type for floating numbers) and parameter is float data type.

## 42. Methods: Introduction + Demo

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 double d = sum(3.0f, 2.0);
 ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```



```
}
```

```
s
```

```
}
```

```
p
```

```
}
```



java

## 42. Methods: Introduction + Demo

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static double sum(float x, float y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 double d = sum(3.0f, (float) 2.0);
 System.out.println(d);
}
```

1x 11:30 / 20:51 CC ANSI

Example 4 :

```
static double sum(float x, float y)
{
 return (x+y);
}

double d = sum(3.0f , (float)2.0)
```

The above code will work, because argument 2.0 of double datatype is explicit type casted to float to match parameter data type.

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 double d = sum(3.0f, 2.0);
 ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
}
```

```
s
```

```
}
```

```
p
```

```
}
```



java

## 42. Methods: Introduction + Demo

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static float sum(double x, double y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 double d = sum(3.0f, (float) 2.0);
 System.out.println(d);
}
```

Example 5 :

```
static float sum(double x, double y)
{
 return (x+y);
}

double d = sum(3.0f , 2.0)
```

The above code will give compilation error “**Incompatible types**”,  
Because the return value of method will be double data type (as x and y are of double data  
type)  
But method type is float data type

## 42. Methods: Introduction + Demo

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 double d = sum(3.0f, 2.0);
 ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:107: error: incompatible types: possible lossy conversion from double to float
 return x + y;
 ^
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>_
```

p

}



42. Methods: Introduction + Demo

```
java
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static float sum(double x, double y) {
 return (float) (x + y);
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 double d = sum(3.0f, (float) 2.0);
 System.out.println(d);
}
```

Example 6 :

```
static float sum(double x, double y)
{
 return (float) (x+y) ;
}

double d = sum(3.0f , 2.0)
```

The above code will run fine.

The return value of method will be double data type (as x and y are of double data type) , is explicit type casted to float to match method type.

When the return value(float type) will be assigned to variable d (of double data type), then it will be implicit type casted to double.

## 42. Methods: Introduction + Demo

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 double d = sum(3.0f, 2.0);
 ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
} C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:107: error: incompatible types: possible lossy conversion from double to float
 return x + y;
 ^
1 error
```

```
} C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
C:\javaindepth\src\com\semanticsquare\basics>_
```



java

## 42. Methods: Introduction + Demo

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static double sum(double x, double y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();
 //
 float d = sum(3.0f, (float) 2.0);
 System.out.println(d);
}
```

1x 13:38 / 20:51 © semanticsquare.com 1/21 In: 118 Col: 12 Sel: 0 DosWindows ANSI

Example 7 :

```
static double sum(double x, double y)
{
 return (x+y) ;
}

float d = sum(3.0f , 2.0)
```

The above code will give compilation error “**Incompatible types**”,  
The return value is double type and variable d is float type.

## 42. Methods: Introduction + Demo

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 double d = sum(3.0f, 2.0);
 ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
} C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:107: error: incompatible types: possible lossy conversion from double to float
 return x + y;
 ^
1 error
```

```
} C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 float d = sum(3.0f, (float) 2.0);
 ^
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>_
```





java

```
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0},
 {0,0,0,0}
 }
};

System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static double sum(double x, double y) {
 return x + y;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 float d = (float) sum(3.0f, (float) 2.0);
 System.out.println(d);
}
```



Example 7 :

```
static double sum(double x, double y)
{
 return (x+y);
}

float d = (float) sum(3.0f , 2.0)
```

The above code will work fine.

The return value double type will be type casted to match , variable d of float type.

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 double d = sum(3.0f, 2.0);
 ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
} C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:107: error: incompatible types: possible lossy conversion from double to float
 return x + y;
 ^
1 error
```

```
} C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

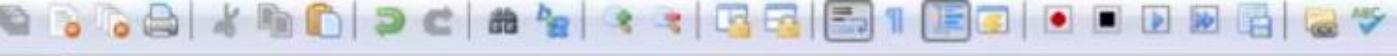
```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:118: error: incompatible types: possible lossy conversion from double to float
 float d = sum(3.0f, (float) 2.0);
 ^
1 error
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```





java

```
 System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
 }

 static double sum(double x, double y) {
 return x + y;
 }

 static double avg(double x, double y) {
 double sum = sum(x, y);
 return sum/2;
 }

 public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 float d = (float) sum(3.0f, (float) 2.0);
 System.out.println(d);

 double d2 = avg(3.0, 2.0);
 System.out.println(d2);
```

I



Example 8 :

```
static double sum(double x, double y)
{
 return (x+y);
}

static double avg(double x, double y)
{
 double sum = sum(x , y);
 return sum/2 ;
}

double d2 = avg(3.0 , 2.0)
```

The above code will work fine.

42. Methods: Introduction + Demo

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
}
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
2.5
C:\javaindepth\src\com\semanticsquare\basics>_
```





java

```
 return sum/2;
 }

 static boolean search(int[] list, int key) {
 return true;
 }

 public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 float d = (float) sum(3.0f, (float) 2.0);
 System.out.println(d);

 double d2 = avg(3.0, 2.0);
 System.out.println(d2); I

 int[] list = {1, 2};

 search(list, 2);
 }
}
```



Example 9 :

```
static boolean search(int[] list, int key)
{
 return true ;
}
```

```
Int[] list = {1,2};
search(list, 2)
```

The above code will work fine.

```
C:\Windows\system32\cmd.exe
java
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
}
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
2.5
S
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
}
C:\javaindepth\src\com\semanticsquare\basics>
p
}

}
```





java

```
 return sum/2;
 }

 static boolean search(int[] list, int key) {
 return true;
 }

 public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 float d = (float) sum(3.0f, (float) 2.0);
 System.out.println(d);

 double d2 = avg(3.0, 2.0);
 System.out.println(d2);

 int[] list = {1, 2};

 search({1, 2}, 2);
 }
}
```



Example 10 :

```
static boolean search(int[] list, int key)
{
 return true ;
}
```

```
Int[] list = {1,2};
search({1,2} , 2)
```

The above code will give compilation error because in java you can not pass list in {1,2} format,  
we need new keyword along with it.

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
 return sum;
}
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
5.0
2.5
static boolean search(int[] list, int key) {
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
 return true;
}
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
BasicsDemo.java:135: error: illegal start of expression
 search({1, 2}, 2);
public static void main(String[] args) {
BasicsDemo.java:135: error: ';' expected
 search({1, 2}, 2);
 ^
BasicsDemo.java:135: error: illegal start of expression
 search({1, 2}, 2);
 ^
BasicsDemo.java:135: error: ';' expected
 search({1, 2}, 2);
 ^
BasicsDemo.java:135: error: illegal start of type
float d = (1, 2); ^BasicsDemo.java:135: error: ';' expected
 search({1, 2}, 2);
 ^
BasicsDemo.java:135: error: illegal start of type
System.out.println(d); ^BasicsDemo.java:135: error: <identifier> expected
int[] l = search({1, 2}, 2); ^BasicsDemo.java:135: error: ';' expected
search(|search({1, 2}, 2); ^

}
9 errors
C:\javaindepth\src\com\semanticsquare\basics>
```



42. Methods: Introduction + Demo

```
java
 return sum/2;
}

static boolean search(int[] list, int key) {
 return true;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 float d = (float) sum(3.0f, (float) 2.0);
 System.out.println(d);

 double d2 = avg(3.0, 2.0);
 System.out.println(d2);

 int[] list = {1, 2};

 search(new int[]{1, 2}, 2);
}
```

1x 19:50 / 20:51 © semanticsquare.com/37 In: 135 Col: 15 Sel: 0 DosWindows ANSI

Example 11 :

```
static boolean search(int[] list, int key)
{
 return true ;
}
```

```
Int[] list = {1,2};
search(new int[]{1,2} , 2)
```

The above code will work fine.

## 42. Methods: Introduction + Demo

```
BasicsDemo.java:135: error: ';' expected
 search({1, 2}, 2);
 ^
BasicsDemo.java:135: error: illegal start of type
 search({1, 2}, 2);
 ^
BasicsDemo.java:135: error: <identifier> expected
 search({1, 2}, 2);
 ^
BasicsDemo.java:135: error: ';' expected
 search({1, 2}, 2); ^
p9 errors
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
C:\javaindepth\src\com\semanticsquare\basics>
```



# Method Types

Dheeru Mundluru



The method can be of two types.

1. Instance method
2. Static method

# Instance Methods

- ▶ Object-level methods
- ▶ **Invocation:** *objectReference.methodName()*
- ▶ Affect *object state*
  - Instance variables
  - Other instance methods



### **Instance methods,**

- They are methods associated with instances (and does not contain static keyword).
- Instance methods represent behavior of objects
- Instance methods can be accessed via object references.
- We use a dot operator on an object reference in order to access instance methods.
- Instance method perform some action on behalf of the object affect the object state by directly/indirectly (by invoking other instance method) manipulating instance variables.

### **Accessibility from Instance Methods:**

- Can access **anything** from an instance method.
- So, we can **even access static variables/methods** defined in the same class as the instance method.

# Static Methods

- ▶ Keyword **static** in declaration
- ▶ Class-level methods
- ▶ **No access to state** (instance variables/methods)
  - Serve as **utility** methods, e.g., `sum(double x, double y)`
  - Can access **static variables**
  - Can access other static methods
- ▶ **Invocation:** `className.methodName()`
- ▶ ***main*** method is static

### **Static methods**

- Static methods have the keyword static associated with method signature
- Static methods are class level methods.
- Static methods do not deal with objects.
- Static method can directly access other static methods in the same class.
- Static methods do not have access to instance variables or instance methods within the same class.
- Static methods serve as utility methods and affect Static variable (which are shared across different objects of the same class).
- Static methods are involved using dot operator on the class name.

### **Accessibility from Static Methods:**

1. Can directly access static variables/methods defined in the same class
2. Cannot directly access instance variables/methods defined in the same class as the static method
3. Can access anything via an object reference. So, from a static method by using an object reference, we can access instance variables/methods



converter.java BasicsDemo.java

```
public class CurrencyConverter {
 // Currency exchange rates of different currencies relative to 1 US dollar
 double[] exchangeRates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};

 void printCurrencies() {
 System.out.println("\nruppee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
 }

 public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();
 cc.printCurrencies();
 }
}
```

This is our currency converter class,

```
double exchangeRates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
```

```
void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}
```

```
Public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();
 cc. printCurrencies();
}
```

This code will run fine , printCurrencies() is instance method and should be accessible using object reference,

```
C:\javaindepth\src\com\semanticsquare\basics>javac CurrencyConverter.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java CurrencyConverter
```

```
pu
rupee: 63.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```





nverter.java

BasicsDemo.java

```
public class CurrencyConverter {

 // Currency exchange rates of different currencies relative to 1 US dollar
 double[] exchangeRates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};

 void printCurrencies() {
 System.out.println("\nruppee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
 }

 public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();
 printCurrencies();
 }
}
```



```
double[] exchangeRates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};

void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}

Public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();
 printCurrencies();
}
```

This code will give compilation error “**non static method can not access using static context**” , printCurrencies() is instance method and can be accessed using object reference only.

```
C:\javaidepath\src\com\semanticsquare\basics>javac CurrencyConverter.java
```

## 43. Method Types + Demo

```
pu rupee: 63.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
yen: 107.0
$australian: 2.0
```

```
C:\javaidepath\src\com\semanticsquare\basics>javac CurrencyConverter.java
CurrencyConverter.java:19: error: non-static method printCurrencies() cannot be referenced from a static context
```

```
 printCurrencies();
 ^
```

```
1 error
```

```
C:\javaidepath\src\com\semanticsquare\basics>_
```

```
}
```



## 43. Method Types + Demo

```
public class CurrencyConverter {

 // Currency exchange rates of different currencies relative to 1 US dollar
 double[] exchangeRates;

 void setExchangeRates(double[] rates) {
 exchangeRates = rates;
 }

 void printCurrencies() {
 System.out.println("\nruppee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
 }

 public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();

 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();
 }
}
```

```
double[] exchangeRates;

void setExchangeRates(double[] rates)
{
 exchangeRates = rates;
}

void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}

public static void main(String[] args)
{
 CurrencyConverter cc = new CurrencyConverter();

 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();
}
```

This code will run fine

```
C:\java\indepth\src\com\semanticsquare\basics>javac CurrencyConverter.java
```

```
43. Method Types + Demo
C:\java\indepth\src\com\semanticsquare\basics>java CurrencyConverter
```

```
rupee: 63.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
yen: 107.0
$australian: 2.0
```

```
C:\java\indepth\src\com\semanticsquare\basics>
```



## 43. Method Types + Demo

```
void printCurrencies() {
 System.out.println("nrupee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
}

public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();

 // Jan 1st
 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();

 rates = [65.0, 5.0, 3.0, 595.5, 18.0, 107.0, 2.0];
 cc.setExchangeRates(rates);

 cc.printCurrencies();
}
```

```
double[] exchangeRates;

void setExchangeRates(double[] rates)
{
 exchangeRates = rates;
}

void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}

public static void main(String[] args)
{
 CurrencyConverter cc = new CurrencyConverter();

 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc. setExchangeRates(rates);
 cc.printCurrencies();

rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc. setExchangeRates(rates);
 cc.printCurrencies();

}
```

This code will give error because array reinitialization does not support below format , we need new keyword.  
**{63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};**

C:\javaindepth\src\com\semanticsquare\basics>javac CurrencyConverter.java

## 43 Method Types + Demo

```
rupee: 63.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac CurrencyConverter.java
CurrencyConverter.java:30: error: illegal start of expression
 rates = {65.0, 5.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 ^
CurrencyConverter.java:30: error: not a statement
 rates = {65.0, 5.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 ^
CurrencyConverter.java:30: error: ';' expected
 rates = {65.0, 5.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 ^
3 errors
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```



## 43. Method Types + Demo

```
void printCurrencies() {
 System.out.println("nrupee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
}

public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();

 // Jan 1st
 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();

 rates = new double[]{65.0, 5.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();
}
```

```
double[] exchangeRates;

void setExchangeRates(double[] rates)
{
 exchangeRates = rates;
}

void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}

public static void main(String[] args)
{
 CurrencyConverter cc = new CurrencyConverter();

 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);
 cc.printCurrencies();

rates = new double[]{63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);
 cc.printCurrencies();
}
```

This code will run fine, because array reinitialization support below format  
**new double[] {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};**

```
C:\javaindepth\src\com\semanticsquare\basics>javac CurrencyConverter.java
```

## 43 Method Types + Demo

```
rupee: 63.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
yen: 107.0
$australian: 2.0
```

```
rupee: 65.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
yen: 107.0
$australian: 2.0
```

```
C:\javaindepth\src\com\semanticsquare\basics>_
```



## 43. Method Types + Demo

```
public class CurrencyConverter {

 // Currency exchange rates of different currencies relative to 1 US dollar
 double[] exchangeRates;

 void setExchangeRates(double[] rates) {
 exchangeRates = rates;
 }

 void updateExchangeRates(int arrayIndex, double newVal) {
 exchangeRates[arrayIndex] = newVal;
 }

 void printCurrencies() {
 System.out.println("\nrupee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
 }

 public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();
 }
}
```

```
double[] exchangeRates;

void updateExchangeRate(int arrayIndex, double newVal)
{
 exchangeRates[arrayIndex] = newVal;
}

void setExchangeRates(double[] rates)
{
 exchangeRates = rates;
}

void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}
```

```
public static void main(String[] args)
{
 CurrencyConverter cc = new CurrencyConverter();

 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);
 cc.printCurrencies();

 rates = new double[]{63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);
 cc.printCurrencies();

cc.updateExchangeRate(0, 66.0);
 cc.printCurrencies();

}
```

This code will run fine



nverter.java

BasicsDemo.java

```
System.out.println("\nruppee: " + exchangeRates[0]);
System.out.println("dirham: " + exchangeRates[1]);
System.out.println("real: " + exchangeRates[2]);
System.out.println("chilean_peso: " + exchangeRates[3]);
System.out.println("mexican_peso: " + exchangeRates[4]);
System.out.println("_yen: " + exchangeRates[5]);
System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
}

public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();

 // Jan 1st
 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();

 rates = new double[]{65.0, 5.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();

 cc.updateExchangeRate(0, 66.0);
 cc.printCurrencies();
}
```



C:\javaindepth\src\com\semanticsquare\basics>java CurrencyConverter

### 43. Method Types + Demo

```
rupee: 59.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

```
rupee: 65.0
dirham: 5.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

```
rupee: 66.0
dirham: 5.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

C:\javaindepth\src\com\semanticsquare\basics>





nverter.java

BasicsDemo.java

```
public class CurrencyConverter {

 // Currency exchange rates of different currencies relative to 1 US dollar
 double[] exchangeRates;

 void setExchangeRates(double[] rates) {
 exchangeRates = rates;
 }

 void updateExchangeRate(int arrayIndex, double newVal) {
 exchangeRates[arrayIndex] = newVal;
 }

 double getExchangeRate(int arrayIndex) {
 return exchangeRates[arrayIndex];
 }

 double computeTransferAmount(int arrayIndex, double amount) {
 return amount * getExchangeRate(arrayIndex);
 }

 void printCurrencies() {
 System.out.println("\nrupee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 }
}
```

The above will also run fine

```
System.out.println("mexican_peso: " + exchangeRates[4]);
System.out.println("_yen: " + exchangeRates[5]);
System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
}

public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();

 // Jan 1st
 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();

 rates = new double[]{65.0, 5.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);

 cc.printCurrencies();

 cc.updateExchangeRate(0, 66.0);
 cc.printCurrencies();

 double amount = cc.computeTransferAmount(0, 1000);
 System.out.println("\nTransferred amount: " + amount);
}
```



C:\java\indepth\src\com\semanticsquare\basics>javac CurrencyConverter.java  
CurrencyConverter.java:52: error: illegal start of expression

## 43. Method Types + Demo

1 error

C:\java\indepth\src\com\semanticsquare\basics>javac CurrencyConverter.java

C:\java\indepth\src\com\semanticsquare\basics>java CurrencyConverter

```
rupee: 63.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

```
rupee: 65.0
dirham: 5.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

```
rupee: 66.0
dirham: 5.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0
```

Transferred amount: 66000.0

C:\java\indepth\src\com\semanticsquare\basics>



## 43. Method Types + Demo

```
}

double getExchangeRate(int arrayIndex) {
 return exchangeRates[arrayIndex];
}

double computeTransferAmount(int arrayIndex, double amount) {
 return amount * getExchangeRate(arrayIndex);
}

static void printCurrencies() {
 System.out.println("\nruppee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
}

public static void main(String[] args) {
 CurrencyConverter cc = new CurrencyConverter();

 // Jan 1st
 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);
}
```

```
double[] exchangeRates;

static void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}

public static void main(String[] args)
{
 CurrencyConverter cc = new CurrencyConverter();

 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);
 cc.printCurrencies();
}
```

This code will run give compilation error “**non static variable exchangeRates can not be accessed by static context**”

Because double[] exchangeRates variable is instance variable and printCurrencies is static method.

## 43. Method Types + Demo

```
C:\java\javaindepth\src\com\semanticsquare\basics>javac CurrencyConverter.java
CurrencyConverter.java:24: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("\nrupee: " + exchangeRates[0]);
 ^
CurrencyConverter.java:25: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("dirham: " + exchangeRates[1]);
 ^
CurrencyConverter.java:26: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("real: " + exchangeRates[2]);
 ^
CurrencyConverter.java:27: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("chilean_peso: " + exchangeRates[3]);
 ^
CurrencyConverter.java:28: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("mexican_peso: " + exchangeRates[4]);
 ^
CurrencyConverter.java:29: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("_yen: " + exchangeRates[5]);
 ^
CurrencyConverter.java:30: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
 ^
CurrencyConverter.java:30: error: non-static variable exchangeRates cannot be referenced from a static context
 System.out.println("$australian: " + exchangeRates[exchangeRates.length-1]);
 ^
8 errors
```

C:\java\javaindepth\src\com\semanticsquare\basics>\_





CurrencyConverter.java BasicsDemo.java

```
public class CurrencyConverter {

 // Currency exchange rates of different currencies relative to 1 US dollar
 static double[] exchangeRates;

 void setExchangeRates(double[] rates) {
 exchangeRates = rates;
 }

 void updateExchangeRate(int arrayIndex, double newVal) {
 exchangeRates[arrayIndex] = newVal;
 }

 double getExchangeRate(int arrayIndex) {
 return exchangeRates[arrayIndex];
 }

 double computeTransferAmount(int arrayIndex, double amount) {
 return amount * getExchangeRate(arrayIndex);
 }

 static void printCurrencies() {
 System.out.println("nrupee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 }
}
```

```
static double[] exchangeRates;

static void printCurrencies()
{
 System.out.println(exchangeRates[0]);
}

public static void main(String[] args)
{
 CurrencyConverter cc = new CurrencyConverter();

 double[] rates = {63.0, 3.0, 3.0, 595.5, 18.0, 107.0, 2.0};
 cc.setExchangeRates(rates);
 cc.printCurrencies();
}
```

This code will run fine.

Because double[] exchangeRates variable and printCurrencies both are static.

```
C:\javaindepth\src\com\semanticsquare\basics>javac CurrencyConverter.java
C:\javaindepth\src\com\semanticsquare\basics>java CurrencyConverter
pu
rupee: 63.0
dirham: 3.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0

rupee: 65.0
dirham: 5.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0

rupee: 66.0
dirham: 5.0
real: 3.0
chilean_peso: 595.5
mexican_peso: 18.0
_yen: 107.0
$australian: 2.0

Transferred amount: 66000.0
C:\javaindepth\src\com\semanticsquare\basics>
```





converter.java

BasicsDemo.java

```
// Currency exchange rates of different currencies relative to 1 US dollar
static double[] exchangeRates;

void setExchangeRates(double[] rates) {
 exchangeRates = rates;
}

void updateExchangeRate(int arrayIndex, double newVal) {
 exchangeRates[arrayIndex] = newVal;
}

double getExchangeRate(int arrayIndex) {
 return exchangeRates[arrayIndex];
}

double computeTransferAmount(int arrayIndex, double amount) {
 return amount * getExchangeRate(arrayIndex);
}

static void printCurrencies() {
 System.out.println("nrupee: " + exchangeRates[0]);
 System.out.println("dirham: " + exchangeRates[1]);
 System.out.println("real: " + exchangeRates[2]);
 System.out.println("chilean_peso: " + exchangeRates[3]);
 System.out.println("mexican_peso: " + exchangeRates[4]);
 System.out.println("_yen: " + exchangeRates[5]);
```

Since static variable can be accessed by instance methods, hence all other instance methos will still work fine too.

# Methods: Passing Data

Dheeru Mundluru





## 45. How Data is Passed to Methods in Java? + Demo

```
void updateId(int newId) {
 newId = 1001;
}
```

```
int id = 1000;
updateId(id);
System.out.println(id); 1000 or 1001?
```



## **Java always supports Pass by value**

Let's see this in case we pass primitive variable to method.

Data can be passed to methods during method invocation, the way this data is passed can vary from one programming language to another

Let's consider the above code **updateIt**.

Now, let's say we are invoking updateId method with a variable **id = 1000**.

After invoking the method updateId, if we print a variable id,

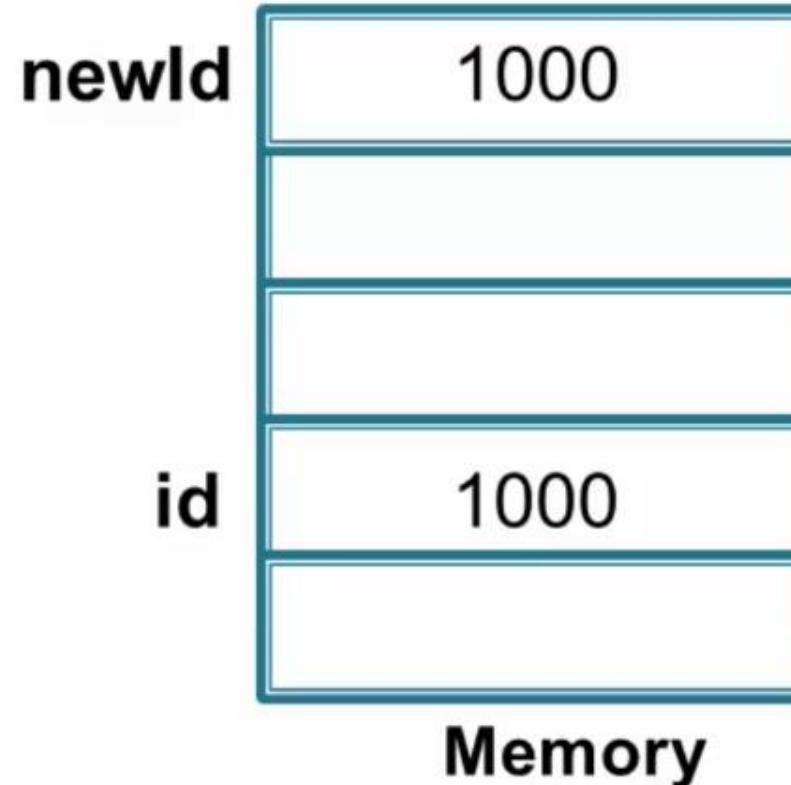
In Java, it would remain as 1000, (whereas in some other programming language, it may get updated 1001).

That's because Java always supports Pass by value.

# Pass by Value: Primitives

```
void updateId(int newId) {
 #2 ✓
 newId = 1001;
 #3
}

int id = 1000;
#1 ✓
updateId(id);
#4
```



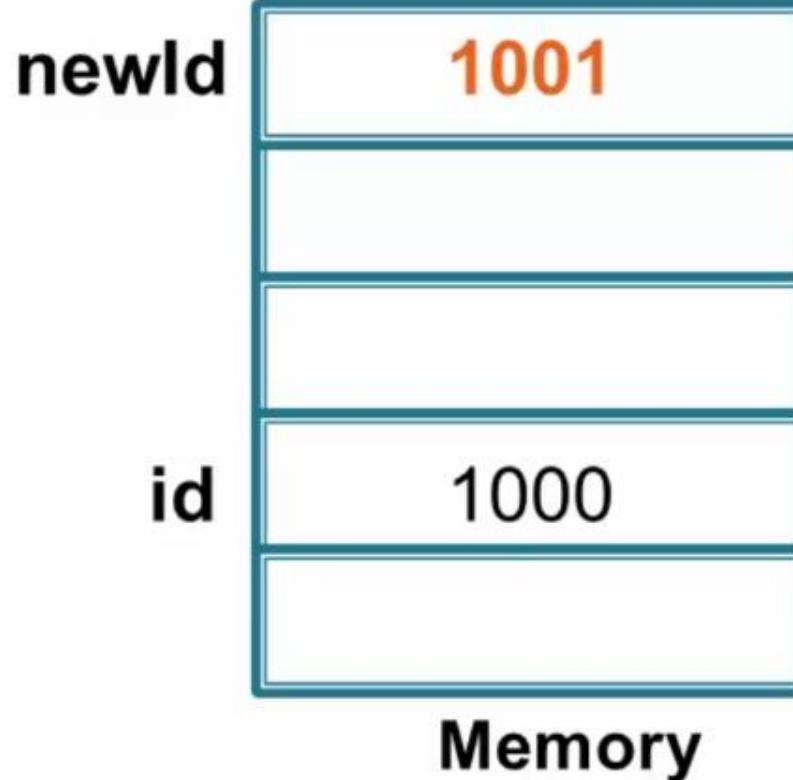
Step 1 : int id =1000; \\id is instance variable

Step 2 : int newid =1000; \\newid is local variable

# Pass by Value: Primitives

```
void updateId(int newId) {
 #2 ✓
 newId = 1001;
 #3 ✓
}
```

```
int id = 1000;
#1 ✓
updateId(id);
#4
```



Step 3 : newid =1001; \\newid value will be updated

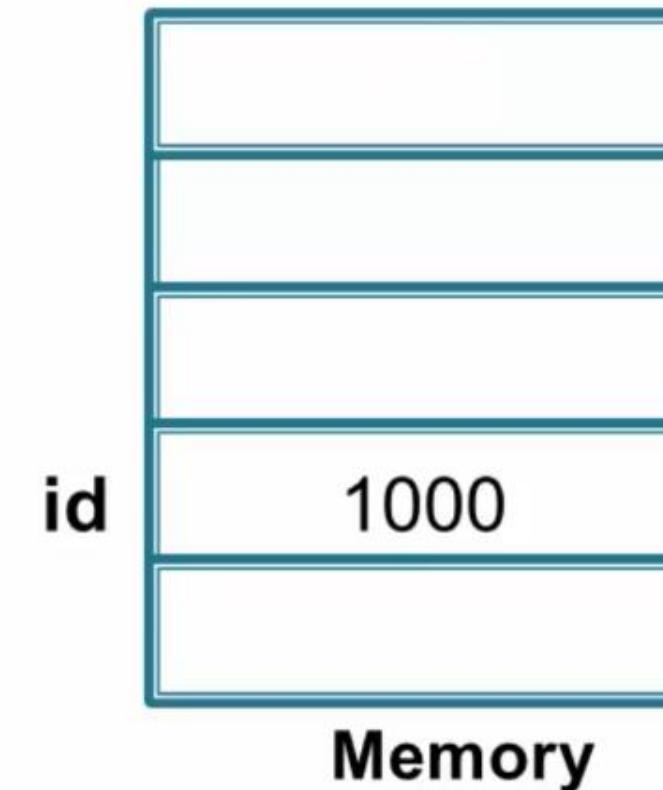
# Pass by Value: Primitives

```
void updateId(int newId) {
 #2 ✓
 newId = 1001;
 #3 ✓
}

```

```
int id = 1000;
#1 ✓
updateId(id);
#4 ✓

```



Step 4 : id =1000; \\newid is local variable, its value will be updated to instance variable id.

```
void updateId(Student s1) {
 s1.id = 1001;
}
```

```
Student s = new Student();
s.id = 1000;
updateId(s);
System.out.println(s.id); 1000 or 1001? 
```

## **Java always supports Pass by value**

Let's see this in case we pass an object reference variable to method.

Student s = new Student(); will create object reference variable s.  
and id attribute of s object reference has been assigned the value 1000.

Now when we pass object s to updateId method , we pass the value (not  
reference/memory address of s).

Reference variable s has value of heap memory address of the object.

So s and s1 both refer to heap memory address of the object.

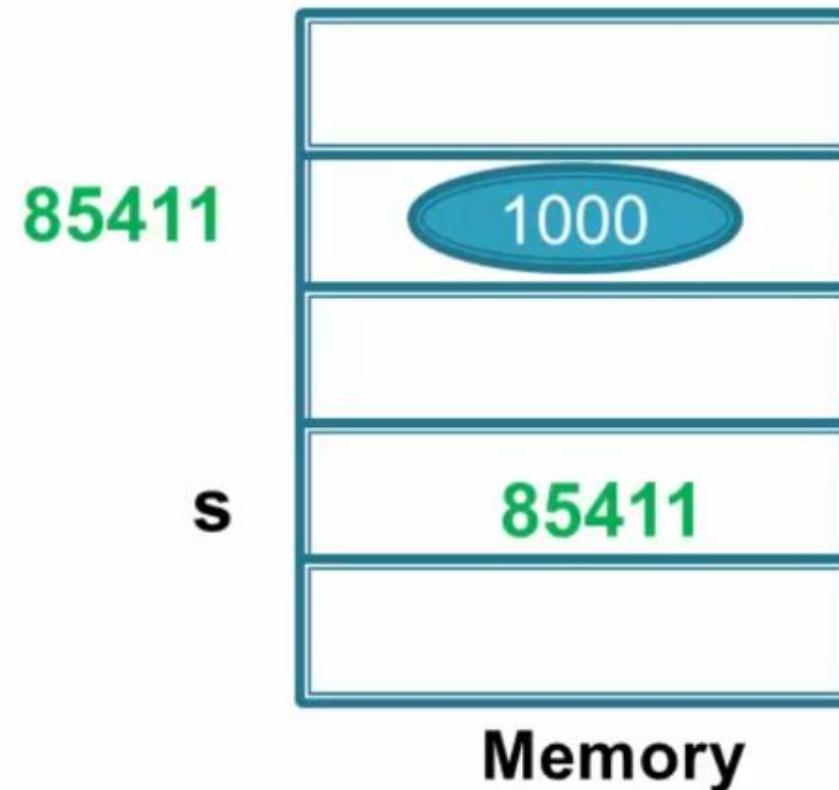
With s1.id = 10001; we updated the value of id.

After the updateId execution completed , s will also consist id = 1001 because s  
and s1 was pointing same object.

# Pass by Value: Object References

```
void updateId(Student s1) {
 #2
 s1.id = 1001;
 #3
}
#4
```

```
Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4
```

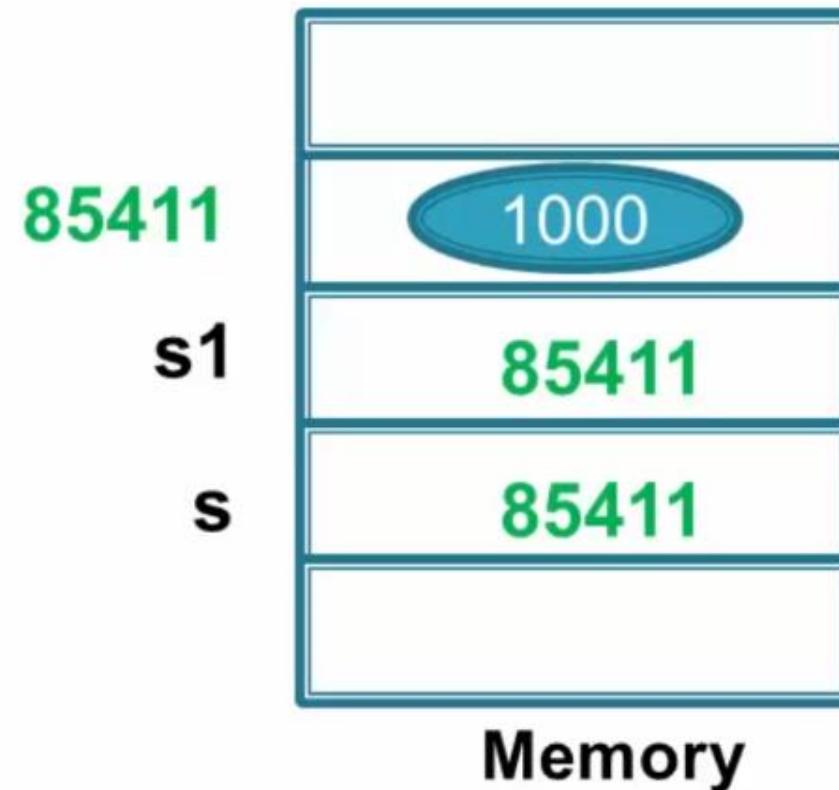


```
Step 1 : Student s = new Student(); // Creates a new object s
 s.id = 1000; // assign value 1000 to id instance variable
```

# Pass by Value: Object References

```
void updateId(Student s1) {
 #2 ✓
 s1.id = 1001;
 #3
}
```

```
Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4
```



Step 2 :

void update(Student s1) // value of s (memory refence of the object) and value of s1 will be same.

# Pass by Value: Object References

```
void updateId(Student s1) {
 #2 ✓
 s1.id = 1001;
 #3 ✓
}
```

```
Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4
```



Step 3 :

```
s1.id = 1001; // id attribute of the object will be updated to 1001 (from it's previous value
1000).
```

# Pass by Value: Object References

```
void updateId(Student s1) {
 #2 ✓
 s1.id = 1001;
 #3 ✓
}

Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4 ✓
```



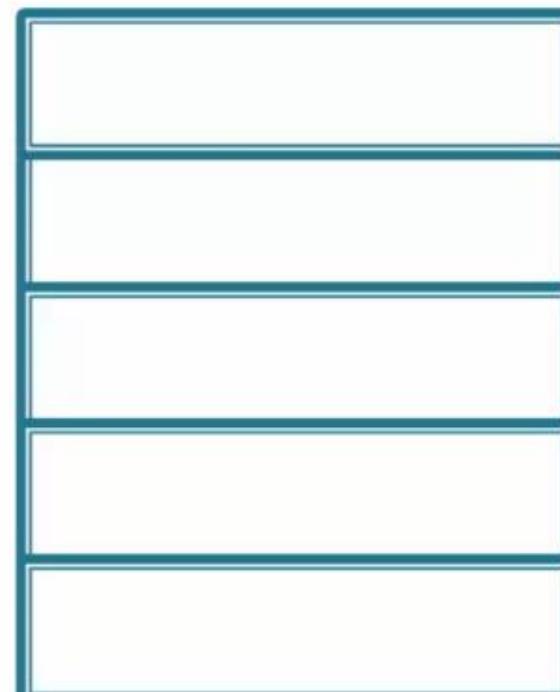
Step 4 :

After updateId execution finished , even s1 will be destroyed but s still has the memory refence of the object hence s.id will be 1001.

# Pass by Value: Reassignment

```
void updateId(Student s1) {
 #2
 s1 = new Student(); ←
 s1.id = 1001;
 #3
}
#4
```

```
Student s = new Student();
s.id = 1000;
#1
updateId(s);
#4
```



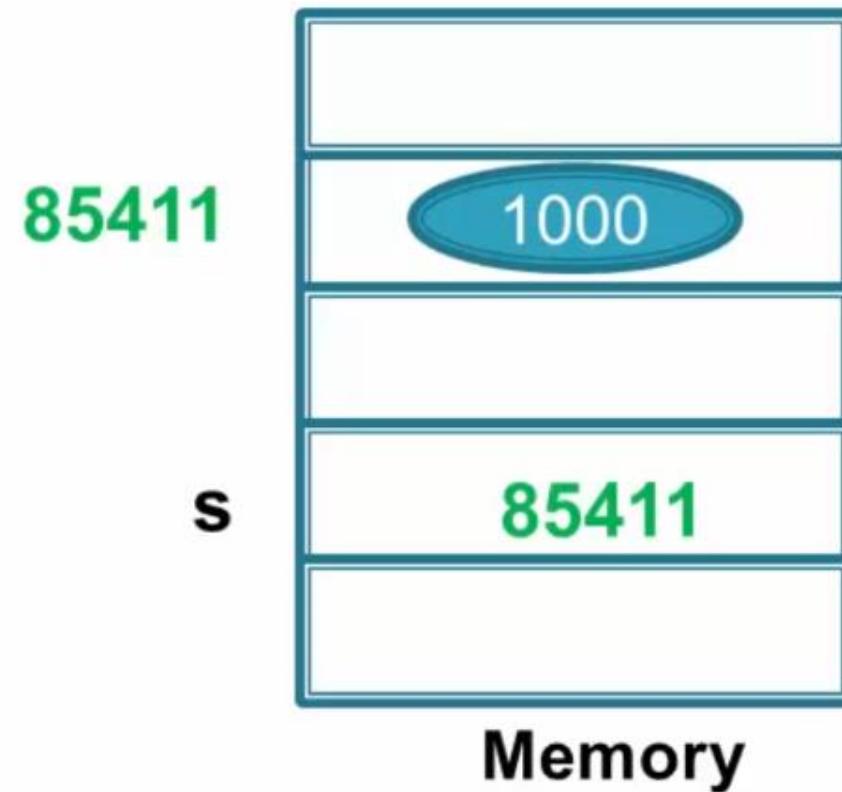
Memory

Now let see , if we add just **s1 = new Student();** , then the s.id value will be remain unchanged.

# Pass by Value: Reassignment

```
void updateId(Student s1) {
 #2
 s1 = new Student();
 s1.id = 1001;
 #3
}
#4
```

```
Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4
```



```
Step 1 : Student s = new Student(); // Creates a new object s
s.id = 1000; // assign value 1000 to id instance variable
```

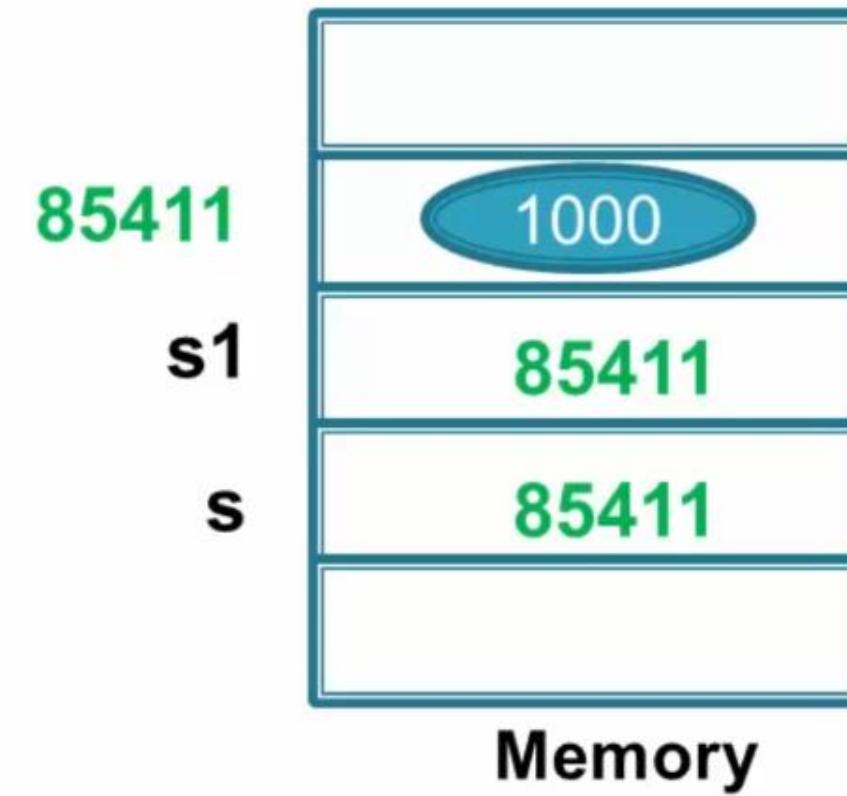
# Pass by Value: Reassignment

```

void updateId(Student s1) {
 #2 ✓
 s1 = new Student();
 s1.id = 1001;
 #3
}

Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4

```



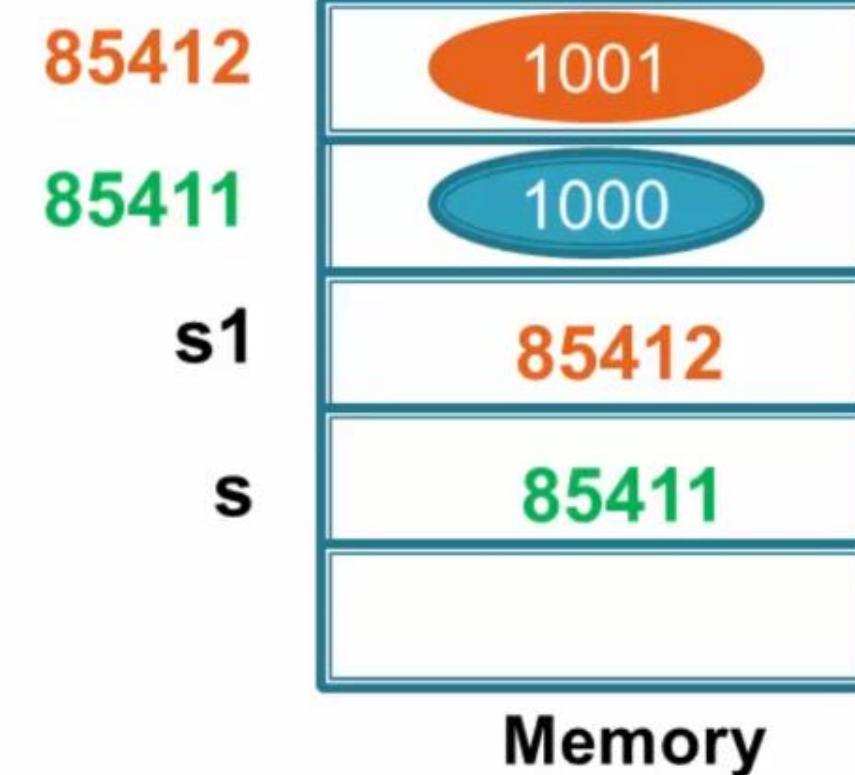
Step 2 :

void update(Student s1) // value of s (memory refence of the object) and value of s1 will be same.

# Pass by Value: Reassignment

```
void updateId(Student s1) {
 #2 ✓
 s1 = new Student();
 s1.id = 1001;
 #3 ✓
}
}
```

```
Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4
```



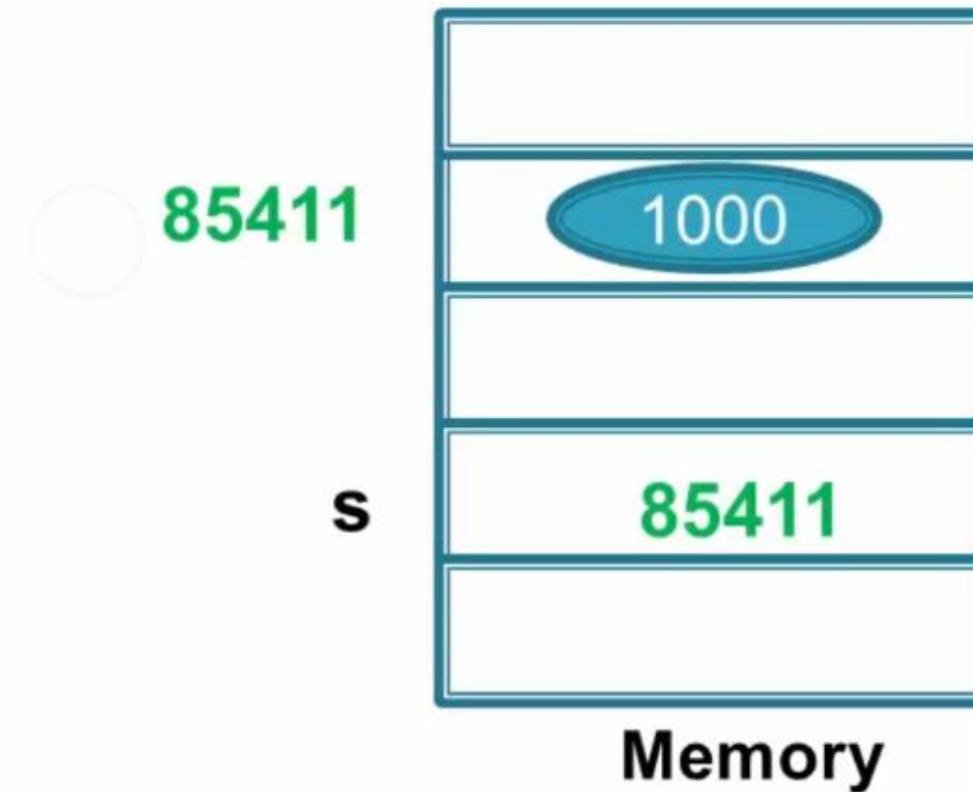
Step 3 :

With below statement, s1 will start pointing to a new object  
s1 = new Student(); //s1 will start pointing to a new object  
S1.id = 1001 // id attribute of new object will be updated to 1001

# Pass by Value: Reassignment

```
void updateId(Student s1) {
 #2 ✓
 s1 = new Student();
 s1.id = 1001;
 #3 ✓
}
```

```
Student s = new Student();
s.id = 1000;
#1 ✓
updateId(s);
#4 ✓
```



Step 4 :

At the end updateId method completed and s1 will be destroyed and it's new object will be unreferenced.

The original object will still be pointed by reference variable s.

In the original object id was never changed hence s.id will still be 1000.

*Java is always pass by value!!*

**Java always supports Pass by value only.**  
**Java does not support Pass by reference.**



java

```
 {0,0,0,0},
 {0,0,0,0}
 }

 System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static void go(int[] array) {
 System.out.println("array[0]: " + array[0]);
 System.out.println("array[1]: " + array[1]);
 array[1] = 22;
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 int[] array = {1, 2};
 go(array);
 System.out.println("array[1]: " + array[1]);
```



Example 1 :

```
Static void go(int[] array)
```

```
{
```

```
System.out.println("array[0] : " + array[0]);
```

```
System.out.println("array[1] : " + array[1]);
```

```
array[1] = 22;
```

```
}
```

```
public static void main(String[] args) {
```

```
int[] array = {1,2};
```

```
go(array);
```

```
System.out.println("array[1] : " + array[1]);
```

```
}
```

The above code work fine and update array[1] with new value.

## 45. How Data is Passed to Methods in Java? + Demo

```
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
array[0]: 1
array[1]: 2
array[1]: 22
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```

```
1]);
```



1x 17:46 / 18:18





# Method Overloading

Dheeru Mundluru



# Method Overloading

*Same name, different parameter lists*

- ▶ **MUST** change parameter list
  - # parameters or parameter types or both must vary
- ▶ Changing only *return type* doesn't matter
- ▶ Applies to *instance & static* methods

## Method overloading

- Sometimes it makes sense to maintain multiple versions of the same method (within the same class) which takes different input data.
- only the **method parameters would differ** (either number of parameters or parameter types or both) on, this feature is referred to as method overloading.
- Simply changing the return type does not overload the method.
- Method overloading is possible for instance as well as static methods.

# Valid Examples

```
void updateProfile(int newId) {}
```

```
void updateProfile(int newId, char gender) {}
```

```
void updateProfile(char gender, int newId) {}
```

```
< void updateProfile(short newId) {} >
```

Let see some valid example of Method overloading.

- In the second and third example,  
the order of the method parameter types has changed, and that's good enough to  
make it a valid method overloading example.  
In second example, we have int followed by char,  
in the third we have char followed by int.
- In first and fourth example,  
Fourth has short as parameter type while,  
First has int as parameter type, that is both are integer data types.  
The two methods still qualify as valid examples of method overloading as the  
parameter type has changed

# Valid Examples

```
void updateProfile(int newId) {}
```

```
void updateProfile(int newId, char gender) {}
```

```
void updateProfile(char gender, int newId) {}
```

```
void updateProfile(short newId) {}
```

```
updateProfile(1000, 'F');
```

```
updateProfile(1000);
```

*byte b = 50;*

```
updateProfile(b);
```

### **Invocation Example 1 :**

```
updateProfile(1000, 'F')
```

It would invoke the second overloaded method as first argument is int and second is char.

### **Invocation Example 2 :**

```
updateProfile(1000);
```

This will invoke the first method because 1000 is an integer literal

Even the fourth method has short as a data type and it also accept integer literals, but still first one is an exact match.  
So the **compiler tries to find a method having parameters with the exact same data type.**

### **Invocation Example 3 :**

```
byte b = 50;
updateProfile(b);
```

But if we have something like this where the variable B is a byte  
Then, the last method which short parameter type would be invoked

Compiler would first try to find a method with the same parameter.  
And if it's not there, then compiler would see a method that has the **next larger data type,**

### **Note :**

**Compiler picks** what method needs to be invoked.

During completion process, Compiler writes into the bytecode about the method that needs to be invoked at runtime.

# Invalid Examples

```
void updateProfile(int newId) {}
```

*boolean* updateProfile(int newId) {}

void updateProfile(int *id*) {}

*static* void updateProfile(int newId) {}

***duplicate method***

Now, let's look at a few invalid overloading examples :

Let's say we have below method definition.

```
void updateProfile(int newId) { }
```

Then below method would not be valid method overloading.

**1. boolean updateProfile(int newId) { } // only the return type is different. Compiler Error : duplicate method.**

**2. void updateProfile(int id) { } //simply changing only parameter name will not work. Compiler Error : duplicate method.**

**3. static void updateProfile(int newId) { } //adding a static method with the same method signature will also not work.**

The only requirement is **parameter list must be different** and the **method names should be same**.

46. Method Overloading + Demo

```
System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static void go(int[] array) {
 System.out.println("array[0]: " + array[0]);
 System.out.println("array[1]: " + array[1]);
 array[1] = 22;
}

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

static void go(short s) {
 System.out.println("go(short s)");
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 int[] array = {1, 2},
 }
}
```

```
static void go(int[] array)
{
 System.out.print("array[0]" + array[0]);
 System.out.print("array[0]" + array[0]);
 array[1] = 22;
}

static void go(int i)
{
 System.out.print("go (int i)");
}

static void go(short s)
{
 System.out.print("go (short s)");
}

public static void main(String[] args)
{
 int[] array = [1, 2];
 go(array); // This will invoke method with signature "go(int[] array)"

 System.out.println("array[1] : " + array[1]);

 go(1000); // This will invoke method with signature "go(int i)"

 byte b = 22;
 go(b); // This will invoke method with signature "go(short s)"
}
```



## 46. Method Overloading + Demo

```
}

static void go(short s) {
 System.out.println("go(short s)");
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();

 int[] array = {1, 2};
 go(array);
 System.out.println("array[1]: " + array[1]);

 go(1000);

 byte b = 22;
 go(b);
}

< >
```



## 46. Method Overloading + Demo

```
java 07. javaindex.html com.semanticsquare.basics>javac BasicsDemo.java
```

```
C:\javaindex\src\com\semanticsquare\basics>java BasicsDemo
array[0]: 1
array[1]: 2
array[1]: 22
go(int i)
go(short s)
```

```
C:\javaindex\src\com\semanticsquare\basics>
```





| OVERVIEW                                  |                                 | MODULE |           | PACKAGE                                                             |  | CLASS | USE | TREE | DEPRECATED | INDEX | HELP | Java SE 1                                                                                                                            |
|-------------------------------------------|---------------------------------|--------|-----------|---------------------------------------------------------------------|--|-------|-----|------|------------|-------|------|--------------------------------------------------------------------------------------------------------------------------------------|
| PREV CLASS                                | NEXT CLASS                      | FRAMES | NO FRAMES | ALL CLASSES                                                         |  |       |     |      |            |       |      | SEARCH: <input type="text"/> Search                                                                                                  |
| SUMMARY: NESTED   FIELD   CONSTR   METHOD | DETAIL: FIELD   CONSTR   METHOD |        |           |                                                                     |  |       |     |      |            |       |      |                                                                                                                                      |
| static void                               |                                 |        |           | sort(double[] a, int fromIndex, int toIndex)                        |  |       |     |      |            |       |      | Sorts the specified range of the array into ascending order.                                                                         |
| static void                               |                                 |        |           | sort(float[] a)                                                     |  |       |     |      |            |       |      | Sorts the specified array into ascending numerical order.                                                                            |
| static void                               |                                 |        |           | sort(float[] a, int fromIndex, int toIndex)                         |  |       |     |      |            |       |      | Sorts the specified range of the array into ascending order.                                                                         |
| static void                               |                                 |        |           | sort(int[] a)                                                       |  |       |     |      |            |       |      | Sorts the specified array into ascending numerical order.                                                                            |
| static void                               |                                 |        |           | sort(int[] a, int fromIndex, int toIndex)                           |  |       |     |      |            |       |      | Sorts the specified range of the array into ascending order.                                                                         |
| static void                               |                                 |        |           | sort(long[] a)                                                      |  |       |     |      |            |       |      | Sorts the specified array into ascending numerical order.                                                                            |
| static void                               |                                 |        |           | sort(long[] a, int fromIndex, int toIndex)                          |  |       |     |      |            |       |      | Sorts the specified range of the array into ascending order.                                                                         |
| static void                               |                                 |        |           | sort(short[] a)                                                     |  |       |     |      |            |       |      | Sorts the specified array into ascending numerical order.                                                                            |
| static void                               |                                 |        |           | sort(short[] a, int fromIndex, int toIndex)                         |  |       |     |      |            |       |      | Sorts the specified range of the array into ascending order.                                                                         |
| static void                               |                                 |        |           | sort(Object[] a)                                                    |  |       |     |      |            |       |      | Sorts the specified array of objects into ascending order according to the natural ordering of its elements.                         |
| static void                               |                                 |        |           | sort(Object[] a, int fromIndex, int toIndex)                        |  |       |     |      |            |       |      | Sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements. |
| static <T> void                           |                                 |        |           | sort(T[] a, int fromIndex, int toIndex,<br>Comparator<? super T> c) |  |       |     |      |            |       |      | Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.              |

This is a glance of method overriding in java library.

# Methods: varargs

Dheeru Mundluru



# varargs

- ▶ Before Java 5 ~ *fixed #* arguments
- ▶ *varargs* ~ variable-length arguments
- ▶ Last parameter can take **variable**# arguments
  - Can be the only parameter



## Varargs

- Methods can have a special type of parameter called **varargs** (variable length arguments), even though it's a single parameter it can take either 0 or 1 or any number of arguments.
- it could be the only parameter of a method or if the matter has more than one parameter then **varargs must be the last one.**

# Syntax & Invocation

- ▶ **Syntax:** three dots following parameter type
  - foo(boolean flag, **int...** items)
- ▶ **Invocation**
  - *Array:* foo(true, **new int[]{1, 2, 3}**)
  - *Comma-separated arguments:* foo(true, 1, 2, 3)
  - *Omitted:* foo(true)

*infinitely overloaded*

Methos Signature Syntax of Varargs :

- methodname(type+parameter1, type+parameter2 , **type ...parameter**)

When it comes to varargs Invocation (any number of arguments of a data-type).

- Arguments can be **an array of any size**
- It can also be a **sequence off any number of comma separated arguments** because compiler automatically can merge them into an array.

**So basically the arguments are getting passed as an array whether you do it explicitly or not. It just gives an illusion that the method is infinitely overloaded.**

Example 1:

```
foo(true, new int[]{1,2,3})
```

Example 2:

```
foo(true, 1,2,3)
```

Example 3:

```
foo(true)
```

# varargs Restrictions

- ▶ Must be *last* parameter
  - `foo(int... items, boolean flag)` - **illegal**
- ▶ Only one varargs parameter
  - `foo(boolean flag, int... v1, int... v2)` - **illegal**



The above examples shows invalid invocation of varargs, as varargs can only be the **last parameter** and there can **only be one vargs parameter** in an invocation.

# Why varargs?

- ▶ Can't we use `foo(boolean flag, int[] items)`?
- ▶ varargs provides **simpler** & **flexible** invocation
  - `foo(true, 1, 2, 3)`
  - `foo(true)`, i.e., no `foo(true, null)` or `foo(true, new int[]{})`
  - `foo(true, veryLargeArray)`
- ▶ **printf**(String format, Object... args)
  - `System.out.printf("DOB: %d/%d/%d", 1, 1, 1978); //DOB: 1/1/1978`

The main advantage of varargs is that, it provides us simpler flexible invocation

- If you don't know how many values to pass you can simply pass varargs
- if there is nothing to pass you simply need not pass anything at all (you don't have to pass null on an empty )
- if you have a large number of values to pass you can simply gather them into an array and pass it to them.

# varargs & main Method

public static void main(**String[]** args) {}

*or*

public static void main(**String...** args)



Since varargs can also represent array, so we can also write “**String[] args**” as “**String...args**”.  
Hence the main method can be written as :

```
public static void main(String... args)
```

Instead of

```
public static void main(String[] args)
```

# varargs & Overloaded Methods

- ▶ Invalid overload example
  - foo(boolean flag, int... items)
  - foo(boolean flag, **int[]** items)
- ▶ varargs method will be matched last
- ▶ **Demo:** basics\BasicsDemo.varargsOverload()

1. Since varargs represents array, we can not overload them with each other :

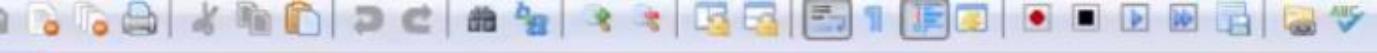
Example :

**foo(boolean flag, int... items)**

Can not be overload with

**foo(boolean flag, int[] items)**

2. If we have multiple overloaded methods , then varargs method will be match at the end



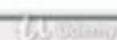
```
va

 }

 System.out.println("unitsSold[0][3][1]: " + unitsSold[0][3][1]);
}

static void varargsOverload(boolean b, int i, int j, int k){
 System.out.println("\nInside varargsOverload without varargs ...");
}
static void varargsOverload(boolean b, int... list){
 System.out.println("\nInside varargsOverload with varargs ...");
 System.out.println("list.length: " + list.length);
}

public static void main(String[] args) {
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();
 varargsOverload(true, 1, 2, 3);
 varargsOverload(true, 1, 2, 3, 4, 5, 6, 7, 8);
 varargsOverload(true);
}
```



Example :

```
static void varargsOverload(boolean b, int i, int j, int k)
{
 System.out.println("\nInside varargsOverload without varargs ...");
}

static void varargsOverload(boolean b, int... list)
{
 System.out.println("\nInside varargsOverload with varargs ...");
 System.out.println("list.length: " + list.length);

}

public static void main(String[] args)
{
 varargsOverload(true, 1, 2, 3); // Will invoke non varargs method
 varargsOverload(true, 1, 2, 3, 4, 5, 6, 7, 8); // Will invoke varargs method
 varargsOverload(true); // Will invoke varargs method
}
```

## 47. Methods: varargs + Demo

```
System.out.println("un
}

static void varargsOverload()
 System.out.println("\nI
}
static void varargsOverload()
 System.out.println("\nI
 System.out.println("lis
}

public static void main(St
 // Language Basics 1
 //print();
 //primitives();
 //typeCasting();
 //arrays();
 //threeDimensionalArrays();
 varargsOverload(true, 1,
 varargsOverload(true, 1,
 varargsOverload(true);
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac BasicsDemo.java
C:\javaindepth\src\com\semanticsquare\basics>java BasicsDemo
Inside varargsOverload without varargs ...
Inside varargsOverload with varargs ...
list.length: 8
Inside varargsOverload with varargs ...
list.length: 0
C:\javaindepth\src\com\semanticsquare\basics>
```





# Constructors

Dheeru Mundluru



```
Student student1 = new Student(); ←
student1.id = 1000;
student1.name = "John";
student1.gender = "male";
student1.age = 18;
student1.phone = 223_456_7890L;
student1.gpa = 3.8;
student1.degree = 'B';
student1.international = false;
student1.compute();
```

```
Student student2 = new Student(); ←
student2.id = 1001;
student2.name = "Raj";
student2.gender = "male";
student2.age = 21;
student2.phone = 223_456_9999L;
student2.gpa = 3.4;
student2.degree = 'M';
student2.international = true;
student2.compute();
```

## Constructor.

- Constructors are all about constructing or creating objects.
- If we don't create constructor, then compiler creates the constructor and insert it into the bytecode for us implicitly.
- The main purpose of a constructor is to **initialize the state of the object**, means **initialize the instance variables** of the class.
- However constructor can do many things like methods.

```
Student student1 = new Student();
student1.id = 1000;
student1.name = "John";
student1.gender = "male";
student1.age = 18;
student1.phone = 223_456_7890L;
student1.gpa = 3.8;
student1.degree = 'B';
student1.international = false;
student1.compute();
```

**Compiler ~ creating constructor**

```
Student student2 = new Student();
student2.id = 1001;
student2.name = "Raj";
student2.gender = "male";
student2.age = 21;
student2.phone = 223_456_9999L;
student2.gpa = 3.4;
student2.degree = 'M';
student2.international = true;
student2.compute();
```

**Main Goal:**  
**Initialize object state**

We use ObjectReference.field to initialize fields (example : student.id = 1000;  
However the professional way is to initialize filed or object state via constructor.

*Invoking constructor*

`Student s = new Student();`

Allocate space for  
**reference variable**

Allocate space for  
new **student object**

`s ← student object's address`

We're using a constructor while object creation statement

Example :

```
Student s = new Student();
```

Let's look at the three parts of object creation

**1. Student s :**

- This part allocate space for a reference variable.

**2. new Student():**

- This part allocate space for a new student object.

**3. = :**

- The assignment operator assigns the memory address of the student object to the variables.

Let see the Second part ( **new Student();** ) in detail :

- Particular fragment(**Student()**) is actually an invocation of the constructor.
- The second step here does three things:
  - a. First it creates an object in memory,
  - b. Then it invokes the object constructor to initialize the state of object.
  - c. Finally, it returns the memory address of the created object.

# Syntax

*varargs*

```
ClassName(type param1, type param2, ...) {
 ...
}
```



Syntax of a constructor in Class :

```
ClassName
{
 ClassName(type parameter1, type parameter1)
 {.
 .
 .
 }
}
```

- It must have the same name as the class
- syntax wise, it looks very similar to a method
- Like methods, it can optionally have parameters.
- A constructor can also have varargs parameters.
- Constructor doesn't contain return type unlike methods.

# Example

```
class Student {
 int id;
 Student(int newId) {
 id = newId;
 }
}
```

```
Student s = new Student();
s.id = 1001;
```

Student s = new Student(1001);

Example :

```
class Student {
 Int id;

 Student(int newId) {
 Id = newId;
 }
}
```

In the above example,

- As a rule, the constructor has the same name as the class.
- Class definition with a constructor has a single parameter.
- The class has a single instance variable **id**, which is initialized to its default zero already. The constructor reinitialize in the id field to add new value.
- Instead of creating object first and then set the instance state with multiple lines as below code.

```
Student s = new Student();
s.id = 1001;
```

- Using the below new student object creation statement, value of **id** has been set to 1001 during object creation itself.

```
Student s = new Student(1001);
```

- In the long run when we create so many object and each with many instance variable, the constructor saves a lot of coding effort.

# Default Constructor

*Constructor not provided*



Compiler **inserts** one with no parameters

*(no-args constructor)*

Constructor Provided → Default constructor **NOT** inserted

## **Default Constructor**

- When we do not provide a constructor in our class, only then the compiler automatically inserts a default constructor.
- The default constructor doesn't have any parameters.
- The constructors without any parameter, is also referred to as **no-args constructor**.

# Example ~ No Constructor

```
class Student {
 int id;
}
```

*compile*



```
class Student {
 int id;
 Student() {}
}
```

Student s = new Student();



## **Default Constructor's Example**

- We know if the class includes any constructor , then the compiler will not insert it's default constructor.

```
class Student {
 int id;
}
```

Above class definition does not have a constructor.

So when we compile the class, the compiler inserts noargs construct into the bytecode , something like below.

```
class Student {
 int id;

 Student() {}

}
```

- Hence **Student s = new Student();** will be valid.

# Example ~ With Constructor

```
class Student {
 int id;

 Student(int newId) {
 id = newId;
 }
}
```

```
Student s = new Student(1001);
Student s = new Student(); // illegal
```

## Non-Default Constructor's Example

```
class Student {
 int id;

 Student(int newId){
 id = newid;
 }
}
```

Above class definition have a constructor.  
So when we compile the class, the compiler will not  
inserts noargs construct into the bytecode

- Hence **Student s = new Student();** will be invalid. Unless we include one args construct ourself.

# return Statement

***cannot return value!*** (like void method)

```
class Student {

 Student(int id) {
 ...
 return;
 // unreachable code
 }
}
```

```
class Student {

 Student(int id) {
 if (someCondition) {
 ...
 return;
 }
 // reachable code
 }
}
```

Constructor can not have return type in the constructor signature.  
However, they can still have an empty return statements (**return;**).

In the left side example above:

When empty return statement gets executed, the control immediately returns back from the constructor.  
Now, if we have any code immediately following the return statement, then it will be unreachable and give us completion error and we have to fix it.

In the right side example above:

If the condition is true, the control will reach till return statement and the code just after return will not be executed.

However in this case the code would be executed when the condition is false, so we will not get any compilation error.

## 48. Constructors + Demo

```
String name;
String gender;
int age;
long phone;
double gpa;
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0; I
```

Student(){}  
I

```
void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 }

 System.out.println("\nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
```

### **Constructor Example 1 : No Args Constructor**

Let's add below no args constructor in student class as below.

**Student(){ }**

Let's just look at the output once again, it will be same as earlier.

## 48. Constructors + Demo

```
String name;
String gender;
int age;
long phone;
double gpa;
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 17000.0;

Student(){}
}

void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 }

 System.out.println("id: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
C:\javaindepth\src\com\semanticsquare\basics>java Student
id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
computeCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
computeCount: 2

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
computeCount: 2@semanticsquare.com
```





```
double tuitionFees = 12000.0;
double internationalFees = 5000.0;
```

```
Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 | | | | | |
 | | | | | | char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;
}
```

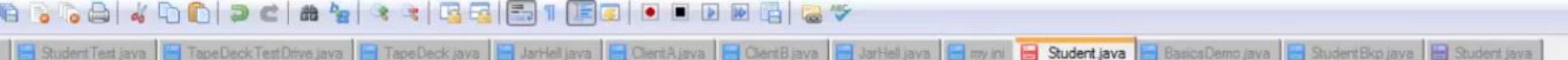
```
Student() {}
```

```
void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 }
}
```

### **Constructor Example 2 : Parameterized Constructor**

Now let's go ahead and add parameterized constructor and remove no-args constructor as above and update the instance creation accordingly (as shown in next few slides).



```
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);

}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);
 student1.id = 1000;
 student1.name = "John";
 student1.gender = "male";
 student1.age = 18;
 student1.phone = 223_456_7890L;
 student1.gpa = 3.8;
 student1.degree = 'B';
 student1.international = false;
 student1.compute();

 Student student2 = new Student();
 student2.id = 1001;
 student2.name = "Raj";
 student2.gender = "male";
 student2.age = 21;
 student2.phone = 223_456_9999L;
 student2.gpa = 3.4;
 student2.degree = 'M';
```



```
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);
}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);
student1.compute();

 Student student2 = new Student();
student2.id = 1001;
student2.name = "Raj";
student2.gender = "male";
student2.age = 21;
student2.phone = 223_456_9999L;
Istudent2.gpa = 3.4;
student2.degree = 'M';
student2.international = true;
student2.compute();

 Student student3 = new Student();
student3.id = 1002;
student3.name = "Anita";
student3.gender = "female";
student3.age = 20;
```



## 48. Constructors + Demo

```
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);
}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);
 student1.compute();

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);
 student2.compute();

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);
 student3.compute();

 System.out.println("Student.computeCount: " + Student.computeCount);
}
```



## 48. Constructors + Demo

```
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);

}

public static void main(String[] args) {
 Student student1 = new Student();
 student1.compute();

 Student student2 = new Student();
 student2.compute();

 Student student3 = new Student();
 student3.compute();

 System.out.println("Student count: " + computeCount);
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
C:\javaindepth\src\com\semanticsquare\basics>java Student
id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
computeCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
computeCount: 2

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
computeCount:@semanticsquare.com
```

Let see the output, it's same as earlier.

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java JarHello.java ClientA.java ClientB.java JarHello.java my.ini Student.java BasicsDemo.java StudentBkp.java Student.java

System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);
}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);
 student1.compute();

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);
 student2.compute();

 Student student3 = new Student(); // 1002, "Anita", "Female", 20, 223_456_8888L, 4.0, 'M',
 student3.compute();

 System.out.println("Student.computeCount: " + Student.computeCount);
}

}
```

## **Constructor Example 2 : Need of No-args constructor while using constructors**

Now let's go ahead and remove the state information from object 3 creation when we don't have just added parameterized constructor (no args constructor is not added by us).

As compiler will not add no-args constructor this time, so it should give us a completion error "**actual and formal argument lists differ in length**".

So we cannot do that.

## 48. Constructors + Demo

```
System.out.println("next line");
System.out.println("name");
System.out.println("gender");
System.out.println("age");
System.out.println("phone number");
System.out.println("gpa");
System.out.println("degree");
System.out.println("tuition");
System.out.println("computation");
}
```

```
public static void main(String[] args) {
 Student student1 = new Student();
 student1.compute();

 Student student2 = new Student();
 student2.compute();

 Student student3 = new Student();
 student3.compute();

 System.out.println("Student object created");
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:57: error: constructor Student in class Student cannot be applied to given types;
 Student student3 = new Student(); // 1002, "Anita", "female",
20, 223_456_8888L, 4.0, 'M', true
 ^
 required: int,String,String,int,long,double,char,boolean
 found: no arguments
 reason: actual and formal argument lists differ in length
1 error
C:\javaindepth\src\com\semanticsquare\basics>
```





```
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 | | | char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;
}

Student() {}

void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 }
}
```

Now to fix this let us add a no-args constructor ourself.

**Now the compilation will works fine and initialize the object 3 with default value.**

## 48. Constructors + Demo

```
double tuitionFees = 12000.
double internationalFees =

Student(int newId, String name, char newDegree)
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInter
}

Student()

void compute() {
 computeCount = computeC
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuiti
 }
}
```

Command Prompt

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
C:\javaindepth\src\com\semanticsquare\basics>java Student
id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
computeCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
computeCount: 2
```

```
id: 0
nextId: 1
name: null
gender: null
age: 0
phone: 0
gpa: 0.0
degree:
tuitionFees: 12000.0
computeCount:@3semanticsquare.com
```



## 48. Constructors + Demo

```
 }

 System.out.println("\nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
 }

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);
 student1.compute();

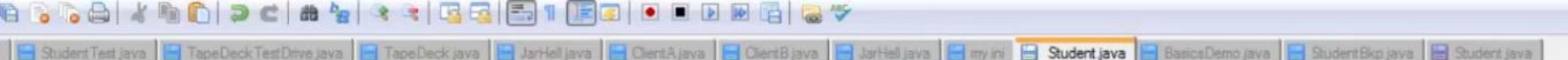
 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);
 student2.compute();

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);
 student3.compute();
}
```

#### **Constructor Example 4 : Constructor , more then just setting the object state**

Now let's restore the object 3 with it's previous values.

We can see the compute(); statement is repetitive for all the object , so there is a possibility of code refactoring.



```
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 compute();
}

Student() {}

void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 }
}
```

Let's add `compute();` call in the constructor itself.

## 48. Constructors + Demo

```
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);
}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);
 student1.compute();

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);
 student2.compute();

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);
 student3.compute();

 System.out.println("Student.computeCount: " + Student.computeCount);
}
```

And remove the individual calls of compute() method from object reference variables.

## 48. Constructors + Demo

TapeDeck.java JarHello.java ClientA.java ClientB.java JarHello.java my ini Student.java BasicDemo.java StudentBkp.java Student.java

```
}

System.out.println("\nid: " + id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);
}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);

 System.out.println("Student.computeCount: " + Student.computeCount);
}
```



## 48. Constructors + Demo

```
Student() {}

void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + 1000;
 }

 System.out.println("id: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
}

public static void main(String[] args) {
 Student s1 = new Student("John", "male", 18, "2234567890", 3.8, "B", 12000.0);
 Student s2 = new Student("Raj", "male", 21, "2234569999", 3.4, "M", 17000.0);
 Student s3 = new Student("Anita", "female", 20, "2234568888", 4.0, "M", 17000.0);
}
```

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java

C:\javaindepth\src\com\semanticsquare\basics>java Student

```
id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
computeCount: 1
```

```
id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
computeCount: 2
```

```
id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
computeCount:@semanticsquare.com
```

Let's compile + execute, the code will run as earlier.

## 48. Constructors + Demo

```
Student() {}
```

udemy.com is now full screen [Exit Full Screen \(Esc\)](#)

```
void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 }

 return;
 System.out.println("nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
}
```

#### **Constructor Example 5 : Constructor with empty return statement**

The above code will give compilation error “**Unreachable statement**”, because part of code (just after return statement) will always be unreachable.

```
Student() {}

void compute() {
 computeCount = computeC
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuiti
 }

 return;
 System.out.println("\nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
}
```

Command Prompt

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:42: error: unreachable statement
 System.out.println("\nid: " + id);
 ^
1 error
```

C:\javaindepth\src\com\semanticsquare\basics>





```
Student() {}

void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 return;
 }

 System.out.println("nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
}
```

### **Constructor Example 6 : Constructor with empty return statement**

The above code will work fine, because there is no dead code.

## 48. Constructors + Demo

```
Student() {}

void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees * 1.2;
 return;
 }

 System.out.println("id: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
 System.out.println("Student.computeCount: " + Student.computeCount);
}
```

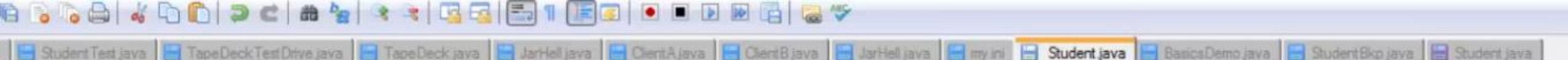
```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java

C:\javaindepth\src\com\semanticsquare\basics>java Student

id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
computeCount: 1
Student.computeCount: 3
```

```
C:\javaindepth\src\com\semanticsquare\basics>
```





```
Student() {}

void compute() {
 computeCount = computeCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 return;
 }

 System.out.println("nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("computeCount: " + computeCount);
}
```

### **Constructor Example 7 :**

Let's put the compute methods code in constructor itself. The code will still work fine.

## 48. Constructors + Demo

```
phone = newPhone;
gpa = newGpa;
degree = newDegree;
international = isInternational;

compute();

}

Student() {}

void compute() {

}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);

 System.out.println("Student.computeCount: " + Student.computeCount);
}
```



## 48. Constructors + Demo

```
phone = newPhone;
gpa = newGpa;
degree = newDegree;
international = isInternational;

computeCount = computeCount + 1;
int nextId = id + 1;

if (international) {
 tuitionFees = tuitionFees + internationalFees;
 return;
}

System.out.println("\nid: " + id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("computeCount: " + computeCount);
```

}



## 48. Constructors + Demo

```
Student(int newId, String newName, char newDegree) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees * 1.2;
 //return;
 }

 System.out.println("\nNew Student Added!");
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("studentCount: " + studentCount);
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
```

```
C:\javaindepth\src\com\semanticsquare\basics>java Student
```

```
id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
studentCount: 1
```

```
id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
studentCount: 2
```

```
id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
studentCount:@semanticsquare.com
```





# Constructor Overloading

Dheeru Mundluru



# Constructor Overloading

- ▶ Same *overloading rules* as methods

Parameter list **must** be different

- ▶ Create objects using *any* constructor

## **Constructor Overloading**

It similar to method overloading, here in a class, multiple constructors with different parameters creates constructor overloading.

# “Why” Constructor Overloading?

## Convenience

*simplifies object creation statements*

They make object creation statements more convenient.

# Example

`new FileOutputStream(new File("blah.txt"), false)`

`new FileOutputStream("blah.txt")`

`FileOutputStream(String name, boolean append)` ~ *append*

`FileOutputStream(String name)` ~ *overwrite*

`FileOutputStream(File file)`



`FileOutputStream(File file, boolean append)`

true - append  
false - overwrite

`FileOutputStream(FileDescriptor fdObj)`

**TreeMap ~ independent constructors**

Let's look at an example from the Java Library.

There are few constructors above from **File Output Stream**, class, with different set of parameters and internally connected by invocation.

## 49. Constructor Overloading + Demo

```
boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 System.out.println("\nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
```

### **Constructor Overloading Example 1 :**

Let's add a new constructor just for non-international students.

The only difference will be parameter isInternational will not be used there..

```
boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree){
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
}

studentCount = studentCount + 1;
int nextId = id + 1;

System.out.println("nid: " + id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
```



#### 49. Constructor Overloading + Demo

```
System.out.println("\nid: " + id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);
}
```

```
Student() {}
```

```
public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B', false);

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);

 System.out.println("Student.studentCount: " + Student.studentCount);
}
```

Let's also update the invocation of first non international student by removing isInternational parameter.

## 49. Constructor Overloading + Demo

```
System.out.println("\nid: " + id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);
}
```

```
Student() {}
```

```
public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B');
 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);
 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);
 System.out.println("Student.studentCount: " + Student.studentCount);
}
```



The screenshot shows a Java development environment with two windows. On the left is the code editor window for `Student.java`, which contains the following code:

```
System.out.println("\n");
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);
}

Student() {}

public static void main(String[] args) {
 Student student1 = new Student("John", "male", 18, "2234567890", 3.8, "B", 12000.0);

 Student student2 = new Student("Raj", "male", 21, "2234569999", 3.4, "M", 17000.0);

 Student student3 = new Student("Anita", "female", 20, "2234568888", 4.0, "M", 17000.0);

 System.out.println("Student count: " + studentCount);
}
```

The right window is the Command Prompt showing the execution of the code:

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
C:\javaindepth\src\com\semanticsquare\basics>java Student
id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
studentCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
studentCount: 2

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
studentCount: 2
```

The code run fine.

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java JarHello.java ClientA.java ClientB.java JarHello.java my.ini Student.java BasicsDemo.java StudentBkp.java Student.java

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
 }

 if (gpa >= 3.5) {
 tuitionFees = tuitionFees - 1000;
 }

 System.out.println("\nid: " + id);
 System.out.println("nextId: " + nextId);
```

### **Constructor Overloading Example 2 :**

Let now add some discount for all those students having GPA > 3.5.

So we have added the code in international student constructor.

## 49. Constructor Overloading + Demo

```
Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 System.out.println("\nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("studentCount: " + studentCount);
```

Suppose, mistakenly we forgot to add the code in non-international student constructor.

#### 49. Constructor Overloading + Demo

```
System.out.println("\nid: " + id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + name);
System.out.println("gender: " + gender);
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);
}

Student() {}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B');

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);

 System.out.println("Student.studentCount: " + Student.studentCount);
}
```

Let's invoke and see if the discount is applied to deserving student1 and student3 or not.

## 49. Constructor Overloading + Demo

```
 }
 }

 System.out.println("id: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("studentCount: " + studentCount);

}

Student() {}

public static void main(String[] args) {
 Student student1 = new Student("John", "male", 18, "2234567890", 3.8, "B", 12000.0);
 Student student2 = new Student("Raj", "male", 21, "2234569999", 3.4, "M", 17000.0);
 Student student3 = new Student("Anita", "female", 20, "2234568888", 4.0, "M", 16000.0);

 System.out.println("Student1 Details");
 System.out.println("id: " + student1.id);
 System.out.println("nextId: " + student1.nextId);
 System.out.println("name: " + student1.name);
 System.out.println("gender: " + student1.gender);
 System.out.println("age: " + student1.age);
 System.out.println("phone: " + student1.phone);
 System.out.println("gpa: " + student1.gpa);
 System.out.println("degree: " + student1.degree);
 System.out.println("tuitionFees: " + student1.tuitionFees);
 System.out.println("studentCount: " + student1.studentCount);

 System.out.println("Student2 Details");
 System.out.println("id: " + student2.id);
 System.out.println("nextId: " + student2.nextId);
 System.out.println("name: " + student2.name);
 System.out.println("gender: " + student2.gender);
 System.out.println("age: " + student2.age);
 System.out.println("phone: " + student2.phone);
 System.out.println("gpa: " + student2.gpa);
 System.out.println("degree: " + student2.degree);
 System.out.println("tuitionFees: " + student2.tuitionFees);
 System.out.println("studentCount: " + student2.studentCount);

 System.out.println("Student3 Details");
 System.out.println("id: " + student3.id);
 System.out.println("nextId: " + student3.nextId);
 System.out.println("name: " + student3.name);
 System.out.println("gender: " + student3.gender);
 System.out.println("age: " + student3.age);
 System.out.println("phone: " + student3.phone);
 System.out.println("gpa: " + student3.gpa);
 System.out.println("degree: " + student3.degree);
 System.out.println("tuitionFees: " + student3.tuitionFees);
 System.out.println("studentCount: " + student3.studentCount);
}
```

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java

C:\javaindepth\src\com\semanticsquare\basics>java Student

```
id: 1000
nextId: 1001
name: John
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
studentCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
studentCount: 2

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 16000.0
studentCount: 3
```

Let's invoke and see if the discount is applied to deserving student1 and student3 or not.

It seems the discount applied only for the international student (student3) , since we forgot to add the respective code to non-international student construct student1's fees is still 12000 instead of 11000.

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java JarHello.java ClientA.java ClientB.java JarHello.java my.ini Student.java BasicsDemo.java StudentBkp.java Student.java

char newDegree) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 System.out.println("nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("studentCount: " + studentCount);
}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
```

It's always risky to keep multiple versions of repeatable code. Method , Constructor , this reference helps to reduce multiple versions of repeatable code.

Let's try to reduce this repeatable code.

#### 49. Constructor Overloading + Demo

```
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;
```

Let's use the `this()` invocation

`this` is a reference variable that **refers to the current object**.

### **Constructor Overloading Example 3 :**

As we know, if the student is not international, the `isInternational` value will be false.

So For non-international student construct, we will redirect it for all the code execution to international student construct by adding below line.

```
this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
```

It will give the correct results as expected.

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java

char degree;

boolean international;
double tuitionFees = 12000;
double internationalFees = 17000;

Student(int newId, String newName, char newDegree, boolean newInternational)
 this(newId, newName, newDegree, newInternational, 12000.0);

Student(int newId, String newName, char newDegree, boolean newInternational, double newTuitionFees)
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;
 studentCount = studentCount + 1;
 int nextId = id + 1;
```

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java

C:\javaindepth\src\com\semanticsquare\basics>java Student

id: 1000  
nextId: 1001  
name: John  
gender: male  
age: 18  
phone: 2234567890  
gpa: 3.8  
degree: B  
tuitionFees: 12000.0  
studentCount: 1

id: 1001  
nextId: 1002  
name: Raj  
gender: male  
age: 21  
phone: 2234569999  
gpa: 3.4  
degree: M  
tuitionFees: 17000.0  
studentCount: 2

id: 1002  
nextId: 1003  
name: Anita  
gender: female  
age: 20  
phone: 2234568888  
gpa: 4.0  
degree: M  
tuitionFees: 17000.0  
studentCount: 3



## 49. Constructor Overloading + Demo

```
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 15000.0;

Student(int newId, String newName, char newGender, double newAge, String newPhone, double newGpa, char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;
 studentCount = studentCount + 1;
}

Student(int newId, String newName, char newDegree) {
 id = newId;
 name = newName;
 degree = newDegree;
 studentCount = studentCount + 1;
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:20: error: call to this must be first statement in constructor
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree,
 ^
e, false);
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>
```

#### **Constructor Overloading Example 4 :**

**While using inside the constructor , this() should always be the first statement** in the construct.

As we can see , if we add anything before this statement, the code will give compilation error "**call to this must be the first statement in constructor**".

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java JarHello.java ClientA.java ClientB.java JarHello.java my.ini Student.java BasicsDemo.java StudentBkp.java Student.java

char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {

 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
 int i = 100; I
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);

}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
```

### **Constructor Overloading Example 5 :**

**Since, this() should always be the first statement** in the construct, we can't even use more than one this statement in a constructor.

As we can see , if we add multiple this statement, the code will again give compilation error "**call to this must be the first statement in constructor**".

## 49. Constructor Overloading + Demo

```
char degree;

boolean international;
double tuitionFees = 12000;
double internationalFees = 15000;

Student(int newId, String newName, char newGender, int newAge, String newPhone, double newGpa, boolean newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;
}

Student(int newId, String newName, char newDegree) {
 id = newId;
 name = newName;
 gender = 'M';
 age = 100;
 phone = null;
 gpa = 0.0;
 degree = newDegree;
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:20: error: call to this must be first statement in constructor
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree,
 ^
 false);
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:22: error: call to this must be first statement in constructor
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree,
 ^
 false);
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>
```



```
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree);
}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;
```

1. **MUST be first statement**
2. **Only one-per-constructor**
3. **No recursive invocation**

### **Constructor Overloading Example 6 :**

**this()** can not be used for recursion in the construct, as above.

The code will again give compilation error "recursive constructor invocation".

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java

char degree;

boolean international;
double tuitionFees = 12000;
double internationalFees = 15000;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa, char newDegree, boolean isInternational) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;
}
```

```
Command Prompt

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:20: error: call to this must be first statement in constructor
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree,
 ^
e, false);
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:22: error: call to this must be first statement in constructor
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree,
 ^
e, false);
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:17: error: recursive constructor invocation
 Student(int newId, String newName, String newGender, int newAge, long
newPhone, double newGpa,
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>
```



```
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree);
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;
```

### **Constructor Overloading Example 7 :**

**this()** can not be used for recursion in the construct, as above which is variation of recursion.

The code will again give compilation error "**recursive constructor invocation**".

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java

char degree;

boolean international;
double tuitionFees = 12000;
double internationalFees = 15000;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa, char newDegree) {
 this(newId, newName, newGender, newAge, newPhone, newGpa);
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
 international = isInternational();
}

studentCount = studentCount + 1;
int nextId = id + 1;
```

Command Prompt

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:17: error: recursive constructor invocation
 Student(int newId, String newName, String newGender, int newAge, long
newPhone, double newGpa,
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>_
```



```
static int studentCount;
```

```
int id;
```

```
String name;
String gender;
int age;
long phone;
double gpa;
char degree;
```

```
boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;
```

```
Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 this(id, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
}
```

```
Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 id = newId;
 name = newName; 18 people have written a note here.
 gender = newGender;
 age = newAge;
```

1. **MUST be first statement**
2. **Only one-per-constructor**
3. **No recursive invocation**
4. **No instance variables as arguments**

#### **Constructor Overloading Example 8 :**

**this()** can not use instance variable as argument in the construct, as above.

The code will again give compilation error "can not reference id , before supertype constructor called".

```
StudentTest.java TapeDeckTestDrive.java TapeDeck.java

static int studentCount;

int id;
String name;
String gender;
int age;
long phone;
double gpa;
char degree;

boolean international;
double tuitionFees = 12000;
double internationalFees = 18000;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa, char newDegree) {
 this(id, newName, newGender, newAge, newPhone, newGpa, newDegree);
}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa, char newDegree) {
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:17: error: recursive constructor invocation
 Student(int newId, String newName, String newGender, int newAge, long
newPhone, double newGpa,
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:19: error: cannot reference id before supertype constructor has bee
n called
 this(id, newName, newGender, newAge, newPhone, newGpa, newDegree,
false);
 ^
1 error
```



```
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 //this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree);
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
 }
}
```

### **Constructor Overloading Example 9 :**

We can make any constructor as primary depends on the needs,  
In the above code, we made non-international student constructor as primary and it has been  
called from international student constructor.

The code will run fine.

## 50. Demo: Constructor Overloading ~ Alternate way of delegating

```
double internationalFees =

Student(int newId, String n
 | char newDegree
 | //this(newId, newName,
 | id = newId;
 | name = newName;
 | gender = newGender;
 | age = newAge;
 | phone = newPhone;
 | gpa = newGpa;
 | degree = newDegree;
 }

Student(int newId, String n
 | char newDegree
 | this(newId, newName, ne
 | international = isInter
 | studentCount = studentC
 | int nextId = id + 1;

 if (international) {
 tuitionFees = tuiti
 //return;
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java

C:\javaindepth\src\com\semanticsquare\basics>java Student

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
studentCount: 1

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
studentCount: 2
Student.studentCount: 2
```

```
C:\javaindepth\src\com\semanticsquare\basics>_
```



# ‘this’ Reference

We use **This reference** to represent the current object

It can be done using **this.instanceVaraibleName as below** (we can also access instance method and static variable using this refence in similar way , but generally it's not required).



BasicsDemo.java StudentBkp.java new\_2

```
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 //this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
 id = newId;
 name = newName;
 gender = newGender;
 age = newAge;
 phone = newPhone;
 gpa = newGpa;
 degree = newDegree;
}

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree, boolean isInternational) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree);
 international = isInternational;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
 }
}
```

**Example 1 : This Reference under constructor :**

**Part 1 : Requirement**

In the above constructor, let us try to use the same names as respective instance variables.

## 51. Demo: this Reference

```
long phone;
double gpa;
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
}

Student(int id, String name, String gender, int age, long phone, double gpa,
 char degree, boolean international) {
 id = id;
 name = name;
 gender = gender;
 age = age;
 phone = phone;
 gpa = gpa;
 degree = degree;
 international = international;

 studentCount = studentCount + 1;
```





BasicsDemo.java StudentBkp.java new\_2

```
System.out.println("age: " + age);
System.out.println("phone: " + phone);
System.out.println("gpa: " + gpa);
System.out.println("degree: " + degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);
}

Student() {}

public static void main(String[] args) {
 Student student1 = new Student(1000, "John", "male", 18, 223_456_7890L, 3.8, 'B');

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);

 System.out.println("\nStudent.studentCount: " + Student.studentCount);

 System.out.println("\nName of student 1: " + student1.name);
 System.out.println("\nName of student 2: " + student2.name);
 System.out.println("\nName of student 3: " + student3.name);
}
```



Let us also print name attribute from them.

## 51. Demo: this Reference

```
System.out.println("age: " + student1.age);
System.out.println("photo: " + student1.photo);
System.out.println("gpa: " + student1.gpa);
System.out.println("degree: " + student1.degree);
System.out.println("tuitionFees: " + student1.tuitionFees);
System.out.println("studentCount: " + student1.studentCount);

}

Student() {}

public static void main(String[] args) {
 Student student1 = new Student("Raj", "M", 21, 3.4, 12000.0, 1);

 Student student2 = new Student("Anita", "F", 20, 4.0, 17000.0, 2);

 Student student3 = new Student("Vishal", "M", 22, 3.9, 17000.0, 3);

 System.out.println("\nStudent 1 details:");
 System.out.println("Name of student 1: " + student1.name);
 System.out.println("Name of student 2: " + student2.name);
 System.out.println("Name of student 3: " + student3.name);
}
```

```
degree: B
tuitionFees: 12000.0
studentCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
studentCount: 2

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
studentCount: 3

Student.studentCount: 3

Name of student 1: null
Name of student 2: null
Name of student 3: null
```

C:\JavaInDepth\src\com\semanticsquare\basics>

The values are coming as null, let's understand the reason.

## 51. Demo: this Reference

```
String gender;
int age;
long phone;
double gpa;
char degree;

boolean international;
double tuitionFees = 12000.0;
double internationalFees = 5000.0;

Student(int newId, String newName, String newGender, int newAge, long newPhone, double newGpa,
 char newDegree) {
 this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
}

Student(int id, String name, String gender, int age, long phone, double gpa,
 char degree, boolean international) {
 id = id;
 name = name;
 gender = gender;
 age = age;
 phone = phone;
 gpa = gpa;
 degree = degree;
 international = international;
```

The reason is **Variable hiding or shadowing**

Since we have mentioned parameter name(or local variable name) and instance variable name same, The local variable is shadowing the instance variable within the constructor (also possible in instance method).

So ,

Id = id

It is working as

**Local id** variable = **Local id** variable

Instead of

**current instance 's id** variable = **Local id** variable

And same applies to other instance variable in the code, so we need a mechanism to refer to represent current object here, this can be done using this Reference (different then this() invocation we used earlier)



BasicsDemo.java StudentBkp.java new\_2

```
 | char newDegree) {
this(newId, newName, newGender, newAge, newPhone, newGpa, newDegree, false);
}

Student(int id, String name, String gender, int age, long phone, double gpa,
 | char degree, boolean international) {
this.id = id;
this.name = name;
this.gender = gender;
this.age = age;
this.phone = phone;
this.gpa = gpa;
this.degree = degree;
this.international = international;

studentCount = studentCount + 1;
int nextId = id + 1;

if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
}
System.out.println("\nid: " + id);
System.out.println("nextId: " + nextId);
```

**Example 1 : This Reference under constructor :**

**Part 2 : Solution**

We use **This reference** to represent the current object

It can be done using **this.instanceVaraibleName as below** (we can also access instance method and static variable using this refence in similar way , but generally it's not required).

`this.id = id // instead of id = id`

## 51. Demo: this Reference

```
}

Student(int id, String name, String gender, int age, long phone, double gpa,
 char degree, boolean international) {
 this.id = id;
 this.name = name;
 this.gender = gender;
 this.age = age;
 this.phone = phone;
 this.gpa = gpa;
 this.degree = degree;
 this.international = international;

 studentCount = studentCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
 }

 System.out.println("\nid: " + id);
 System.out.println("nextId: " + nextId);
 System.out.println("name: " + name);
 System.out.println("gender: " + gender);
```

we can also access static variable ( and even instance method ) using this refence in similar way , but generally it's not required

## 51. Demo: this Reference

```
this.phone = phone;
this.gpa = gpa;
this.degree = degree;
this.international = international;

studentCount = studentCount + 1;
int nextId = id + 1;

if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
}

System.out.println("\nid: " + this.id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + this.name);
System.out.println("gender: " + this.gender);
System.out.println("age: " + this.age);
System.out.println("phone: " + this.phone);
System.out.println("gpa: " + this.gpa);
System.out.println("degree: " + this.degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);

}
```

Let us do the same in print statements under the constructor like below:

```
System.out.println("\nid : " + this.id); // instead of System.out.println("\nid : " + id);
```

```
BasicsDemo.java StudentBkp.java new_2

 System.out.println("age: " + age);
 System.out.println("phone: " + phone);
 System.out.println("gpa: " + gpa);
 System.out.println("degree: " + degree);
 System.out.println("tuitionFees: " + tuitionFees);
 System.out.println("studentCount: " + studentCount);

 }

 Student() {}

 public static void main(String[] args) {
 Student student1 = new Student("John", "Male", 18, 2234567890, 3.8, "B", 12000.0);
 Student student2 = new Student("Raj", "Male", 21, 2234569999, 3.4, "M", 17000.0);
 Student student3 = new Student("Anita", "Female", 20, 2234568888, 4.0, "M", 17000.0);

 System.out.println("\nNames of students are:");
 System.out.println("Name of student 1: " + student1.name);
 System.out.println("Name of student 2: " + student2.name);
 System.out.println("Name of student 3: " + student3.name);
 }
}
```

```
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
studentCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
studentCount: 2

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
studentCount: 3

Student.studentCount: 3
Name of student 1: John
Name of student 2: Raj
Name of student 3: Anita
```

Let's execute the code it runs fine and printing the name correctly(instead of null as earlier).

## 51. Demo: this Reference

```
if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
}

System.out.println("nid: " + this.id);
System.out.println("nextId: " + nextId);
System.out.println("name: " + this.name);
System.out.println("gender: " + this.gender);
System.out.println("age: " + this.age);
System.out.println("phone: " + this.phone);
System.out.println("gpa: " + this.gpa);
System.out.println("degree: " + this.degree);
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);
}

Student() {}

boolean updateProfile(String name) {
 this.name = name;
 return true;
}
```

### **Example 2 : This Reference under instance method**

Let us also use the this reference in an instance method (for updateProfile )as above.



BasicsDemo.java StudentBkp.java new\_2

```
Student() {}

boolean updateProfile(String name) {
 this.name = name;
 return true;
}

public static void main(String[] args) {
 Student student1 = new Student(1000, "Joan", "male", 18, 223_456_7890L, 3.8, 'B');

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);

 System.out.println("\nStudent.studentCount: " + Student.studentCount);

 System.out.println("\nName of student 1: " + student1.name);
 System.out.println("Name of student 2: " + student2.name);
 System.out.println("Name of student 3: " + student3.name);

 student1.updateProfile("John");
 System.out.println("\nUpdated name of student 1: " + student1.name);
}
```



Let's try to print the updated name for student1, the code works fine.

```
BasicsDemo.java StudentBkp.java new_2

Student() {}

boolean updateProfile(String name) {
 this.name = name;
 return true;
}

public static void main(String[] args) {
 Student student1 = new Student("Joan");
 Student student2 = new Student("Raj");
 Student student3 = new Student("Anita");

 System.out.println("\nNew student details:");
 System.out.println("Name of student 1: " + student1.name);
 System.out.println("Name of student 2: " + student2.name);
 System.out.println("Name of student 3: " + student3.name);

 student1.updateProfile("John");
 System.out.println("\nUpdated name of student 1: " + student1.name);
}
```

```
gender: male
age: 18
phone: 2234567890
gpa: 3.8
degree: B
tuitionFees: 12000.0
studentCount: 1

id: 1001
nextId: 1002
name: Raj
gender: male
age: 21
phone: 2234569999
gpa: 3.4
degree: M
tuitionFees: 17000.0
studentCount: 2

id: 1002
nextId: 1003
name: Anita
gender: female
age: 20
phone: 2234568888
gpa: 4.0
degree: M
tuitionFees: 17000.0
studentCount: 3

Student.studentCount: 3

Name of student 1: Joan
Name of student 2: Raj
Name of student 3: Anita
```



## 51. Demo: this Reference

```
System.out.println("tuitionFees: " + tuitionFees);
System.out.println("studentCount: " + studentCount);
}

Student() {}

boolean updateProfile(String name) {
 this.name = name;
 return true;
}

public static void main(String[] args) {
 Student student1 = new Student(1000, "Joan", "male", 18, 223_456_7890L, 3.8, 'B');

 Student student2 = new Student(1001, "Raj", "male", 21, 223_456_9999L, 3.4, 'M', true);

 Student student3 = new Student(1002, "Anita", "female", 20, 223_456_8888L, 4.0, 'M', true);

 System.out.println("\nStudent.studentCount: " + Student.studentCount);

 System.out.println("\nName of student 1: " + this.name);
 System.out.println("Name of student 2: " + student2.name);
 System.out.println("Name of student 3: " + student3.name);
```

**Example 3 : This Reference under Static Method (not possible) :**

Under static method main , let us try to use this.instanceVaraible instead of instance variable.

**Example :**

```
System.out.println ("\n Name of Student 1 : " + this.name);
// instead of System.out.println ("\n Name of Student 1 : " + student1.name);
```

The code will give us compilation error “**Non static variable this can not be referenced from Static context**”

## 51. Demo: this Reference

```
System.out.println("tui");
System.out.println("stu");
}

Student() {}

boolean updateProfile(String name) {
 this.name = name;
 return true;
}

public static void main(String[] args) {
 Student student1 = new Student();
 Student student2 = new Student();
 Student student3 = new Student();

 System.out.println("\nStudent 1 Details");
 System.out.println("Name: " + student1.name);
 System.out.println("Name: " + student2.name);
 System.out.println("Name: " + student3.name);
}
```

```
C:\javaindepth\src\com\semanticsquare\basics>javac Student.java
Student.java:70: error: non-static variable this cannot be referenced from a static context
 System.out.println("\nName of student 1: " + this.name);
 ^
1 error

C:\javaindepth\src\com\semanticsquare\basics>
```





BasicsDemo.java StudentBkp.java new\_2

```
double internationalFees = 5000.0;

Student(int id, String name, String gender, int age, long phone, double gpa,
 char degree) {
 this(id, name, gender, age, phone, gpa, degree, false);
}

Student(int id, String name, String gender, int age, long phone, double gpa,
 char degree, boolean international) {
 this.id = id;
 this.name = name;
 this.gender = gender;
 this.age = age;
 this.phone = phone;
 this.gpa = gpa;
 this.degree = degree;
 this.international = international;
 studentCount = studentCount + 1;
 int nextId = id + 1;

 if (international) {
 tuitionFees = tuitionFees + internationalFees;
 //return;
 }
}
```

Note : This reference is different then this() invocation.

- this() invocation should be the first line in the constructor
- This reference does not have any such restriction.