



Method Overloading

- If 2 or multiple methods have same name but different signatures, then those methods are called as overloaded methods.
- It's a compile time polymorphism.



Method Overloading

- E.g.

```
class Test {  
    void test() { }  
    void test(int a) { }  
    void test(int a, int b) { }  
    void test(int a, String b) { }  
    void test(String a, int b) { }  
}
```



Method Overloading

```
Test t = new Test();
```

```
t.test();
```

```
t.test(10);
```

```
t.test(10, 20);
```

```
t.test(10, "Hello");
```

```
t.test("Welcome", 20);
```

```
t.test("Hello", "Welcome"); → Error
```



Constructor

- Constructor is a special member within a class having same name as that of a class name.
- It is invoked implicitly as soon as an object is created.



Constructor

- Does not have any return type.
- Constructors are used for object initialization.



Constructor

- A constructor without any parameter is known as no-argument constructor.
- Constructors can be overloaded.

Constructor

```
class Box {  
    int length, width, height;  
    Box() {          //No-Argument  
        length = 10;  
        width = 8;  
        height = 5;  
    }  
    Box(int l, int w, int h) { //Parameterized  
        length = l;  
        width = w;  
        height = h;  
    }  
}
```



Constructor

```
Box box1 = new Box();
```

```
//Invokes no-argument constructor
```

```
Box box2 = new Box(20,15,12);
```

```
//Invokes parameterized constructor
```

```
Box box3 = new Box(12);
```

```
//Error
```




'this' reference

- Every class member function gets a hidden parameter known as `this` reference.
- `this` is a keyword in Java.
- It always refers to the object that is currently invoked.



'this' reference

```
class Box {  
    int length, width, height;  
    Box() {  
        length = 10;  
        //Is equivalent to  
        //this.length = 10;  
        ...  
    }  
}
```

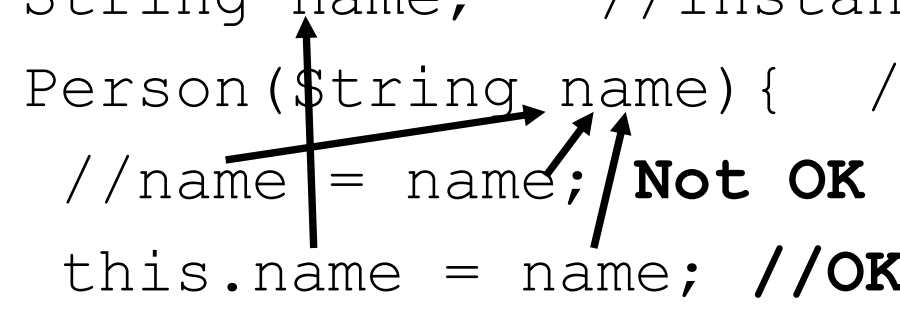


'this' reference

- It becomes mandatory to use `this` reference when local variable names conflict with instance variable names.
- It is used to resolve the scope.

'this' reference

```
class Person {  
    String name;    //Instance Variable  
    Person(String name){    //Local Variable  
        //name = name; Not OK  
        this.name = name; //OK  
    }  
}
```





Static Members

- Whatever declarations, definitions are done within a class are collectively called as members of a class e.g. variables, methods and constructors.



Static Members

- Members of a class are of 2 types:
 - Non-Static – These are associated with a particular instance of a class.
 - Static – These are not associated with any particular instance; rather they are associated with the whole class.



Static Members

- Static members are either variables or methods.
- Constructors cannot be declared `static`.



Static Members

- A static variable has a single copy irrespective of the number of objects created.
- Hence they are also known as class variables.



Static Members

```
class Person {  
    //Instance Variables  
    String name;  
    int age;  
  
    //Class Variables  
    static float avgAge;  
    static int personCount;  
    ...  
}
```



Static Members

- A `static` modifier is also used to introduce global variables.
- E.g.

```
public class Math {  
    public static float PI = 3.14f;  
    //Can be accessed from anywhere  
    //by using Math.PI  
}
```



Static Members

- Like variables, methods can also be declared as `static`.
- Can be invoked without any object.



Static Members

```
class Math {  
    static int getFactorial(int num) {  
        //Code to get factorial  
    }  
}
```

```
//int fact = Math.getFactorial(5);
```



Static Members

- Static methods can access only static members.
- `this` reference is not accessible within static methods.
- `main()` is a static method because it is required to be called without object creation as it's an entry point of the application.



Static Initialization Blocks

- An arbitrary block of code.
- Executes when the class is loaded.
- Used for initializing static variables.



Static Initialization Blocks

```
class MyClass {  
    static {  
        //Some Code  
    }  
}
```



Variables in Java

- Variables in Java are divided into 3 types:
 - Class Variables (Static Variables)
 - Instance Variables (Non-Static Variables)
 - Local Variables (Must be initialized before usage)