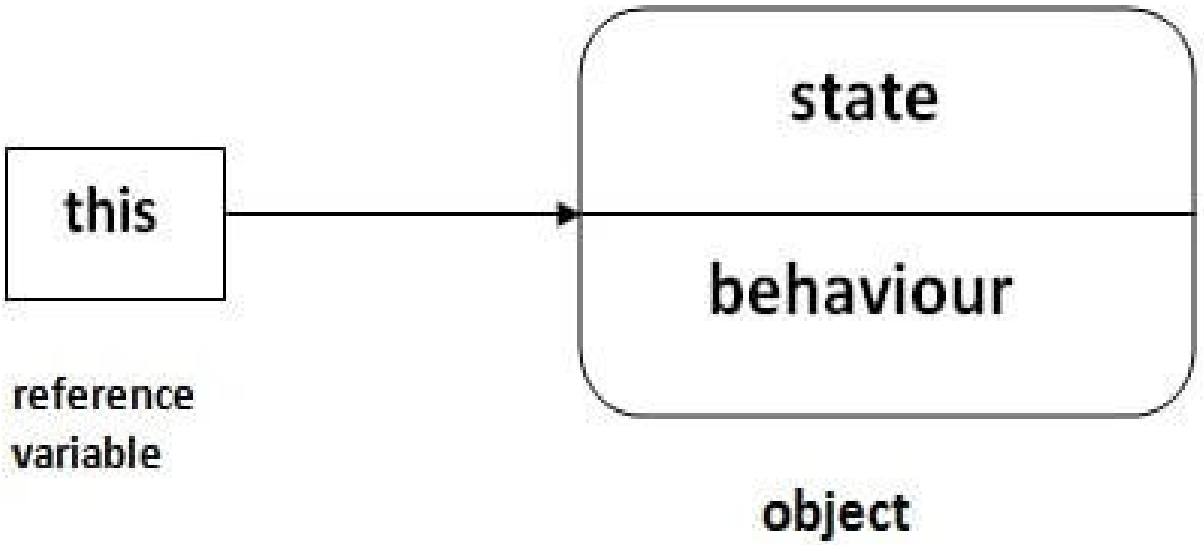
this keyword in java, polymorphism And encapsulation

Reference: Liang, Y., 2015. Intro To Java Programming. 10th ed. Pearson Education Limited.

this keyword in Java

- There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.



Usage of Java this keyword

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

1) this: to refer current class instance variable

- **this** keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
class Student{
int rollno;
String name;
double fee;
Student(int rollno,String name,double fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){
System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis{
public static void main(String[] args){
Student s1=new Student(111,"ankit",5000.0);
Student s2=new Student(112,"sumit",6000.0);
s1.display();
s2.display();
}}
```

```
0 null 0.0
0 null 0.0
```

```
class Student{
int rollno;
String name;
double fee;
Student(int rollno,String name,double fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){
System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis{
public static void main(String[] args){
Student s1=new Student(111,"ankit",5000.0);
Student s2=new Student(112,"sumit",6000.0);
s1.display();
s2.display();
}}
```

```
111 ankit 5000.0
112 sumit 6000.0
```

2) this: to invoke current class method

- You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example

```
class A{
void m(){
System.out.println("hello m");}
void n(){
System.out.println("hello n");
m();  //this.m();}
}
class TestThis{
public static void main(String[] args){
A a=new A();
a.n();
}}
```

```
hello n
hello m
```

3) this() : to invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
}
class TestThis{
public static void main(String[] args){
A a=new A(10);
}
}
```

hello a
10

```
class A{
A(){
this(10);
System.out.println("hello a");
}
A(int x){
System.out.println(x);
}
}
class TestThis{
public static void main(String[] args){
A a=new A();
}
}
```

10
hello a

```
class Student{
int rollno;
String name,course;
double fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,double fee){
this(rollno,name,course);//reusing constructor
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
class TestThis{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000.0);
s1.display();
s2.display();
}}
```

```
111 ankit java 0.0
112 sumit java 6000.0
```

4) this: to pass as an argument in the method

- The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods. Let's see the example:

```
class S2{
  void m(S2 obj){
  System.out.println("method is invoked");
  }
  void p(){
  m(this);
  }
  public static void main(String[] args){
  S2 s1 = new S2();
  s1.p();
  }
}
```

```
method is invoked
```

5) this: to pass as argument in the constructor call

- We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
class B{
 A4 obj;
 B(A4 obj){
  this.obj=obj;  }
 void display(){
  System.out.println(obj.data);//using data member of A4 class
 }
}
class A4{
 int data=10;
 A4(){
  B b=new B(this);
  b.display();
 }
 public static void main(String[] args){
  A4 a=new A4();
 }
}
```


10

Output:10
Hello java

6) this keyword can be used to return current class instance

- We can return this keyword as an statement from the method.
- In such case, return type of the method must be the class type (non-primitive). Let's see the example:

```
class A{
A getA(){
return this;
}
void msg(){System.out.println("Hello java");}
}
class Test1{
public static void main(String[] args){
new A().getA().msg();
}
}
```

Hello java

Method Overloading

- *Overloading is a language mechanism that allows two or more different methods belonging to the same class to have the **same** name as long as they have **different** argument signatures*

- The methods are said to be overloaded, and the process is referred to as method overloading.
- Method overloading is one of the ways that implements **polymorphism**.

Method Overloading

```
void print (String fileName)
 { ... // version #1
void print (int detailLevel)
{ ... // version #2
void print (int detailLevel, String fileName)
 { ... // version #3
int print (String reportTitle, int maxPages)
 { ... // version #4
boolean print ()
 { ... // version #5
```

Method Overloading

- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- Overloaded methods must differ in the type and/or number of their parameters.
- The return type alone is insufficient to distinguish two versions of a method.
- Method overloading increases the readability of the program

Implementing Method Overloading

```
class OverloadDemo {
  void test() {
    System.out.println("No parameters");
  }

  // Overload test for one integer parameter.
  void test(int a) {
    System.out.println("a: " + a);
  }

   // Overload test for two integer parameters.
   void test(int a, int b) {
     System.out.println("a and b: " + a + " " + b);
   }

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

Calling overloaded Methods

```
class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    double result;

    // call all versions of test()
    ob.test();
    ob.test(10);
    ob.test(10, 20);
    result = ob.test(123.25);
    System.out.println("Result of ob.test(123.25): " + result);
  }
}
```

Type conversion in Method Overloading

- The method matching need not always be exact.
- In some cases, Java's automatic type conversions can play a role in overload resolution.

```
class OverloadDemo {
  void test() {
    System.out.println("No parameters");
  }

  // overload test for a double parameter
  void test(double a) {
    System.out.println("Inside test(double) a: " + a);
  }
}
```

Type conversion in Method Overloading

- Java will employ its automatic type conversions only if **no** exact match is found.

```
class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    int i = 88;

    ob.test();
    ob.test(i);

    }
  }
```

Constructor Overloading

- In addition to this, constructors can also be overloaded.
- *Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.*
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.
- Constructor returns current class instance yet it does not have a return type.

Which method will be called?

```
public class Demo
{
public static void main(String[] args)
{
System.out.println(max(1,2));
}
public static double max(int num1,double num2)
{ if (num1>num2)
return num1;
else
return num2;
}
public static double max(double num1,int num2)
{ if (num1>num2)
return num1;
else
return num2;
}
}
```

```
public class Demo
{
    public static void main(String[] args)
    {
        System.out.println(max(1, 2));
    }

    public static double ma
        if (num1 > num2)
            return num1;
        else
            return num2;
    }

    public static double max(double num1, int num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }
}
```

Error!

The method max(int, double) is ambiguous for the type Demo

Press 'F2' for focus


THANK YOU