**Java this Keyword**

**this Keyword**

In Java, this keyword is used to refer to the current object inside a method or a constructor. For example,

class Main {

int instVar;

Main(int instVar){

this.instVar = instVar;

System.out.println("this reference = " + this);

}

public static void main(String[] args) {

Main obj = new Main(8);

System.out.println("object reference = " + obj);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

this reference = Main@23fc625e

object reference = Main@23fc625e

In the above example, we created an object named obj of the class Main. We then print the reference to the object obj and this keyword of the class.

Here, we can see that the reference of both obj and this is the same. It means this is nothing but the reference to the current object.

**Use of this Keyword**

There are various situations where this keyword is commonly used.

**Using this for Ambiguity Variable Names**

In Java, it is not allowed to declare two or more variables having the same name inside a scope (class scope or method scope). However, instance variables and parameters may have the same name. For example,

class MyClass {

// instance variable

int age;

// parameter

MyClass(int age){

age = age;

}

}

In the above program, the instance variable and the parameter have the same name: age. Here, the Java compiler is confused due to name ambiguity.

In such a situation, we use this keyword. For example,

First, let's see an example without using this keyword:

class Main {

int age;

Main(int age){

age = age;

}

public static void main(String[] args) {

Main obj = new Main(8);

System.out.println("obj.age = " + obj.age);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

obj.age = 0

In the above example, we have passed 8 as a value to the constructor. However, we are getting 0 as an output. This is because the Java compiler gets confused because of the ambiguity in names between instance the variable and the parameter.

Now, let's rewrite the above code using this keyword.

class Main {

int age;

Main(int age){

this.age = age;

}

public static void main(String[] args) {

Main obj = new Main(8);

System.out.println("obj.age = " + obj.age);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

obj.age = 8

Now, we are getting the expected output. It is because when the constructor is called, this inside the constructor is replaced by the object obj that has called the constructor. Hence the age variable is assigned value 8.

Also, if the name of the parameter and instance variable is different, the compiler automatically appends this keyword. For example, the code:

class Main {

int age;

Main(int i) {

age = i;

}

}

is equivalent to:

class Main {

int age;

Main(int i) {

this.age = i;

}

}

**this with Getters and Setters**

Another common use of this keyword is in *setters and getters methods* of a class. For example:

class Main {

String name;

// setter method

void setName( String name ) {

this.name = name;

}

// getter method

String getName(){

return this.name;

}

public static void main( String[] args ) {

Main obj = new Main();

// calling the setter and the getter method

obj.setName("Toshiba");

System.out.println("obj.name: "+obj.getName());

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

obj.name: Toshiba

Here, we have used this keyword:

* to assign value inside the setter method
* to access value inside the getter method

**Using this in Constructor Overloading**

While working with [constructor overloading](https://www.programiz.com/java-programming/constructors#overloading), we might have to invoke one constructor from another constructor. In such a case, we cannot call the constructor explicitly. Instead, we have to use this keyword.

Here, we use a different form of this keyword. That is, this(). Let's take an example,

class Complex {

private int a, b;

// constructor with 2 parameters

private Complex( int i, int j ){

this.a = i;

this.b = j;

}

// constructor with single parameter

private Complex(int i){

// invokes the constructor with 2 parameters

this(i, i);

}

// constructor with no parameter

private Complex(){

// invokes the constructor with single parameter

this(0);

}

@Override

public String toString(){

return this.a + " + " + this.b + "i";

}

public static void main( String[] args ) {

// creating object of Complex class

// calls the constructor with 2 parameters

Complex c1 = new Complex(2, 3);

// calls the constructor with a single parameter

Complex c2 = new Complex(3);

// calls the constructor with no parameters

Complex c3 = new Complex();

// print objects

System.out.println(c1);

System.out.println(c2);

System.out.println(c3);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

2 + 3i

3 + 3i

0 + 0i

In the above example, we have used this keyword,

* to call the constructor Complex(int i, int j) from the constructor Complex(int i)
* to call the constructor Complex(int i) from the constructor Complex()

Notice the line,

System.out.println(c1);

Here, when we print the object c1, the object is converted into a string. In this process, the toString() is called. Since we override the toString() method inside our class, we get the output according to that method.

One of the huge advantages of this() is to reduce the amount of duplicate code. However, we should be always careful while using this().

This is because calling constructor from another constructor adds overhead and it is a slow process. Another huge advantage of using this() is to reduce the amount of duplicate code.

**Note**: Invoking one constructor from another constructor is called explicit constructor invocation.

**Passing this as an Argument**

We can use this keyword to pass the current object as an argument to a method. For example,

class ThisExample {

// declare variables

int x;

int y;

ThisExample(int x, int y) {

// assign values of variables inside constructor

this.x = x;

this.y = y;

// value of x and y before calling add()

System.out.println("Before passing this to addTwo() method:");

System.out.println("x = " + this.x + ", y = " + this.y);

// call the add() method passing this as argument

add(this);

// value of x and y after calling add()

System.out.println("After passing this to addTwo() method:");

System.out.println("x = " + this.x + ", y = " + this.y);

}

void add(ThisExample o){

o.x += 2;

o.y += 2;

}

}

class Main {

public static void main( String[] args ) {

ThisExample obj = new ThisExample(1, -2);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Before passing this to addTwo() method:

x = 1, y = -2

After passing this to addTwo() method:

x = 3, y = 0

In the above example, inside the constructor ThisExample(), notice the line,

add(this);

Here, we are calling the add() method by passing this as an argument. Since this keyword contains the reference to the object obj of the class, we can change the value of x and y inside the add() method.